COURS 5

Calculs modulaires, évaluation et interpolation

Résumé

Nous introduisons la notion d'algorithme modulaire. Nous nous intéressons aux cas particuliers importants de l'évaluation multipoint et de l'interpolation, pour lesquels nous donnons des algorithmes rapides. Ces algorithmes peuvent être vus comme des généralisations de la FFT.

1. Introduction

Un ingrédient essentiel en calcul formel est l'utilisation de différents types de représentations pour les objets manipulés. Par exemple, un polynôme est classiquement codé par la liste de ses coefficients, mais on peut très bien le représenter par les valeurs qu'il prend en un nombre suffisant de points. D'ailleurs, nous avons vu au Cours 2 que c'est bien cette seconde représentation qui est la plus adaptée pour le calcul efficace, via la FFT, du produit de polynômes. Par ailleurs, d'autres opérations (comme la division polynomiale, traitée dans le Cours 3) se font mieux dans la première représentation. D'autres exemples de codages alternatifs déjà rencontrés dans ce cours sont l'écriture des entiers en différentes bases de numération, ou encore la représentation des suites récurrentes linéaires à coefficients constants, soit par leurs éléments, soit par leurs séries génératrices, soit par une récurrence et des conditions initiales.

L'exemple de la FFT montre à quel point il est crucial d'examiner dans quelle représentation un problème donné est plus facile à traiter, et aussi, de trouver des algorithmes rapides pour la conversion d'une représentation à l'autre.

Une incarnation de ce concept général est la notion d'*algorithme modulaire*, dont le paradigme *évaluation-interpolation* est un cas particulier très important. L'approche modulaire consiste à choisir des *moduli* m_i, à faire des calculs modulo chaque m_i et à reconstruire tout à la fin le résultat à l'aide d'une version effective du *théorème des restes chinois*. Pour assurer l'unicité du résultat et la correction de ce schéma, il faut que les moduli soient suffisamment *nombreux et indépendants*. Typiquement, s'il s'agit d'entiers, ils doivent être choisis premiers entre eux et tels que leur produit dépasse le résultat final. En particulier, pour mettre en place une approche modulaire, on a besoin de bornes *a priori* sur la taille de l'objet calculé.

Cette approche porte ses fruits surtout lorsque les coefficients dans les calculs intermédiaires sont beaucoup plus gros que ceux du résultat final. Il s'agit du phénomène de *l'explosion des expressions intermédiaires*, qui se présente par exemple dans le calcul du déterminant d'une matrice entière ou polynomiale, ou encore au cours du calcul du pgcd de polynômes à coefficients entiers.

Exemple 1 (calcul du déterminant d'une matrice polynomiale). Soit à calculer le déterminant d'une matrice $n \times n$, aux entrées polynômes de degrés au plus d (le cas particulier $d = 1$ correspond au calcul d'un polynôme caractéristique). Le point de départ est la remarque que le pivot de Gauss produit sur la i^{e} ligne des fractions rationnelles de degré $2^i d$. Ainsi, sa complexité ne semble pas polynomiale par rapport à n. Une approche évaluation-interpolation résout cette anomalie. Le déterminant étant un polynôme de degré au plus nd, il suffit de choisir un ensemble de $nd+1$ points, d'y évaluer les entrées de la matrice, de calculer les $nd+1$ déterminants de matrices scalaires et de finir par une interpolation fournissant le determinant cherché. Le même raisonnement s'applique aux matrices entières.

Choix des moduli. Dans certaines situations, le programmeur a le choix des moduli. Dans l'exemple précédent, on peut choisir comme moduli des polynômes de la forme $X - \omega^i$, dans le cas favorable où l'anneau de base contient une racine primitive de l'unité ω d'ordre suffisamment élevé ($\approx 2dn$). En théorie ce choix est donc possible dès lors que l'anneau de base permet une multiplication polynomiale à base de FFT. En pratique, il existe des plages de degrés pour lesquels, même si elle est faisable, la FFT n'est pas encore rentable et pour lesquels ce choix est déconseillé. Dans d'autres situations, le choix des moduli est intrinsèquement imposé par le problème à traiter ; c'est le cas pour le calcul de la factorielle exposé au Cours 6.

2. Présentation, résultats

Les problèmes d'évaluation et d'interpolation sont inverses l'un de l'autre. Dans leur version standard, ils s'énoncent ainsi. Soient $a_0, \ldots, a_{n-1}$ des points dans $\mathbb{A}$, où $\mathbb{A}$ est un anneau commutatif et unitaire.

Évaluation. Étant donné un polynôme P dans $\mathbb{A}[X]$, de degré strictement inférieur à n, calculer les valeurs :

$$P(a_0), \ldots, P(a_{n-1}).$$

Interpolation. Étant donnés $b_0, \ldots, b_{n-1} \in \mathbb{A}$, trouver un polynôme $P \in \mathbb{A}[X]$ de degré strictement inférieur à n tel que

$$P(a_0) = b_0, \ldots, P(a_{n-1}) = b_{n-1}.$$

Le problème d'interpolation admet toujours une solution unique sous l'hypothèse technique suivante :

(H) $\quad a_i - a_j$ est inversible dans $\mathbb{A}$ lorsque $i \neq j$.

