

Développements de fonctions D-finies sur des polynômes de Tchebychev

Alexandre Benoit
directeur de stage : Bruno Salvy
Centre de Recherche Commun INRIA-Microsoft Research

24 août 2008

Le contexte général

Les fonctions D-finies sont les solutions d'équations différentielles linéaires à coefficients polynomiaux. Ces fonctions sont développables en série sur certaines familles de fonctions. Sous certaines conditions, les coefficients du développement en série vérifient une récurrence linéaire.

La famille de fonctions où ce problème est le plus étudié est la base monomiale. On a pour cette famille des algorithmes pour le calcul des récurrences quasi-optimaux en complexité arithmétique, c'est-à-dire que le nombre d'opérations arithmétiques est linéaire en la taille de sortie à des logarithmes près. De nombreux articles ont été publiés pour ce cas. Ils sont de plus implantés sur des logiciels de calcul formel comme Maple, avec Diffeqtorec dans le package gfun [17].

L'étude d'algorithmes de récurrence similaires à Diffeqtorec qui permettent efficacement de calculer les récurrences dans différentes bases de fonctions est moins approfondi, et il n'existe pas d'algorithme quasi-optimaux dans tous les cas. On sait réduire ce problème à un calcul de base de Gröbner non commutatif, mais les algorithmes sur les polynômes multivariés ne sont pas rapides. L'objectif serait donc de se ramener à du calcul sur les polynômes univariés comme le fait Diffeqtorec.

Le problème étudié

Les familles de polynômes orthogonaux classiques vérifient les conditions nécessaires à un développement en série de solutions d'équations différentielles linéaires dont les coefficients vérifient une récurrence linéaire. L'objectif de mon stage était de concevoir des algorithmes qui calculent efficacement ces récurrences.

Pour obtenir l'approximation d'une fonction par un polynôme sur un segment, la série tronquée à l'ordre n d'une fonction développée dans une base de polynômes de Tchebychev nous donne une meilleure approximation que la série de Taylor tronquée au même ordre. Cette propriété des polynômes de Tchebychev est certainement la principale raison pour laquelle des algorithmes qui calculent les récurrences dans cette base de polynômes existent déjà. Le premier objectif de ce stage était de comprendre le lien entre les différents algorithmes de calcul de récurrence dans le cas des polynômes de Tchebychev. Aucune analyse de complexité n'existait pour ces algorithmes, mon travail consistait donc aussi à calculer la complexité de ces algorithmes pour les comparer, puis à construire des algorithmes plus rapides.

La contribution proposée

Je donne un cadre algébrique et algorithmique commun aux algorithmes de récurrences existants. Ce cadre m'a permis de les comparer et de vérifier leur validité. Il m'a aussi permis d'effectuer l'analyse de complexité de ces algorithmes. De cette analyse de complexité, j'ai pu déduire des algorithmes rapides construisant ces mêmes récurrences.

Je propose aussi du code Maple qui permet de calculer les récurrences dans le cas des polynômes de Tchebychev avec des algorithmes rapides. Mon code permet aussi le calcul des récurrences dans d'autres bases que les polynômes de Tchebychev comme la famille des polynômes de Jacobi.

Les arguments en faveur de sa validité

Le cadre que je donne pour calculer des récurrences est mathématiquement correct, il permet de tirer parti de l'algorithmique rapide comme la multiplication rapide de polynômes tordus. Ce cadre permet aussi facilement d'adapter les mêmes idées voire les mêmes algorithmes au calcul de récurrences dans d'autres bases de fonctions.

Je propose des algorithmes plus rapides que ceux existants, cette rapidité est prouvée sur le plan théorique par une analyse de complexité. On retrouve aussi cette efficacité sur des tests sur ordinateurs. Je peux facilement calculer les récurrences pour des équations différentielles d'ordre supérieur à 20, ce que les algorithmes déjà existants implantés sur les mêmes ordinateurs pouvaient difficilement effectuer.

Le bilan et les perspectives

J'ai un code permettant de calculer des récurrences pour des familles de polynômes comme Tchebychev ou Jacobi ; la complexité et la correction de mes algorithmes sont prouvées. Ces algorithmes existent car les familles de fonctions sur lesquelles je travaille obéissent à certaines propriétés de D-finitudes que j'ai déterminées. Un prochain travail est donc d'obtenir des résultats similaires avec les autres familles de fonctions qui vérifient ces propriétés, certaines sont connues comme les fonctions de Bessel ou l'ensemble des polynômes orthogonaux classiques, d'autres seront à déterminer. Les algorithmes que je donne sont spécifiquement adaptés pour les familles de polynômes de Tchebychev. Pour d'autres familles de polynômes comme celle d'Hermite, ces algorithmes ne sont plus optimaux. Mais la structure d'opérateurs de récurrence que je donne permet de les adapter à ces familles. On peut par la suite espérer obtenir des algorithmes efficaces pour toutes ces familles de polynômes.

