Transformée de Fourier rapide

1. Manipulation de polynômes

1.1 Représentation par les coefficients

Durant ce TD nous allons nous intéresser aux algorithmes qui manipulent les polynômes. Ces derniers seront tout d'abord représentés par le vecteur formé de la liste de leurs coefficients :

le polynôme $p(x) = \sum_{j=0}^{n-1} p_j x^j$ de degré inférieur ou égal à $n-1$ est représenté par le vecteur $\vec{p} = (p_0, p_1, \dots, p_{n-1})$.

Les trois opérations les plus communes sur les polynômes sont :

- l'*évaluation* : étant donnés un polynôme p et un nombre x, calculer le nombre $p(x)$;
- l'*addition* : étant donnés deux polynômes p et q, calculer le polynôme $p+q$;
- la *multiplication* : étant donnés deux polynômes p et q, calculer le polynôme pq.

Question 1. L'évaluation d'un polynôme utilise la règle de Horner :

$$p(x) = p_0 + x\left(\sum_{j=1}^{n-1} p_j x^{j-1}\right).$$

Rédiger la fonction Python correspondante et dénombrer le nombre d'additions et de multiplications effectuées.

Question 2. L'addition de deux polynômes de degrés inférieurs ou égaux à $n-1$ utilise la formule :

$$(p+q)(x) = \sum_{j=0}^{n-1} (p_j + q_j) x^j.$$

Rédiger la fonction Python correspondante et dénombrer le nombre d'additions effectuées.

Question 3. Le produit de deux polynômes de degrés inférieurs ou égaux à $n-1$ utilise la formule :

$$(pq)(x) = \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} (p_j q_k) x^{j+k}.$$

Rédiger la fonction Python correspondante et dénombrer le nombre d'additions et de multiplications effectuées.

1.2 Représentation par échantillonnage

L'étude de l'interpolation de Lagrange prouve qu'étant donnés n couples $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$ (les x_j étant deux à deux distincts) il existe un unique polynôme p de degré inférieur ou égal à $n-1$ tel que pour tout $j \in [\![0, n-1]\!]$, $p(x_j) = y_j$. Les coefficients x_j étant fixés, il est donc possible de représenter le polynôme p par le vecteur $\vec{y} = (y_0, y_1, \dots, y_{n-1})$.

Additionner deux polynômes suivant cette représentation est simple : il suffit d'additionner terme à terme les éléments des deux tableaux. Pour les multiplier il suffit là encore de faire le produit terme à terme des éléments des deux tableaux, à condition cependant de posséder un échantillonnage de cardinal $2n$.
Malheureusement, l'évaluation d'un polynôme représenté par échantillonnage est plus délicat et nécessite de passer par la formule de Lagrange :

$$p(x) = \sum_{j=0}^{n-1} y_j l_j(x) \qquad \text{avec} \qquad l_j(x) = \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}.$$

Pour chaque valeur de j le calcul de $l_j(x)$ est linéaire donc l'algorithme est de coût quadratique.

On peut résumer le coût de chacune des trois opérations pour les deux représentations des polynômes dans le tableau suivant :

	évaluation	addition	multiplication
représentation par coefficients	$O(n)$	$O(n)$	$O(n^2)$
représentation par échantillonnage	$O(n^2)$	$O(n)$	$O(n)$

Comme on peut le constater aucune des deux représentations n'est idéale, et chacune a son point fort et son point faible. L'idéal serait de disposer d'un moyen rapide de passer d'une représentation à une autre de façon à pouvoir exploiter les points forts de chacune d'entre elles.

Question 4. Si on utilise un algorithme naïf, quels est le coût requis pour passer de la représentation par coefficients à une représentation par échantillonnage ?

Question 5. Montrer que la conversion réciproque se ramène à la résolution d'un système linéaire $V\vec{p} = \vec{y}$, où $V \in \mathcal{GL}_n(\mathbb{K})$. Si on applique la méthode du pivot de Gauss, quel coût total obtient-on pour cette conversion ? Pourquoi peut-on réduire ce coût à un $O(n^2)$?

2. Transformée de Fourier rapide

Les deux questions précédentes montrent qu'il n'y a aucun intérêt à passer d'une représentation à une autre en utilisant les algorithmes naïfs : ceux-ci sont d'un coût trop important pour améliorer le coût d'un produit ou d'une évaluation dans les cas où ceux-ci sont quadratiques. Il y a heureusement un degré de liberté que nous n'avons pas encore exploité : le choix des valeurs x_j. Nous allons voir que couplé à une approche récursive, le bon choix de l'échantillonnage va permettre une conversion bien plus rapide.

2.1 Diviser pour régner

Lorsque n est un entier pair, tout polynôme de degré inférieur ou égal à n peut s'exprimer à l'aide de deux polynômes de degrés inférieurs ou égaux à $n/2$ par le biais de la formule :

$$p(x) = q(x^2) + xr(x^2),$$

les coefficients de q étant les coefficients d'indices pairs du polynôme p et ceux de r, ceux d'indices impairs.