En effet, si $\mathbb{V} = (a_{j-1}^{i-1})$ est la matrice de Vandermonde associée aux points a_i, dont le déterminant vaut $\det(\mathbb{V}) = \prod_{i<j} (a_i - a_j)$, alors le problème d'interpolation se traduit par la résolution en $\mathbf{p}$ du système linéaire $\mathbb{V}\mathbf{p} = \mathbf{b}$, où $\mathbf{b}$ est le vecteur des b_i. Or, ce système admet une solution unique, puisque l'hypothèse **(H)** entraîne l'inversibilité du déterminant $\det(\mathbb{V})$ et donc aussi celle de la matrice $\mathbb{V}$.

Exercice 1. Montrer que $\det(\mathbb{V}) = \prod_{i<j}(a_i - a_j)$.

Cette discussion suggère un algorithme naïf pour l'interpolation, produisant l'unique solution $\mathbf{p} = \mathbb{V}^{-1}\mathbf{b}$ du système $\mathbb{V}\mathbf{p} = \mathbf{b}$ à l'aide du pivot de Gauss, en $O(n^3)$ opérations dans $\mathbb{A}$. De manière analogue, l'évaluation multipoint de P en

les a_i se traduit par le produit matrice-vecteur $\mathbb{V}\mathbf{p}$, où $\mathbf{p}$ est le vecteur des coefficients de P ; elle peut donc être effectuée de manière naïve en $O(n^2)$ opérations arithmétiques. En s'y prenant *vraiment naïvement*, l'interpolation est donc plus couteuse que l'évaluation multipoint. Nous verrons un peu plus loin qu'une méthode exploitant *la formule d'interpolation de Lagrange* permet de résoudre le problème d'interpolation en complexité *quadratique* en n.

L'objectif de ce cours est de montrer qu'il est possible d'atteindre un complexité quasi-optimale, tant pour l'évaluation multipoint que pour l'interpolation, en utilisant des algorithmes de type « diviser pour régner » qui ramènent ces questions à des multiplications de polynômes. Les principaux résultats sont les suivants :

THÉORÈME 1. *On peut effectuer l'évaluation et l'interpolation sur n points en utilisant $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ de $\mathbb{A}$. Cette complexité peut être abaissée à $O(\mathsf{M}(n))$ si les points sont en progression géométrique.*

En termes pratiques, si l'anneau de base $\mathbb{A}$ est un corps fini, les algorithmes rapides deviennent avantageux pour des degrés de l'ordre de quelques dizaines : c'est sensiblement mieux que pour les algorithmes rapides de type « diviser pour régner » utilisés dans le Cours 10 pour le calcul de pgcd ou d'approximants de Padé-Hermite rapide ; cela reflète une relative simplicité des algorithmes.

Extensions. L'évaluation multipoint et l'interpolation sont des instances particulières des problèmes *modulos multiples* et *restes chinois* s'énonçant comme suit :

Modulos multiples. Étant donné un polynôme P dans $\mathbb{A}[X]$, de degré strictement inférieur à $\sum_i \deg m_i$, calculer les restes :

$$P \bmod m_1, \ldots, P \bmod m_r.$$

Restes Chinois. Étant donnés des polynômes $b_1, \ldots, b_r, m_1, \ldots, m_r$ dans $\mathbb{A}[X]$, avec m_i unitaires, retrouver le polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que

$$P \bmod m_1 = b_1, \ldots, P \bmod m_r = b_r.$$

Les techniques de ce cours s'y généralisent[1] et mènent aux résultats suivants :

THÉORÈME 2. *On peut résoudre les deux problèmes ci-dessus en $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ dans $\mathbb{A}$.*

Naturellement, on peut se poser les questions précédentes dans le cas où l'anneau de polynômes $\mathbb{A}[X]$ est remplacé par l'anneau $\mathbb{Z}$, les polynômes m_i sont remplacés par des moduli entiers a_i et où l'on cherche un P entier à n chiffres. Il est possible d'obtenir le même type de résultats de complexité, cette fois en comptant les opérations binaires, mais nous nous limiterons dans la suite au cas polynomial.

La suite de ce cours est organisée comme suit : dans la Section 3 nous décrivons la méthode d'interpolation de Lagrange, de complexité quadratique. La Section 4 est consacrée aux algorithmes rapides sous-jacents à la preuve du Théorème 1.

[1] Pour les restes chinois, l'unicité du résultat est garantie par l'hypothèse que le résultant $\mathrm{Res}(m_i, m_j)$ est un élément inversible dans $\mathbb{A}$, pour tous $i \neq j$. Cette condition technique généralise la condition d'inversibilité des $a_i - a_j$; se rapporter au Cours 10 pour la définition du résultant.

