# Arithmétique polynomiale efficace par codage entier et évaluation de sommes combinatoires

Sujet proposé par Frédéric Chyzak

frederic.chyzak@inria.fr

## 1 Objectifs

Parce qu'une grande majorité des manipulations du calcul formel reposent ultimement sur des polynômes à coefficients entiers, il est crucial pour tout système de calcul formel de disposer d'une librairie arithmétique optimisée pour des polynômes de degré arbitraire. Dans le cas de polynômes comme dans le cas des entiers, le développement d'une telle librairie arithmétique serait un projet très technique et hors d'atteinte dans le cadre d'un projet informatique : rien que pour le produit, l'utilisation de plusieurs algorithmes récursifs et s'appelant les uns les autres serait indispensable ! Néanmoins, une librairie bien optimisée pour le calcul entier est disponible et interfaçable avec divers langages de programmation. On propose dans ce sujet d'employer un codage astucieux des polynômes par des entiers de façon à obtenir des calculs sur les polynômes en temps à peine plus que linéaire en le degré, et non quadratique comme par un algorithme naïf.

Nombre de sommes de la combinatoire, sommes de produits de factorielles et coefficients binomiaux, peuvent être calculées symboliquement en se ramenant à vérifier des égalités entre polynômes, ce qui fait de la sommation symbolique une bonne application de l'arithmétique polynomiale exacte. On propose d'implanter la méthode de Fasenmyer, approche élémentaire qui est à l'origine des algorithmes de sommation implantés dans les systèmes de calcul formel actuels.

## 2 Le codage de polynômes en entiers

Un codage simple des polynômes par des entiers repose sur l'idée que dans une base de numération $B$ donnée, on n'aura pas de retenue dans un produit si on se restreint à n'utiliser que des chiffres « petits ». Autrement dit, pour des polynômes à coefficients « pas trop grands », l'indéterminée $X$ se comporte comme une base $B$ « suffisamment grande ».

Prenons un exemple. Soit à calculer le produit des polynômes $p = 12X^2 + 47X + 34$ et $q = 32X^2 + 84X + 51$, et pour visualiser le calcul, choisissons la base $B = 10^6$. Alors $p(B) = 12\,000047\,000034$ et $q(B) = 32\,000084\,000051$, et le produit d'entiers $p(B)q(B)$ vaut $384\,002512\,005648\,005253\,001734$, où l'on lit, en se servant des doubles 0 comme de séparateurs, le produit les polynômes, $pq = 384X^4 + 2512X^3 + 5648X^2 + 5253X + 1734$.

Reprenons l'exemple en ajoutant quelques signes moins : posons $p = 12X^2 + 47X - 34$ et $q = 32X^2 - 84X + 51$. Cette fois-ci, $p(B) = 12000046999966$ et $q(B) = 31999916000051$. Remarquons l'apparition de trainées de 9, qui s'expliquent par la formule $uB - v = (u-1)B + w$ avec $w = B - v \geq 0$ dès lors que $u$ et $v$ sont positifs petits devant $B$. Le produit d'entiers $p(B)q(B)$ de notre exemple vaut

384 000495 995576 005252 998266, induisant le produit de polynômes $pq = 384X^4 + 496X^3 - 4424X^2 + 5253X - 1734$.

Un dernier exemple explicite la règle générale : pour $p = 2X^4 - 23X^3 - 12X^2 - 57X - 93$, l'entier $p(B)$ vaut 1 999976 999987 999942 999907. On extrait donc le polynôme en lisant l'entier de la droite vers la gauche, une tranche $c$ commençant par un 9 donnant un coefficient $-(B - c)$ du polynôme et devant transmettre une retenue de 1 sur sa gauche.

Notons que, même pour ces exemples très simples, des entiers de plus de 64 bits (tels les `long` de Java) sont déjà nécessaires (ci-dessus, plus de 80 bits sont nécessaires). C'est pourquoi des entiers en précision arbitraire doivent être utilisés (tels ceux de la librairie `gmp` écrite en C et disponible en Java via la classe `java.math.BigInteger`.)