En ce qui concerne l'implantation, mes algorithmes utilisent de la multiplication rapide de polynômes et de matrices, Maple ne possède pas ces multiplications. On pourrait réécrire ces algorithmes sur des logiciels possédant des multiplications rapides.

Table des matières

1	Algorithmique des corps de fractions rationnelles de récurrences	8
1.1	L'anneau des opérateurs de récurrence	8
1.2	Polynômes de Laurent	12
1.3	Corps des fractions rationnelles de récurrences	13
1.3.1	Fractions irréductibles	14
2	Développements en séries de Tchebychev	15
2.1	La méthode de Clenshaw	16
2.2	L'algorithme de Paszkowski	16
2.3	L'algorithme de Lewanowicz	18
2.4	Equivalence entre les algorithmes de Paszkowski et Lewanowicz	19
2.5	L'algorithme de Rebillard	21
2.6	Nouveaux algorithmes	22

Introduction

Les fonctions D-finies sont les solutions d'équations différentielles linéaires à coefficients polynomiaux. Une fonction D-finie peut être développée en série entière au voisinage d'un point, on parle alors de série formelle D-finie. Une propriété importante de D-finitude est qu'une série formelle est D-finie si et seulement si la suite de ses coefficients est P-récurrente, c'est-à-dire qu'elle vérifie une récurrence linéaire à coefficients polynomiaux. Le calcul de ces récurrences permet d'exprimer formellement et efficacement une solution d'équation différentielle linéaire. En développant ces fonctions dans certaines autres bases de fonctions, on obtient également une récurrence linéaire sur les coefficients, le propos de mon stage est de calculer ces récurrences.

Pour calculer une récurrence linéaire sur les coefficients dans la base des polynômes de Taylor $\{y_n(x); y_n(x) = x^n\}$, on utilise les deux relations suivantes :

$$xy_n(x) = y_{n+1}(x), \quad (1)$$

$$y'_n(x) = ny_{n-1}(x). \quad (2)$$

Par exemple l'équation différentielle

$$(4 + x^2)y'' + 2xy' = 0 \quad (3)$$

a pour solution la fonction $\arctan(\frac{x}{2})$. Cette fonction se développe en série de Taylor au voisinage de 0, $f := \sum c_n y_n(x) = \sum \frac{(-1)^n}{(2n+1)!} x^{2n+1}$. En appliquant l'équation différentielle à l'égalité (2) on a :

$$(4 + x^2) \sum_{n=2}^{+\infty} n(n-1)c_n y_{n-2}(x) + 2x \sum_{n=0}^{+\infty} n c_n y_{n-1}(x) = 0.$$

Puis avec l'égalité (1), on a :

$$4 \sum_{n=2}^{+\infty} n(n-1)c_n y_{n-2}(x) + \sum_{n=2}^{+\infty} n(n-1)c_n y_n(x) + 2 \sum_{n=0}^{+\infty} n c_n y_n(x) = 0.$$

En mettant le tout sous la même base, on obtient :

$$\sum_{n=0}^{+\infty} (4(n+1)(n+2)c_{n+2} + n(n+1)c_n) y_n(x) = 0$$

On déduit de cette égalité la récurrence suivante :

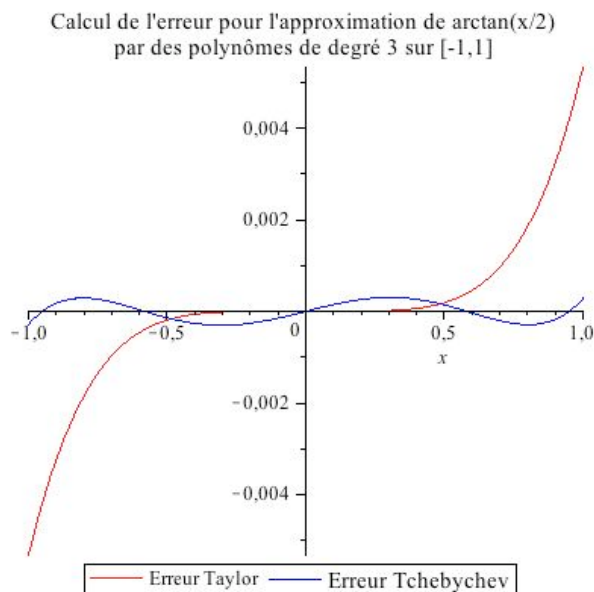
$$4(n+2)c_{n+2} + n c_n = 0 \quad (4)$$

vérifiée par les coefficients de $\arctan(\frac{x}{2})$.