On dit qu'un ensemble X de n nombres est *contractant* lorsqu'il possède une des deux conditions suivantes :

- $n = 1$;
- l'ensemble $X^2 = \left\{x^2 \;\middle|\; x \in X\right\}$ est de cardinal $n/2$ et est contractant.

Étant donné un polynôme p de degré inférieur ou égal à $n-1$ et un ensemble contractant X de cardinal n, on peut calculer l'ensemble $\left\{p(x) \;\middle|\; x \in X\right\}$ en suivant l'algorithme récursif :

1. on calcule récursivement l'ensemble $\left\{q(x^2) \;\middle|\; x \in X\right\} = \left\{q(y) \;\middle|\; y \in X^2\right\}$;
2. on calcule récursivement l'ensemble $\left\{r(x^2) \;\middle|\; x \in X\right\} = \left\{r(y) \;\middle|\; y \in X^2\right\}$;
3. pour tout $x \in X$ on calcule $p(x) = q(x^2) + xr(x^2)$.

Question 6. Établir une relation de récurrence vérifiée par le coût $C(n)$ de cet algorithme, et en déduire que $C(n) = O(n \log n)$.

Il reste donc à choisir un ensemble contractant de cardinal n pour obtenir un algorithme rapide de conversion de la représentation d'un polynôme par ses coefficients à sa représentation par échantillonnage.

Question 7. Montrer que X est un ensemble contractant de cardinal n si et seulement si n est une puissance de 2 et s'il existe $z \in \mathbb{C}^*$ tel que X soit l'ensemble des racines n-ième de z.

2.2 Transformée de Fourier discrète

Nous allons désormais noter $X_n = \left\{\omega_n^k \;\middle|\; k \in [\![0, n-1]\!]\right\}$ l'ensemble des racines n-ième de l'unité, avec $\omega_n = e^{2i\pi/n}$. Nous venons de prouver que les ensembles X_1, X_2, X_4, X_8, … sont des ensembles contractants.
Étant donné un polynôme p de degré inférieur ou égal à $n-1$ (n étant une puissance de 2), on appelle *transformée de Fourier discrète* (ou DFT pour *Discrete Fourier Transform*) du vecteur $\vec{p} = (p_0, p_1, \ldots, p_{n-1})$ le vecteur $\vec{p}^{\,*} = (p_0^*, p_1^*, \ldots, p_{n-1}^*)$ défini par :

$$\forall j \in [\![0, n-1]\!], \quad p_j^* = p(\omega_n^j) = \sum_{k=0}^{n-1} p_k \omega_n^{jk}.$$

Le vecteur $\vec{p}^{\,*}$ est donc la représentation du polynôme p lorsqu'on l'échantillonne sur X_n.
L'algorithme de calcul de $\vec{p}^{\,*}$ à partir de $\vec{p}$ en $O(n \log n)$ que nous venons de décrire s'appelle l'algorithme de *transformée de Fourier rapide* (ou FFT pour *Fast Fourier Transform*) ; il a été découvert par Cooley et Tukey en 1965.

Question 8. Rédiger en Python l'algorithme FFT sous forme récursive.

> Le module `cmath` permet de manipuler les nombres complexes en Python. En mémoire, un nombre complexe z est représenté par le couple (x, y) formé de ses parties réelle et imaginaire (qu'on peut obtenir à l'aide des méthodes `real` et `imag`) et sera affiché sous la forme `x + yj` (ainsi pour obtenir le nombre complexe i il faut écrire `1j`). En d'autres termes, `z == z.real + z.imag * 1j`.
> On dispose (entre autre) des fonctions suivantes :
>
> - `abs(z)` calcule le module de z ;
> - `phase(z)` calcule un argument (dans $]-\pi, \pi]$) de z ;
> - `polar(z)` retourne la paire (r, θ) du module et le l'argument de z ;
> - `rect(r, theta)` est l'application réciproque de `polar`.
>
> La plupart des fonctions réelles présentes dans le module `math` possèdent leur équivalent complexe dans `cmath` : `exp`, `sqrt`, `pi`, `cos`, `sin`, …, et les opérations algébriques sont bien entendu possibles suivant la même syntaxe que pour le type *float*.

Figure 1 – Les nombres complexes en Python.

2.3 Inverser la FFT

Nous avons aussi besoin de la fonction qui retrouve la représentation par coefficients d'un polynôme à partir de sa DFT. Nous avons vu à la question 5 que les vecteurs $\vec{p}$ et $\vec{p}^{\,*}$ sont liés par la relation linéaire : $\vec{p}^{\,*} = V\vec{p}$, où $V = (v_{jk})$ est la matrice de Vandermonde définie par $v_{jk} = \omega_n^{jk}$.

Question 9. Montrer que $V^{-1} = \frac{1}{n}\overline{V}$, où $\overline{V}$ désigne la matrice conjuguée de la matrice V.

Question 10. En déduire l'algorithme inverse de la FFT, que l'on rédigera là encore en Python.

2.4 Multiplication rapide de deux polynômes

Nous disposons désormais de deux algorithmes en $O(n \log n)$ permettant la conversion d'une représentation vers l'autre.

Question 11. Rédiger un algorithme effectuant la multiplication rapide de deux polynômes p et q représentés par leurs coefficients.

♦ ♦
♦