3. Interpolation de Lagrange

Un algorithme de complexité quadratique pour l'interpolation polynomiale repose sur une écriture explicite du polynôme interpolant. Soient b_i les valeurs à interpoler. On peut alors écrire P sous la forme :

$$P(X) \;=\; \sum_{i=0}^{n-1} b_i \prod_{0\le j\le n-1}^{j\neq i} \frac{X-a_j}{a_i-a_j}.$$

Cette égalité est usuellement appelée la *formule d'interpolation de Lagrange*. Pour la prouver, il suffit d'observer que pour tout i, le produit s'annule en a_j $(j \neq i)$, et vaut 1 en a_i. Afin de simplifier l'écriture de la formule de Lagrange, posons

$$A = \prod_j (X-a_j) \quad \text{et} \quad A_i = \prod_{j\neq i}(X-a_j) = \frac{A}{X-a_i},$$

pour $i = 0, \ldots, n-1$. Pour $i \neq j$, on a donc les relations $A(a_i) = 0$ et $A_i(a_j) = 0$. De plus, $A_i(a_i)$ est inversible sous l'hypothèse **(H)**. On obtient alors l'écriture

$$P \;=\; \sum_{i=0}^{n-1} b_i \frac{A_i(X)}{A_i(a_i)}.$$

Une approche directe pour l'interpolation revient à calculer tout d'abord les polynômes A_i, puis leurs valeurs en les points a_i, et enfin à faire les combinaisons linéaires nécessaires. Le seul point non-trivial est le calcul des A_i : si on n'y fait pas attention, on risque de sortir des bornes en $O(n^2)$ (par exemple, si on les calcule tous indépendamment, et chacun de manière quadratique). Pour faire mieux, on partage le gros des calculs, en calculant d'abord le polynôme A.

Interpolation de Lagrange

Entrée : $a_0, \ldots, a_{n-1} \in \mathbb{A}$ vérifiant **(H)** et $b_0, \ldots, b_{n-1}$ dans $\mathbb{A}$.

Sortie : L'unique polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que $P(a_i) = b_i$ pour tout i.

1. $A \leftarrow 1$, $P \leftarrow 0$
2. Pour $i = 0, \ldots, n-1$ faire
   $A \leftarrow A \cdot (X - a_i)$
3. Pour $i = 0, \ldots, n-1$ faire
   $A_i \leftarrow A/(X-a_i)$
   $q_i \leftarrow A_i(a_i)$
   $P \leftarrow P + b_i A_i / q_i$

PROPOSITION 1. *L'algorithme ci-dessus utilise $O(n^2)$ opérations dans $\mathbb{A}$.*

DÉMONSTRATION. La multiplication d'un polynôme de degré d par un polynôme de degré 1 prend un temps linéaire en d, disons Cd opérations, C étant une constante. Calculer A demande donc $C(1 + 2 + \cdots + (n-1)) = O(n^2)$ opérations. Ensuite, chaque passage dans la seconde boucle prend un temps linéaire en n. Il y a n passages, d'où à nouveau du $O(n^2)$. □

4. Algorithmes rapides

4.1. Les idées. L'idée fondamentale est à nouveau d'utiliser une technique de type « diviser pour régner ». Supposons pour simplifier que n est pair et séparons l'ensemble des points $a_0, \dots, a_{n-1}$ en deux paquets, $a_0, \dots, a_{n/2-1}$ et $a_{n/2}, \dots, a_{n-1}$. Il est naturel de *scinder* également P en deux polynômes P_0 et P_1 de degré strictement inférieur à $n/2$, tels que :

$$P_0(a_0) = P(a_0), \dots, P_0(a_{n/2-1}) = P(a_{n/2-1}) \quad \text{et}$$
$$P_1(a_{n/2}) = P(a_{n/2}), \dots, P_1(a_{n-1}) = P(a_{n-1}).$$

Pour effectuer ce scindage, le choix suivant s'impose :

$$P_0 = P \bmod (X - a_0) \cdots (X - a_{n/2-1})$$
$$P_1 = P \bmod (X - a_{n/2}) \cdots (X - a_{n-1}).$$

On a donc ramené le calcul en degré n à deux calculs en degré $n/2$, plus deux divisions euclidiennes ; on va ensuite itérer ce processus. En bout de course, on retrouve les valeurs $P(a_i)$, car celles-ci peuvent également s'écrire $P \bmod (X - a_i)$.

Remarquons de suite que les polynômes par lesquels on effectue les divisions euclidiennes ne sont pas « gratuits » : on a besoin de les calculer. Pour cela, l'idée est de les empiler dans une structure d'arbre binaire, dont les nœuds internes sont étiquetés par des polynômes.

4.2. L'arbre des sous-produits. Pour simplifier, dans tout ce qui suit, on suppose que le nombre de points est une puissance de 2, $n = 2^\kappa$. Quand ce n'est pas le cas, on adapte toutes les constructions (on fabrique toujours un arbre binaire, mais il lui manque alors des nœuds internes à certains niveaux). On peut donner la définition récursive suivante pour cet arbre (noté $\mathcal{B}$).

