INF431

La transformée de Fourier rapide
Sujet proposé par Bruno Salvy
Version: 2063:2113M

Introduction

La transformée de Fourier discrète joue un rôle central en traitement du signal et notamment dans les mécanismes de compression avec perte en MP3 et en JPEG. Dans une version simplifiée, un signal (par exemple du son) est d'abord discrétisé en temps ; chaque (petite) tranche de temps fournit n valeurs $(a_0, \ldots, a_{n-1})$ qui sont alors converties en n coefficients $(b_0, \ldots, b_{n-1})$ exprimant le même signal discret, mais dans le domaine fréquentiel. La compression est obtenue en enlevant les coefficients correspondant aux fréquences moins audibles (trop hautes ou trop basses).

Les coefficients $(b_0, \ldots, b_{n-1})$ sont obtenus par l'application :

$$\text{DFT} : (a_0, \ldots, a_{n-1}) \mapsto (A(1), A(\omega), \ldots, A(\omega^{n-1})), \quad \text{où} \quad A(X) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1},$$

et $\omega \in \mathbb{C}$ est une racine $n$ième de l'unité ($\omega^n = 1$), et de surcroît primitive ($\omega^\ell \neq 1$ pour $1 \leq \ell < n$). Le calcul rapide de la transformée de Fourier discrète (DFT) fournit une illustration des techniques de diviser-pour-régner qui mène à l'algorithme de transformée de Fourier rapide (souvent abrégé FFT pour *fast Fourier transform*) [1].

Cet algorithme est par ailleurs au cœur de l'algorithmique rapide du calcul sur les entiers, les polynômes et les séries et nous en fournirons quelques exemples (multiplication et, si le temps le permet, division et applications). Dans tout ce sujet, il sera commode de considérer le n-uplet $(a_0, \ldots, a_{n-1})$ comme une structure de données pour représenter le polynôme $A(X)$ et calculer un polynôme signifie en calculer les coefficients.

1 L'algorithme de FFT

Question 1 Écrire le pseudo-code d'un algorithme naïf de calcul de DFT et estimer sa complexité. ◇

L'observation suivante fournit le point de départ de la récursion qui mène à l'algorithme rapide.

Question 2 Montrer que si $n = 2m$, alors ω^k est une racine du polynôme $X^m + 1$ lorsque k est impair et du polynôme $X^m - 1$ sinon. ◇

Pour un polynôme $A(X) = a_0 + a_1 X + \cdots$ de degré inférieur à $2m$, on définit alors des polynômes $R_0(X)$ et $R_1(X)$ de degrés inférieurs à m par divisions euclidiennes :

$$A(X) = Q_0(X)(X^m - 1) + R_0(X), \quad A(X) = Q_1(X)(X^m + 1) + R_1(X).$$

Question 3 Montrer que si k est pair, alors $A(\omega^k) = R_0(\omega^k)$ et que si k est impair, $A(\omega^k) = R_1(\omega^k)$.◇

Question 4 Montrer que les polynômes R_0 et R_1 sont donnés par les formules

$$R_0 = (a_0 + a_m) + (a_1 + a_{m+1})X + \cdots, \quad R_1 = (a_0 - a_m) + (a_1 - a_{m+1})X + \cdots.$$

En déduire que le calcul du polynôme R_0 s'effectue en au plus m opérations, et que, les puissances de ω étant données, celui du polynôme $\overline{R}_1(X) = R_1(\omega X)$ s'effectue en au plus $2m$ opérations. ◇

1. Cet algorithme, connu de Gauss et redécouvert par Cooley & Tuckey en 1965, figure dans la liste des *«Top 10 Algorithms of the 20th century»* à l'url http://www.siam.org/pdf/news/637.pdf

Question 5 En déduire le pseudo-code d'un algorithme de calcul récursif de la transformée de Fourier discrète dans le cas où n est une puissance de 2. Estimer sa complexité. $\diamond$

Un aspect important que nous ne développons pas ici mais qui explique le succès de la FFT est qu'outre son efficacité, elle présente une excellente stabilité numérique : un calcul avec des coefficients flottants ne développe pas d'erreurs excessives.