On s'est jusque-là limité au cas de coefficients dominants positifs. Pour traiter de coefficients dominants entiers quelconques, on remplace éventuellement un polynôme par son opposé en sauvegardant à part le « signe » du polynôme d'origine ; lors d'un produit, ces signes sont multipliés indépendamment du reste des polynômes. De même, pour traiter de polynômes à coefficients rationnels, on considèrera toujours ces derniers en factorisant un dénominateur convenable, de façon à se ramener au cas de coefficients entiers.

En utilisant deux bases adaptées, il est possible d'étendre l'approche précédente à des polynômes de deux indéterminées. Même si l'idée fonctionne en théorie pour un plus grand nombre d'indéterminées, disons $v$, le codage requiert $v!$ fois plus de mémoire que l'écriture du polynôme produit lui-même, ce qui induit le même facteur dans la complexité en temps.

# 3 Analyse de la base de numération à utiliser

Il n'est pas possible de fixer une base $B$ une fois pour toute, et celle-ci doit dépendre des arguments de chaque opération. Pour deux polynômes $p = p_0 + \cdots + p_d X^d$ et $q = q_0 + \cdots + q_d X^d$ de degré $d$ et à coefficients entiers, notons $|p|$ et $|q|$ le maximum des valeurs absolues des coefficients de chaque polynôme. Alors, le coefficient de $X^i$ dans le produit de polynômes $pq$ est donné comme la somme d'au plus $d+1$ produits d'entiers $p_j q_{i-j}$. On obtient ainsi la relation $|pq| \leq (d+1)|p||q|$. Pour le calcul de $pq$, on prendra donc pour $B$ une puissance de 2 supérieure à cette borne. Mieux, on s'arrangera pour que cette puissance correspondent à un nombre entier de mots machines (ou de `bytes` de Java, utilisés par certaines méthodes utiles de la classe `java.math.BigInteger`).

Cette analyse peut s'adapter au calcul d'une division euclidienne : étant donnés deux polynômes $a$ et $b$ à coefficients entiers, on s'intéresse à déterminer un quotient $q$ et un reste $r$ de degré strictement inférieur à celui de $b$, tous deux à coefficients rationnels, tels que $a = qb + r$. Notons $\delta$ l'expression $\deg a - \deg b + 1 = \deg q + 1$ et $c$ le coefficient dominant de $b$. Alors, le calcul revient à chercher une relation $(c^\delta a) = (c^\delta q)b + (c^\delta r)$ où chaque expression entre parenthèses est un polynôme à coefficients entiers. On vérifiera que la division euclidienne entière de $c^\delta a(B)$ par $b(B)$ fournit les polynômes $q$ et $r$ à condition de prendre pour $B$ la base qui serait utilisée pour le produit de $c^\delta q$ par $b$.

De même, pour le calcul du p.g.c.d. de deux polynômes $p$ et $q$ de degrés respectifs $d$ et $d'$ comme polynôme représenté par le p.g.c.d. entier d'évaluations en une même base entière $B$, une borne inférieure acceptable sur la base est $(d + d' + 2)! \max\{|p|, |q|\}^{d+d}$. Notons qu'un test rapide pour savoir si le p.g.c.d. de deux polynômes vaut 1 est de tester si le p.g.c.d. entier des évaluations des deux polynômes en un même entier $N$ vaut 1. En effet, si deux polynômes sont premiers entre, leurs évaluations le sont nécessairement. L'avantage de ce test probabiliste est que le nombre $N$ n'a pas à être choisi aussi grand que la base $B$ utilisée pour le calcul du p.g.c.d.

Pour des polynômes $p$ et $q$ en deux indéterminées $X$ et $Y$ de degré en $X$ au plus $d$ et de degré en $Y$ au plus $d'$, on a la formule $|pq| \leq (d+1)(d'+1)|p||q|$. En prenant une base $B$ supérieure à cette borne, une formule d'évaluation qui convient est $p(B, B^{d+1})$, une autre étant $p(B^{d'+1}, B)$.