- Si $\kappa = 0$, $\mathcal{B}$ se réduit à un nœud unique qui contient le polynôme $X - a_0$.
- Si $\kappa > 0$, soient $\mathcal{B}_0, \mathcal{B}_1$ les arbres associés à $a_0, \dots, a_{2^{\kappa-1}-1}$ et $a_{2^{\kappa-1}}, \dots, a_{2^\kappa-1}$. Soient M_0 et M_1 les polynômes présents aux racines de $\mathcal{B}_0$ et $\mathcal{B}_1$. Alors $\mathcal{B}$ est l'arbre dont la racine contient le produit $M_0 M_1$ et dont les fils sont $\mathcal{B}_0$ et $\mathcal{B}_1$.

Graphiquement, on a la représentation suivante :

$$A = \textstyle\prod_{i=0}^{n-1}(X - a_i)$$

$$B_0 = \textstyle\prod_{i=0}^{n/2-1}(X - a_i) \qquad\qquad B_1 = \textstyle\prod_{i=n/2}^{n-1}(X - a_i)$$

$$\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots$$

$$(X - a_0)(X - a_1) \qquad\qquad (X - a_{n-2})(X - a_{n-1})$$

$$X - a_0 \qquad X - a_1 \qquad \cdots \qquad X - a_{n-2} \qquad X - a_{n-1}$$

De manière équivalente, on peut représenter cet arbre par un tableau à 2 dimensions $B_{i,j}$, pour $0 \leq i \leq \kappa$, $0 \leq j \leq 2^{\kappa-i} - 1$. Alors

$$B_{i,j} = \prod_{\ell=2^i j}^{2^i(j+1)-1} (X - a_\ell).$$

Par exemple, si $\kappa = 2$ et $n = 4$, l'arbre associé à a_0, a_1, a_2, a_3 est représenté par :

$$B_{0,0} = X - a_0, B_{0,1} = X - a_1, B_{0,2} = X - a_2, B_{0,3} = X - a_3,$$
$$B_{1,0} = (X - a_0)(X - a_1), B_{1,1} = (X - a_2)(X - a_3),$$
$$B_{2,0} = (X - a_0)(X - a_1)(X - a_2)(X - a_3).$$

L'intérêt de cette structure d'arbre est double : d'une part, il contient tous les polynômes par lesquels on va diviser lors de l'algorithme d'évaluation, d'autre part, le coût de son calcul s'amortit, et devient essentiellement linéaire en le degré.

PROPOSITION 2. *On peut calculer tous les polynômes contenus dans l'arbre $\mathcal{B}$ pour $O(\mathsf{M}(n) \log n)$ opérations $(+, -, \times,)$ dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $\mathsf{T}(n)$ le coût du calcul de l'arbre pour n points. La définition récursive implique l'inégalité suivante pour $\mathsf{T}(n)$:

$$\mathsf{T}(n) \leq 2\mathsf{T}\left(\frac{n}{2}\right) + \mathsf{M}\left(\frac{n}{2}\right).$$

Le résultat s'ensuit aisément en invoquant le lemme « diviser pour régner ». □

4.3. Évaluation. L'algorithme d'évaluation devient simple à écrire de manière récursive, si on s'autorise un peu de souplesse dans l'écriture, en supposant précalculés tous les nœuds de l'arbre.

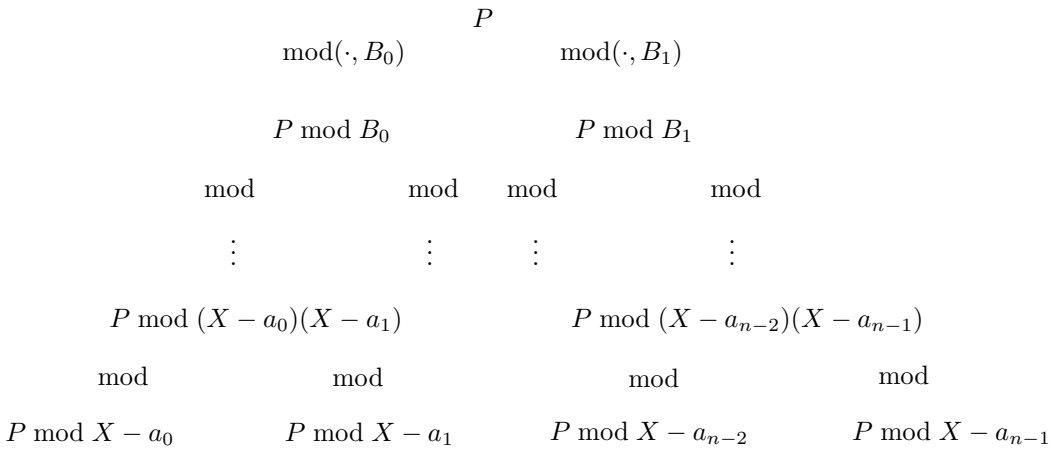
EvaluationRapide

Entrée : P dans $\mathbb{A}[X]$ et $a_0, \ldots, a_{n-1}$ dans $\mathbb{A}$, avec $\deg P < n$.