Diffeqtoec, implanté en Maple dans le package gfun [17], donne une telle récurrence pour toutes les équations différentielles linéaires à coefficients polynomiaux. Pour le calcul, Diffeqtoec se base comme dans l'exemple sur les égalités (1) et (2). Ces égalités donnent des relations entre l'opérateur de dérivation qui s'applique à la fonction et une récurrence entre les coefficients de la série.

Il existe pour ce calcul des algorithmes quasi-optimaux, c'est-à-dire que le nombre asymptotique d'opérations arithmétiques pour ce calcul est linéaire en la taille de l'entrée à des logarithmes près, voir par exemple [2, p.67]. La taille est ici le nombre de coefficients de l'équation différentielle, c'est-à-dire approximativement l'ordre de l'équation multiplié par le degré des coefficients. De manière générale, on considère ici la taille d'une entrée d'algorithme comme le nombre d'éléments du corps de base \mathbb{K} , qui sera toujours de caractéristique nulle. Dans l'exemple on a $\mathbb{K} = \mathbb{Q}$ et les opérations arithmétiques sont les opérations effectuées dans ce corps.

Une des applications possibles d'un développement d'une fonction D-finie en série de Taylor est son approximation par un polynôme de degré n . En tronquant la série à l'ordre n , on obtient une bonne approximation sur un disque dans le plan complexe. Si on veut approcher la même fonction par un polynôme de degré n sur un segment, le développement de celle-ci en série sur une base des polynômes de Tchebychev donnera la meilleure approximation [5, Section 3.5.1].



Calculer la récurrence vérifiée par les coefficients de cette série a un sens car la notion de D-finitude d'une fonction implique une relation de récurrence entre les coefficients de cette fonction développée dans la base de polynômes de Tchebychev [16, proposition 22]. Par la suite je nommerai cette récurrence la récurrence de Tchebychev.

On peut définir le n ème polynôme de Tchebychev, T_n , comme l'unique polynôme vérifiant

$$T_n(\cos(x)) = \cos(nx).$$

Cette égalité permet de construire les premiers polynômes

$$T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, \dots \tag{5}$$

Comme toutes les familles de polynômes orthogonaux classiques, cette famille vérifie des relations de récurrences [1, chapitre 22], dont les deux suivantes :

$$2xT_n = T_{n+1} + T_{n-1} \tag{6}$$

$$(1 - x^2)T'_n = \frac{n}{2}(T_{n+1} - T_{n-1}), \quad (7)$$

qui sont les analogues de (1) et (2) pour la base monomiale. Il existe quand même une différence entre (2) et (7) : le polynôme $1 - x^2$ en facteur devant la dérivée. Pour calculer la récurrence de Tchebychev d'une solution d'une équation différentielle, une première méthode consiste à multiplier l'équation par $1 - x^2$ pour faire apparaître l'opérateur différentiel $(1 - x^2)\frac{d}{dx}$ qui s'exprime simplement en terme de récurrence par la relation (7).

Cette méthode permet de calculer des identités remarquables comme :

$$f(x) = \arctan\left(\frac{x}{2}\right) = \sum_{n=0}^{+\infty} (-1)^n \frac{(2 - \sqrt{3})^{2n+1}}{2n+1} T_{2n+1}(x). \quad (8)$$