# 4 Complexité des calculs polynomiaux

La « taille » d'un entier exprime la quantité de mémoire, en nombre de bits ou de mots machines, nécessaire pour le sauvegarder dans sa représentation binaire. Ainsi, les entiers de taille $n$ ont des valeurs au plus exponentielle en $n$. Le coût en temps du produit de deux entiers de taille $n$, noté $M(n)$, dépend de l'algorithme utilisé. L'algorithme naïf, par multiplication d'un nombre par chacun des chiffres de l'autre, décalages et additions, est quadratique : $M(n) = O(n^2)$. Un algorithme récursif du type « diviser pour régner » et dû à Karatsuba a une complexité $M(n) = O(n^\alpha)$ pour $\alpha = \log_2 3 \approx 1.59$, et bat l'algorithme naïf pour des nombres de l'ordre de centaines de chiffres. Un autre algorithme récursif, à base de FFT, a une complexité encore plus basse, $M(n) = O(n \ln n \ln \ln n)$, et bat l'algorithme de Karatsuba pour des nombres de l'ordre de dizaines de milliers de chiffres.

Dans le codage pour le produit de polynômes en une indéterminée de degré $d$ à coefficients entiers de taille $n$, les entiers employés sont de taille $(d+1)(2n+\log_2 d)$ environ. Dans le régime où la FFT s'applique, par exemple à partir de polynômes de degré cent et de coefficients de taille une centaine, le coût du produit de polynômes, $M\big((d+1)(2n+\log_2 d)\big)$, devient ainsi quasiment linéaire en le degré.

La base $B$ des calculs étant une puissance de 2, l'évaluation des polynômes en $X = B$ ainsi que l'extraction des coefficients d'un produit $pq$ à partir de $p(B)q(B)$ pourra se faire par de simples décalages de bits, et non par des produits et divisions de grands entiers. Il est en effet crucial de pouvoir faire ces traductions en temps linéaire en la taille des données si on ne veut pas reperdre dans ces prétraitement et posttraitement l'efficacité obtenue par le calcul sur des entiers.

# 5 La méthode de Fasenmyer

Étant donnée une suite $(f_{n,k})_{n,k}$ hypergéométrique, c'est-à-dire telle que chacun des rapports $f_{n-1,k}/f_{n,k}$ et $f_{n,k-1}/f_{n,k}$ s'exprime comme une fraction rationnelle en $n$ et $k$, on s'intéresse à calculer la suite $(F_n)_n$ des sommes $F_n = \sum_{k=-\infty}^{\infty} f_{n,k}$, en faisant pour simplifier l'hypothèse que pour chaque entier $n$, la valeur $f_{n,k}$ est nulle en dehors d'un nombre fini d'entiers $k$. Ainsi, on veut pouvoir retrouver les identités classiques de la dernière section, dont par exemple l'identité de Dixon

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k}^3 = \begin{cases} (-1)^p \binom{3p}{p,p,p} & \text{si } n = 2p, \\ 0 & \text{si } n = 2p+1. \end{cases}$$

Notons que pour cet exemple comme pour tous ceux de la dernière section, les rapports $f_{n-1,k}/f_{n,k} = \big((n-k)/n\big)^3$ et $f_{n,k-1}/f_{n,k} = -\big(k/(n-k+1)\big)^3$ s'obtiennent immédiatement après reformulation en factorielles de $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$.