Sortie : $P(a_0), \ldots, P(a_{n-1})$.

1. Si $n = 0$, renvoyer P
2. $P_0 \leftarrow P \bmod (X - a_0) \cdots (X - a_{n/2-1})$
3. $P_1 \leftarrow P \bmod (X - a_{n/2}) \cdots (X - a_{n-1})$
4. Calculer (par un appel récursif) $L_0 = P_0(a_0), \ldots, P_0(a_{n/2-1})$
5. Calculer (par un appel récursif) $L_1 = P_1(a_{n/2}), \ldots, P_1(a_{n-1})$
6. Renvoyer L_0, L_1

Graphiquement, cela correspond à faire *descendre* les restes de divisions euclidiennes dans l'arbre des sous-produits :



Or, la division avec reste par un polynôme unitaire de degré n se fait en $O(\mathsf{M}(n))$ opérations de $\mathbb{A}$ (Cours 2). Nous obtenons le résultat de complexité suivant :

PROPOSITION 3. *Supposant l'arbre $\mathcal{B}$ précalculé, la complexité de l'algorithme précédent est de $O(\mathsf{M}(n) \log n)$ opérations dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $\mathsf{T}(n)$ le coût du calcul pour n points. L'algorithme ci-dessus implique la récurrence suivante pour $\mathsf{T}(n)$:

$$\mathsf{T}(n) \leq 2\,\mathsf{T}\left(\frac{n}{2}\right) + O(\mathsf{M}\,(n)),$$

d'où l'on déduit le résultat, à l'aide du lemme « diviser pour régner ». □

EXERCICE 2. Montrer que la FFT est un cas particulier de l'algorithme ci-dessus. (Indication : pour les moduli $m_i X - \omega^i$, où $\omega \in \mathbb{A}$ est une racine primitive de l'unité, l'arbre des sous-produits contient des polynômes de la forme $X^d - c$, $c \in \mathbb{A}$, modulo lesquels les divisions se font en complexité linéaire).

4.4. Sommes de fractions. Un problème intermédiaire à traiter avant de résoudre l'interpolation est celui du calcul efficace d'une somme de fractions. Rappelons les définitions introduites dans la Section 3. À partir des points a_i, nous y avions défini les polynômes

$$A = \prod_j (X - a_j) \qquad \text{et} \qquad A_i = \prod_{j \neq i} (X - a_j) = \frac{A}{X - a_i}.$$

Il était apparu que pour effectuer l'interpolation, il fallait savoir effectuer la tâche suivante : étant données des valeurs c_i, calculer le numérateur et le dénominateur de la fraction rationnelle

$$\sum_{i=0}^{n-1} \frac{c_i}{X - a_i}. \tag{1}$$

C'est à cette question que nous allons répondre maintenant ; à nouveau, on utilise une idée de type « diviser pour régner » : par deux appels récursifs, on calcule

$$S_1 = \sum_{i=0}^{n/2-1} \frac{c_i}{X - a_i} \; 1 \quad \text{et} \quad S_2 = \sum_{i=n/2}^{n} \frac{c_i}{X - a_i}$$

et on renvoie $S = S_1 + S_2$. Soit $\mathsf{T}(n)$ le coût du calcul pour n points. Le coût de l'algorithme SommesFractions esquissé ci-dessus satisfait à la récurrence suivante :

$$\mathsf{T}(n) \leq 2\mathsf{T}\left(\frac{n}{2}\right) + O(\mathsf{M}(n)),$$

d'où l'on déduit le résultat suivant, à l'aide du lemme « diviser pour régner ».

PROPOSITION 4. *L'algorithme* SommesFractions *calcule le dénominateur et le numérateur de la fraction rationnelle (1) en* $O(\mathsf{M}(n)\log n)$ *opérations dans* $\mathbb{A}$.

EXERCICE 3. Estimer la complexité de l'algorithme direct pour évaluer un polynôme de degré au plus n et ses premières n dérivées en un point. Imaginer un algorithme de complexité quasi-linéaire résolvant ce problème.

4.5. Interpolation. Arrivés à ce stade, l'interpolation ne pose plus de réelle difficulté. Au vu des formules de la Section 3, le polynôme interpolant les valeurs b_i aux points a_i est donné par :

$$P(X) \;=\; \sum_{i=0}^{n-1} b_i \frac{A_i(X)}{A_i(a_i)} \;=\; A(X) \times \sum_{i=0}^{n-1} \frac{b_i/A_i(a_i)}{X - a_i}.$$

D'après les paragraphes précédents, on sait comment calculer rapidement la fraction rationnelle $\sum c_i/(X-a_i)$; il ne reste plus qu'à calculer les constantes $c_i = b_i/A_i(a_i)$. Cela devient évident au vu de la proposition suivante :

PROPOSITION 5. *Pour tout* $i = 0, \ldots, n-1$, *on a* $A_i(a_i) = A'(a_i)$.