2 Transformée de Fourier inverse et produit de polynômes

Question 6 Montrer que l'application qui associe au polynôme $A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1}$ les valeurs $(A(1), \ldots, A(\omega^{n-1}))$ est linéaire et indiquer sa matrice V_ω dans la base $\{1, X, \ldots, X^{n-1}\}$.$\diamond$

Question 7 Montrer que lorsque ω est une racine de l'unité, alors $V_{\omega^{-1}}V_\omega = nI_n$. $\diamond$

Question 8 En déduire que l'on peut reconstruire un polynôme de degré inférieur à n à partir de ses valeurs sur $1, \ldots, \omega^{n-1}$ (cette opération s'appelle une *interpolation*) et que ce calcul peut être effectué en $O(n \log n)$ opérations. $\diamond$

Le principe du produit rapide de polynômes est alors de calculer d'abord l'évaluation des deux polynômes, puis de multiplier les valeurs deux à deux, et enfin d'interpoler le résultat.

Question 9 Écrire le pseudo-code d'un algorithme calculant le produit de deux polynômes de $\mathbb{C}[X]$ de somme des degrés au plus n en $O(n \log n)$ opérations dans $\mathbb{C}$. $\diamond$

Cet algorithme forme la base de toute l'algorithmique rapide du calcul formel, avec des applications importantes en cryptographie ou en théorie des codes. En pratique, dans les bonnes implantations, plusieurs algorithmes de multiplication sont utilisés, avec des seuils automatiques choisissant l'algorithme le plus adapté en fonction du degré et du processeur. Pour les petits degrés, on utilise la multiplication naïve (parfois codée en assembleur), puis l'algorithme de Karatsuba et enfin la FFT. Les degrés typiques (par exemple pour la bibliothèque NTL) où l'algorithme de Karatsuba commence à être utilisé sont de l'ordre de la vingtaine, alors que la FFT est l'algorithme standard pour des polynômes dont le degré dépasse la centaine. De tels degrés apparaissent très couramment en cryptographie.

Les mêmes idées (Karatsuba, FFT) s'appliquent aussi au produit d'entiers, mais avec des complications pour la gestion des retenues.

3 Compléments : division et applications

Nous avons déjà obtenu que la multiplication n'est pas beaucoup plus chère que l'addition, et nous allons voir qu'il en va de même pour la division. Là encore, les idées sont à peu près les mêmes pour les entiers et les polynômes, mais plus faciles à décrire sur les polynômes, et encore plus sur les séries tronquées, sur lesquelles nous allons nous concentrer. Le n-uplet $(a_0, \ldots, a_{n-1})$ représente donc maintenant la série

$$A(X) = a_0 + a_1X + \cdots + a_{n-1}X^{n-1} + O(X^n),$$

et pour $k \leq n$, on notera $A(X) \bmod X^k$ le polynôme $a_0 + a_1X + \cdots + a_{k-1}X^{k-1}$ obtenu en tronquant la série. Les opérations d'addition et de multiplication s'obtiennent comme auparavant (pour la multiplication, on doit tronquer le produit des polynômes $A(X) \bmod X^n$ et $B(X) \bmod X^n$).

La multiplication par la série tronquée $A(X)$ est une application linéaire, qui est inversible si $a_0 \neq 0$. L'inverse de $A(X)$ est alors l'unique série tronquée $B(X) = b_0 + \cdots + b_{n-1}X^{n-1} + O(X^n)$ telle que $A(X)B(X) = 1 + O(X^n)$. Multiplier une série par l'inverse d'une autre fournit une division que l'on appelle *division selon les puissances croissantes* (qui est utilisée pour le calcul de la décomposition en éléments simples d'une fraction rationnelle).