Pour un ensemble de décalages $S = \big\{(i_1, j_1), \ldots, (i_s, j_s)\big\}$ donné, par exemple un rectangle $\{1, \ldots, N_1\} \times \{1, \ldots, N_2\}$, la méthode procède en résolvant

$$\sum_{(i,j) \in S} c_{i,j}(n) f_{n-i,k-j} = 0$$

pour des coefficients $c_{i,j}$ polynomiaux en $n$. Après division par $f_{n,k}$ et après avoir chassé les dénominateurs, on trouve des polynômes $p_{i,j}$ tels que le problème soit équivalent à résoudre

$$\sum_{(i,j)\in S} c_{i,j}(n)p_{i,j}(n,k) = 0.$$

En égalant les coefficients des $k^l$ à 0, on obtient un système linéaire en les $c_{i,j}$, que l'on résoud après triangularisation par l'algorithme de Gauss. On obtient alors une ou plusieurs relations de la forme

$$\sum_{(i,j)\in S} c_{i,j}(n)f_{n-i,k} = g_{n,k} - g_{n,k-1}$$

pour un $g$ adéquat que l'on n'a pas besoin de déterminer. Après sommation sur $k$, on a la relation

$$\sum_{(i,j)\in S} c_{i,j}(n)F_{n-i} = \lim_{k\to\infty} g_{n,k} - \lim_{k\to-\infty} g_{n,k} = 0.$$

Autrement dit, on a obtenu une relation de récurrence sur $(F_n)_n$, qui dans les bons cas se résoud pour donner une formule à laquelle la somme est égale.

Pour $S = \{0,1,2,3\}^2$, la méthode de Fasenmyer renvoie un résultat positif, conduisant à la relation $3(3n-2)(3n-4)F_{n-2} + n^2F_n = 0$, et il ne reste plus qu'à vérifier la relation annoncée pour $n \in \{1,2\}$ pour avoir par récurrence sa preuve sur tous les entiers.

# 6 Travail demandé

La relative indépendance entre l'algorithmique sur les polynômes et la mise en œuvre de la méthode de Fasenmyer suggère, en première approximation, la répartition suivante, mais non imposée, d'un travail en binôme.

Sur l'arithmétique des polynômes, on demande :

- de définir une structure de données adaptée pour les polynômes en une indéterminée à coefficients rationnels ;
- de définir une structure de données adaptée pour conserver l'évaluation d'un polynôme en une indéterminée sur un grand entier ;
- d'écrire les opérations de conversion entre ces deux représentations ;
- d'implanter l'opération d'addition par l'algorithme naïf ;
- d'implanter les opérations de produit, puis de division euclidienne et de p.g.c.d. par codage entier ;
- d'implanter une opération de retrait du contenu d'un polynôme, c'est-à-dire de le diviser par le p.g.c.d. de ses coefficients ;
- d'implanter le calcul du décalage $p(X+1)$ d'un polynôme $p$ ;
- d'étendre addition, produit et décalage aux polynômes de deux indéterminées.

Sur l'application aux sommes combinatoires, on demande :

- d'implanter l'algorithme de Gauss sur une matrice rectangulaire à entrées des polynômes en une indéterminée (on s'efforcera de renvoyer une base de solutions sous la forme de vecteurs à coordonnées polynomiales, sans passer par des calculs sur des fractions rationnelles) ;

- d'implanter le calcul des fractions rationnelles $f_{n-i,k-j}/f_{n,k}$ (pour les entrées, on se contentera de fournir au programme les deux fractions rationnelles $f_{n-1,k}/f_{n,k}$ et $f_{n,k-1}/f_{n,k}$ décrivant une suite hypergéométrique) ;
- de mettre en œuvre la méthode de Fasenmyer pour calculer des relations de récurrence définissant les sommes des exemples ci-dessous.

Pour faciliter l'utilisation et le test du programme, on le dotera d'une interface texte minimale (sans analyse syntaxique complexe) permettant :

- de saisir les coefficients de deux polynômes et de faire une opération dessus (multiplication, division euclidienne, p.g.c.d., etc) ;
- de saisir les coefficients des deux fractions rationnelles décrivant une suite hypergéométrique et l'ensemble $S$ des décalages, avant d'appliquer la méthode de Fasenmyer.