DÉMONSTRATION. On dérive A, ce qui donne :

$$A' = \sum_{i=0}^{n-1} \prod_{j \neq i} (X - a_j) = \sum_{i=0}^{n-1} A_i.$$

On a vu précédemment que $A_i(a_j) = 0$ pour $i \neq j$, ce qui donne le résultat. □

L'algorithme d'interpolation et l'estimation de son coût s'en déduisent facilement.

InterpolationRapide

Entrée : $a_0, \ldots, a_{n-1} \in \mathbb{A}$ vérifiant **(H)** et $b_0, \ldots, b_{n-1}$ dans $\mathbb{A}$.

Sortie : L'unique polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que $P(a_i) = b_i$ pour tout i.

1. Calculer l'arbre des sous-produits associé aux points a_i (de racine A).
2. $(d_0, \ldots, d_{n-1}) \leftarrow$ EvaluationRapide$(A', (a_0, \ldots, a_{n-1}))$
3. $(c_0, \ldots, c_{n-1}) \leftarrow (b_0/d_0, \ldots, b_{n-1}/d_{n-1})$
4. $R \leftarrow$ SommesFractions$(c_0/(X - a_0), \ldots, c_{n-1}/(X - a_{n-1}))$
5. Renvoyer le numérateur de R.

PROPOSITION 6. *La complexité de l'algorithme ci-dessus est* $O(\mathsf{M}(n)\log n)$ *opérations de* $\mathbb{A}$.

DÉMONSTRATION. C'est une conséquence des Propositions 2, 3 et 4. □

EXERCICE 4. Soit k un corps et soient $m_1, m_2 \in k[X]$ de degrés au plus n, premiers entre eux et soit $v_1, v_2 \in k[X]$ de degrés inférieurs à n. Donner un algorithme de complexité $O(\mathsf{M}(n)\log n)$ qui calcule $f \in k[X]$ de degré inférieur à $2n$ tel que $f = v_1 \bmod m_1$ et $f = v_2 \bmod m_2$. En déduire un algorithme diviser pour régner pour l'interpolation polynomiale en degré n, de complexité $O(\mathsf{M}(n)\log^2 n)$.

4.6. Évaluation et interpolation sur une suite géométrique. Dans cette section nous montrons que tout polynôme de degré n peut être évalué et interpolé sur des points en progression géométrique $\{1, q, \ldots, q^{n-1}\}$ en $O(\mathsf{M}(n))$ opérations.

PROPOSITION 7. *Soit $q \in \mathbb{A}, n \geq 1$ tels que $q, q-1, q^2-1, \ldots, q^{n-1}-1$ soient inversibles dans $\mathbb{A}$. Si $F \in \mathbb{A}[X]$ est de degré strictement inférieur à n alors :*

- *On peut évaluer F sur les points $a_i = q^i$, pour $0 \leq i \leq n-1$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.*
- *On peut interpoler F sur les points $a_i = q^i$, pour $0 \leq i \leq n-1$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $F = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$. Pour $i = 0, \ldots, 2n-2$, on introduit les nombers triangulaires $t_i = i(i-1)/2$ et la suite $b_i = q^{t_i}$; pour $i = 0, \ldots, n-1$ on construit $c_i = f_i/b_i$. Tous les éléments q^{t_i}, c_i et b_i se calculent en $O(n)$ opérations, car $q^{t_{i+1}} = q^i q^{t_i}$. L'algorithme exploite la formule

$$F(q^i) = \sum_{j=0}^{n-1} f_j q^{ij} = b_i^{-1} \cdot \sum_{j=0}^{n-1} c_j b_{i+j},$$

qui montre que les valeurs $F(q^i)$ sont, à des facteurs constants près, données par les coefficients de $X^{n-1}, \ldots, X^{2n-2}$ dans le produit de $\sum_{i=0}^{n-1} c_i X^{n-i-1}$ et $\sum_{i=0}^{2n-2} b_i X^i$.

Pour l'interpolation, on commence par noter que la racine $A(X)$ de l'arbre des sous-produits peut être calculée en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$. En effet, puisque les points a_i forment une suite géométrique, les nœds $B_{i,j}$ à l'étage i peuvent se déduire l'un de l'autre par une homothétie de coût $O(n)$. Par ailleurs, observons que dans l'algorithme InterpolationRapide, le calcul de tout l'arbre des sous-produits est nécessaire *uniquement* pour l'évaluation multipoint de A'. Or, par la première partie du théorème, l'évaluation de A' sur les points a_i se fait en $O(\mathsf{M}(n))$ par un algorithme *qui ne requiert pas le calcul de tout l'arbre des sous-produits*.