3.1 Approche classique

Question 10 Écrire le pseudo-code de la méthode classique pour effectuer la division selon les puissances croissantes «à la main», et estimer son nombre d'opérations arithmétiques dans $\mathbb{C}$ en fonction de n. ◇

3.2 Itération de Newton et diviser-pour-régner

Le principe de la division rapide est de calculer un développement tronqué de l'inverse du dénominateur et de le multiplier par le numérateur. Nous commençons par étudier le calcul d'inverse rapide.

Question 11 Montrer que le polynôme obtenu par l'itération de Newton [2]

$$Y_0 = 1/a_0, \qquad Y_{k+1} = Y_k + Y_k(1 - AY_k) \mod X^{2^{k+1}} \quad (k > 0)$$

vérifie $A(X)Y_k(X) = 1 + O(X^{\min(2^k,n)})$. ◇

Autrement dit, l'itération de Newton se traduit naturellement en un diviser-pour-régner, où il suffit de résoudre le problème à précision moitié, une seule fois.

Question 12 Écrire le pseudo-code d'une procédure prenant en entrée une série tronquée de terme constant non-nul et renvoyant son inverse. Estimer sa complexité. ◇

Pour parvenir à la division euclidienne des polynômes, il suffit d'effectuer ce calcul «à l'infini», c'est-à-dire en changeant X en $1/X$ et en faisant attention aux degrés. C'est facile, mais technique, et nous ne le traiterons pas. Au final, on peut effectuer la division euclidienne pour le prix d'environ deux multiplications seulement.

3.3 Logarithme et exponentielle

Avec cette division, il devient facile de calculer le développement en série d'un logarithme et d'une exponentielle. On rappelle les formules suivantes, où maintenant $A(X)$ désigne une série tronquée sans terme constant $a_1X + \cdots + a_{n-1}X^{n-1} + O(X^n)$ (c'est-à-dire que $a_0 = 0$ dans le n-uplet de coefficients) :

$$\log(1 + A(X)) = A(X) - \frac{1}{2}A(X)^2 + \frac{1}{3}A(X)^3 + \cdots + O(X^n),$$
$$\exp(A(X)) = 1 + A(X) + \frac{1}{2!}A(X)^2 + \frac{1}{3!}A(X)^3 + \cdots + O(X^n).$$

Question 13 Estimer la complexité d'une utilisation directe de ces formules. ◇

Question 14 Montrer que l'écriture $\log(1 + A) = \int A'/(1 + A)$ permet un calcul du logarithme en seulement $O(n \log n)$ opérations. ◇

Le calcul de l'exponentielle peut alors s'appuyer sur celui du logarithme, grâce à la relation facile $\log(\exp(A(X))) = A(X) + O(X^n)$. Il suffit, à nouveau, d'utiliser une itération de Newton pour résoudre l'équation $\log(Y) - A = 0$, ce qui mène à l'itération

$$Y_{k+1} = Y_k - Y_k(\log Y_k - A) \bmod X^{\min(2^{k+1},n)}, \qquad Y_0 = 1.$$

Question 15 En admettant d'abord que $Y_k(X) - \exp(A(X)) = O(X^{\min(2^k,n)})$, montrer que l'exponentielle d'une série tronquée peut se calculer en seulement $O(n \log n)$ opérations, puis, s'il reste du temps, prouver cette relation. ◇

2. Pour calculer l'inverse d'un nombre a, l'itération de Newton menant à la solution de l'équation $a - 1/y = 0$ s'écrit $y_{n+1} = y_n + y_n(1 - ay_n)$. La même itération est appliquée ici à des polynômes et mène à une bonne complexité.

Références

[1] Joachim von zur Gathen et Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 2nd édition, 2003.

[2] Jon Kleinberg et Éva Tardos. *Algorithm Design*. Addison Wesley, 2005.