# 7 Exemple de calculs

La méthode de Fasenmyer donne un résultat pour un $S$ assez grand sur chacun des exemples suivant. L'ordre de la liste ne préjuge en rien de la difficulté à trouver le bon $S$ ou à résoudre le système une fois le bon $S$ trouvé.

$$\sum_{k=0}^{n}\binom{n}{k} = 2^n, \qquad \sum_{k=0}^{n}(-1)^k\binom{n}{k} = 0, \qquad \sum_{k=0}^{n}k\binom{n}{k} = n2^{n-1},$$

$$\sum_{k=0}^{n}\binom{n}{k}^2 = \binom{2n}{n}, \qquad \sum_{k=0}^{n}(-1)^k\binom{2n}{k}^2 = (-1)^n\binom{2n}{n},$$

$$\sum_{k=0}^{n}(-2)^{n-k}\binom{n}{k}\binom{2k}{k} = \begin{cases}\binom{2p}{p} & \text{si } n = 2p,\\ 0 & \text{si } n = 2p+1,\end{cases}$$

$$\sum_{k=0}^{n}(-4)^k\binom{n}{k}\binom{2k}{k}^{-1} = \frac{1}{1-2n},$$

$$\sum_{k=0}^{n}(-1)^k\binom{n}{k}\binom{tk}{n} = (-t)^n, \quad t = 2,3,\ldots,$$

$$\sum_{k=0}^{n}(-1)^k\binom{n}{k}^3 = \begin{cases}(-1)^p\binom{3p}{p,p,p} & \text{si } n = 2p,\\ 0 & \text{si } n = 2p+1,\end{cases}$$

$$\sum_{k=0}^{n}(-1)^k\binom{2n}{k}\binom{2k}{k}\binom{4n-2k}{2n-k} = \binom{2n}{n}^2.$$

À titre d'exemple, la relation de récurrence intermédiaire pour la première formule (formule du binôme) n'est autre que la relation du triangle de Pascal, $f_{n,k} = f_{n-1,k-1} + f_{n-1,k}$, alors que celle obtenue pour la formule de Dixon est

$$\begin{aligned}
&n^2(3n-5)f_{n,k} + (9n^3 - 24n^2 + 17n - 4)f_{n-1,k-1}\\
&\quad + (3n-4)(21n^2 - 49n + 24)f_{n-2,k-1} + (3n-4)(3n^2 - 7n + 3)f_{n-2,k-2}\\
&\quad + 3(3n-2)(n-2)^2 f_{n-3,k-1} - 3(3n-2)(n-2)^2 f_{n-3,k-2}\\
&\quad + (3n-2)(n-2)^2 f_{n-3,k-3} + (-9n^3 + 24n^2 - 17n + 4)f_{n-1,k}\\
&\quad + (3n-4)(3n^2 - 7n + 3)f_{n-2,k} - (3n-2)(n-2)^2 f_{n-3,k} = 0.
\end{aligned}$$

## 8 Extensions possibles

Les extensions qui suivent ne sont pas nécessaires au fonctionnement minimal du programme, mais pourront améliorer sa convivialité ainsi que sa rapidité :

- une interface texte enrichie d'une analyse syntaxique reconnaissant les polynômes, fractions rationnelles, produits de coefficients binomiaux saisis dans une syntaxe standard ;
- le calcul des fractions rationnelles définissant une suite hypergéométrique à partir de son expression en terme de coefficients binomiaux ;
- l'itération automatique sur des ensembles $S$ de plus en plus grand dans la méthode de Fasenmyer ;
- l'optimisation de l'ensemble $S$ choisi dans la méthode de Fasenmyer : le système linéaire à résoudre ayant autant d'inconnues $c_{i,j}$ que d'éléments de $S$ et un nombre de contraintes donné par le degré $\delta$ en $k$ de la somme sur $S$ des $c_{i,j}(n)p_{i,j}(n,k)$, il est calculatoirement bon de compléter un ensemble $S$ rectangulaire par des paires $(i,j)$ qui ne font pas croître $\delta$, c'est-à-dire par des paires $(i,j)$ qui ne font pas changer le dénominateur commun de la somme des $c_{i,j}(n)f_{n-i,k-j}/f_{n,k}$.