Il nous reste à prouver que la somme des fractions (1), qui devient ici

$$\frac{N}{A} = \sum_{i=0}^{n-1} \frac{c_i}{X - q^i} \tag{2}$$

peut également être déterminée pour le même prix.

On adopte la stratégie suivante : on calcule le développement en série à l'ordre n de la fraction (2). Celui-ci suffit pour en déduire son numérateur $N(X)$. Pour ce faire, on utilise la formule $1/(a-X) = \sum_{i \geq j} a^{-j-1}X^j$ valable dans $\mathbb{A}[[X]]$ pour tout $a \in \mathbb{A}$ inversible. On obtient :

$$\sum_{i=0}^{n-1} \frac{c_i}{X - q^i} \mod X^n = -\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} c_i q^{-i(j+1)} X^j \right) = -\sum_{j=0}^{n-1} C(q^{-j-1}) X^j,$$

où $C(X)$ est le polynôme $\sum_{i=0}^{n-1} c_i X^i$. Or, les points $q^{-1}, q^{-2}, \ldots, q^{-n}$ étant en progression géométrique, on sait évaluer C sur ces points en $O(\mathsf{M}(n))$. □

Exercice 5. Montrer que tout l'arbre des sous-produits associé aux points $a_i = q^i$, $i = 0, \ldots, n-1$, peut être calculé en $\mathsf{M}(n) + O(n \log n)$ opérations dans $\mathbb{A}$.

Exercice 6. Soient $a, b, c, d \in \mathbb{A}$. Montrer qu'on peut évaluer tout polynôme de degré au plus n en $\{ba^{2i} + ca^i + d, \quad 0 \leq i < n\}$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.

Notes

Pour évaluer un polynôme $P(X) = p_{n-1}X^{n-1} + \cdots + p_0$ en un seul point a, le *schéma de Horner* $P(a) = (\cdots((p_{n-1}a + p_{n-2})a + p_{n-3})a + \cdots + p_0)$ minimise le nombre d'opérations (additions ou multiplications). Cet algorithme effectue $n-1$ additions et $n-1$ multiplications dans $\mathbb{A}$, et on ne peut pas faire mieux en général (si les coefficients du polynôme sont algébriquement indépendants — moralement, s'ils sont des indéterminées). Ce résultat d'optimalité a été conjecturé par Ostrowski [13] et prouvé par Pan [14]. Par contre, pour un polynôme donné, il peut être possible de faire mieux : l'évaluation de $P(X) = X^{n-1}$ se fait en temps logarithmique.

Une forme particulière du Théorème de restes Chinois a été enoncée il y plus de 2000 ans par le mathématicien chinois Sun Tsu. Le traitement moderne est dû à Gauss [8]. Ce théorème admet une formulation très générale, en termes d'idéaux d'anneaux non nécessairement commutatifs : Si R est un anneau et $I_1, \ldots, I_r$ des idéaux (à gauche) de R mutuellement premiers (*i. e.* $I_i + I_j = R$ pour $i < j$), alors l'anneau quotient $R/\cap_i I$ est isomorphe à l'anneau produit $\prod R/I_i$ via l'isomorphisme $x \bmod I \quad \mapsto (x \bmod I_1, \ldots, x \bmod I_r)$. La formule d'interpolation de Lagrange est due à Waring [20]. Les avantages pratiques de l'arithmétique modulaire ont été mises en évidence dans les années 1950 par Svoboda et Valach [19].

Le premier algorithme sous-quadratique, de complexité arithmétique $O\big(n^{1.91}\big)$, pour l'évaluation multipoint est dû à Borodin et Munro [4] et repose sur une approche *pas de bébés / pas de géants* exploitant la multiplication sous-cubique des matrices de Strassen [17]. Horowitz [10] a étendu ce résultat à l'interpolation. La Proposition 2 calcule efficacement les fonctions symétriques élémentaires de n éléments $a_0, \ldots, a_{n-1}$ de $\mathbb{A}$; elle est due également à Horowitz [10].

S'inspirant de la FFT, Fiduccia [7] eut l'idée d'un algorithme pour l'évaluation multipoint à base de division polynomiale récursive. Ne disposant pas de division de complexité sous-quadratique, son algorithme récursif reste quadratique. Borodin et Moenck corrigent ce point dans deux articles successifs [11, 3], reposant sur les algorithmes quasi-optimaux pour la division de [11, 18]. Montgomery [12] améliore la constante dans la complexité $O\big(\mathsf{M}(n) \log n\big)$ de l'évaluation multipoint, sous l'hypothèse que la FFT est utilisée pour la multiplication polynomiale. L'algorithme d'évaluation sur une suite géométrique exposé dans ce cours vient de [15, 2]. Les algorithmes fournissant les meilleures constantes actuellement connues dans les $O(\cdot)$ du Théorème 1 sont obtenus à l'aide du principe de transposition de Tellegen [5, 6].

On connaît peu de choses sur la complexité intrinsèque de l'évaluation multipoint et de l'interpolation en degré n. Strassen [18] a prouvé une borne inférieure de type $n \log n$. Shoup et Smolensky [16] ont montré que ces bornes restent essentiellement vraies même dans un modèle où des précalculs en quantité illimitée sur les points d'évaluation sont permis. Cependant, l'optimalité des meilleurs algorithmes connus, de complexité $O(n \log^2 n)$ reste un problème ouvert. Même dans le cas particulier où les points d'évaluation sont en progression arithmétique, on ne dispose d'aucun

algorithme d'évaluation multipoint de complexité $O(\mathsf{M}(n))$, ni d'une preuve qu'un tel algorithme n'existe pas. De même, pour des points quelconques $a_0, \ldots, a_{n-1}$ on ne connaît pas d'algorithme de complexité $O(\mathsf{M}(n))$ pour calculer le polynôme $\prod_i (X - a_i)$. Notons toutefois que tels algorithmes existent dans les cas particuliers où les a_i forment une progression arithmétique ou géométrique [6].

L'exercice 4 est provient de [9] ; historiquement, ce fut le premier algorithme rapide pour les restes chinois. Les exercices 3 et 6 sont tirés de [1].

Bibliographie

[1] Aho (A. V.), Steiglitz (K.), and Ullman (J. D.). – Evaluating polynomials at fixed sets of points. *SIAM Journal on Computing*, vol. 4, n° 4, 1975, pp. 533–539.

[2] Bluestein (L. I.). – A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Trans. Electroacoustics*, vol. AU-18, 1970, pp. 451–455.

[3] Borodin (A.) and Moenck (R. T.). – Fast modular transforms. *Comput. Sys. Sci.*, vol. 8, n° 3, 1974, pp. 366–386.

[4] Borodin (A.) and Munro (I.). – Evaluation of polynomials at many points. *Information Processing Letters*, vol. 1, n° 2, 1971, pp. 66–68.

[5] Bostan (Alin), Lecerf (Grégoire), and Schost (Éric). – Tellegen's principle into practice. In Sendra (J. R.) (editor), *Symbolic and Algebraic Computation*. pp. 37–44. – ACM Press, 2003. Proceedings of ISSAC'03, Philadelphia, August 2003.

[6] Bostan (Alin) and Schost (Éric). – Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, vol. 21, n° 4, 2005, pp. 420–446. – Festschrift for Arnold Schönhage.

[7] Fiduccia (C. M.). – Polynomial evaluation via the division algorithm : The fast fourier transform revisited. In *Conference Record, Fourth Annual ACM Symposium on Theory of Computing*, pp. 88–93. – 1972.

[8] Gauss (Carl Friedrich). – *Disquisitiones arithmeticae*. – Yale University Press, New Haven, Conn., 1966, *Translated into English by Arthur A. Clarke, S. J*, xx+472p.

[9] Heindel (L. E.) and Horowitz (E.). – On decreasing the computing time for modular arithmetic. In *12 th annual symposium on switching and automata theory*. pp. 126–128. – IEEE Computer Society Press, 1971.

[10] Horowitz (E.). – A fast method for interpolation using preconditioning. *Information Processing Letters*, vol. 1, n° 4, 1972, pp. 157–163.

[11] Moenck (R. T.) and Borodin (A.). – Fast modular transforms via division. *Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972)*, 1972, pp. 90–96.

[12] Montgomery (P. L.). – *An FFT extension of the elliptic curve method of factorization*. – PhD thesis, University of California, Los Angeles CA, 1992.

[13] Ostrowski (A.). – On two problems in abstract algebra connected with Horner's rule. In *Studies in mathematics and mechanics presented to Richard von Mises*, pp. 40–48. – Academic Press Inc., New York, 1954.

[14] Pan (V. Ja.). – On means of calculating values of polynomials. *Uspehi Mat. Nauk*, vol. 21, n° 1 (127), 1966, pp. 103–134.

[15] Rabiner (L. R.), Schafer (R. W.), and Rader (C. M.). – The chirp z-transform algorithm and its application. *Bell System Tech. J.*, vol. 48, 1969, pp. 1249–1292.

[16] Shoup (Victor) and Smolensky (Roman). – Lower bounds for polynomial evaluation and interpolation problems. *Computational Complexity*, vol. 6, n° 4, 1996/97, pp. 301–311.

[17] Strassen (V.). – Gaussian elimination is not optimal. *Numerische Mathematik*, vol. 13, 1969, pp. 354–356.

[18] Strassen (V.). – Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik*, vol. 20, 1972/73, pp. 238–251.

[19] Svoboda (A.) and Valach (M.). – Oper'atorov'e obvody (operational circuits). *Stroje na Zpracov 'an'i Informac'i III, Nakl. CSAV, Praha*, 1955, pp. 247–295.

[20] Waring (E.). – Problems concerning interpolations. *Philosophical Transactions of the Royal Society of London*, vol. 59, 1779, pp. 59–67.