Cette fonction vérifie l'équation différentielle suivante :

$$(4 + x^2)y'' + 2xy' = 0 \quad (9)$$

En multipliant par $(1 - x^2)$ l'équation différentielle et en appliquant l'égalité (7) à la série formelle $f := \sum t_n T_n(x)$, on obtient :

$$\begin{aligned} 0 &= t_n(x^2 + 4)(1 - x^2)T''_n(x) + 2 \sum t_n(1 - x^2)T'_n(x) \\ &\quad \sum t_n(x^2 + 4) \left((1 - x^2)T'_n(x) \right)' + \sum t_n(x^2 + 4)2xT'_n(x) + 2 \sum t_n(1 - x^2)T'_n(x) \\ &\quad \sum t_n(x^2 + 4)\frac{n}{2} \left(T'_{n+1}(x) - T'_{n-1}(x) \right) + \sum t_n(x^2 + 4)2xT'_n(x) + 2 \sum t_n\frac{n}{2} (T_{n+1}(x) - T_{n-1}(x)) \\ &\quad \sum (x^2 + 4) \left(\frac{n}{2}t_{n-1} - \frac{n}{2}t_{n+1} + 2xt_n \right) T'_n(x) + 2 \sum \left(\frac{n}{2}t_{n-1} - \frac{n}{2}t_{n+1} \right) T_n(x). \end{aligned}$$

En remultipliant par $(1 - x^2)$ et pour transformer la dernière dérivée avec (7) et en appliquant la relation (6) pour enlever les x , on obtient une récurrence d'ordre 12 entre les t_n :

On a ici une récurrence d'ordre 12 que l'algorithme de Petkovsek [15] permet de résoudre afin de retrouver (8). On peut montrer que les coefficients vérifient aussi une récurrence d'ordre 4 :

$$nt_n + 18(n+2)t_{n+2} + (n+4)t_{n+4} = 0.$$

Cette différence d'ordre nous propose un objectif sur les algorithmes : l'obtention de la récurrence d'ordre minimal. Pour obtenir cet ordre minimal, une solution est de réécrire l'égalité (7) comme

$$T'_n = \frac{n}{2(1-x^2)}T_{n+1} - \frac{n}{2(1-x^2)}T_{n-1}. \quad (10)$$

On écrit ensuite des opérateurs de récurrences X et D associés aux égalités (6) et (10),

$$X := \frac{S + S^{-1}}{2} \text{ et } D := (2(1 - X^2))^{-1}(-S + S^{-1})n, \quad (11)$$

où S est l'opérateur de récurrence qui appliqué à la suite u_n donne $S \cdot u_n = u_{n+1}$. Par exemple $X \cdot u_n = 1/2(u_{n+1} + u_{n-1})$. On a les égalités suivantes :

$$xf(x) = \sum_{n=0}^{+\infty} t_n x T_n(x) = \sum_{n=0}^{+\infty} (X \cdot t_n) T_n(x) \text{ et } f'(x) = \sum_{n=0}^{+\infty} t_n T'_n(x) = \sum_{n=0}^{+\infty} (D \cdot t_n) T_n(x). \quad (12)$$

Le gain d'ordre de récurrence s'obtient grâce à l'égalité

$$1 - X^2 = -1/4 (S^2 + S^{-2} - 2) = -1/4(S^{-1} - S)^2.$$

Par simplification on a donc

$$D = (S^{-1} - S)^{-1}(2n). \quad (13)$$

On transforme donc l'équation différentielle (9), en l'opérateur de récurrence

$$L := (X^2 + 4)D^2 + 2xD = (2(S^{-1} - S))^{-1}n)^2 (S^2 + S^{-2} + 18)n + 2(S + S^{-1})(S^{-1} - S)^{-1}n.$$

En mettant au même dénominateur la fraction rationnelle non commutative L , on peut exhiber un numérateur qui est un opérateur de récurrence d'ordre 4 : $(t_n)_{n \in \mathbb{Z}}$:

$$(n + 2)S^2 + 18n + (n - 2)S^{-2}. \quad (14)$$

Je montre dans la section 1 que cet opérateur annule la suite t_n .

Cet exemple de fraction rationnelle de récurrence ouvre la voie à une autre vision des opérateurs de récurrences que l'on représentait jusque là comme des polynômes. Ore [12, 13] donne une structure mathématique à ces fractions rationnelles, qui sera développée dans la section 1.

Travaux antérieurs

À ma connaissance, Clenshaw [4] a été le premier à donner une méthode pour calculer les coefficients d'une série de Tchebychev à partir d'une équation différentielle. Il ne donne pas les récurrences vérifiées par ceux-ci, mais les idées qu'il utilise permettent de les calculer. Lewanowicz [8] a été très actif sur le calcul de récurrences, il a repris l'algorithme donné par Paszkowski [14] dans son livre en travaillant notamment sur la minimalité de la récurrence obtenue; il propose plusieurs algorithmes et généralise le calcul des récurrences à tous les polynômes de Jacobi [9, 10]. Rebillard dans sa thèse [16] donne un algorithme pour le calcul de cette récurrence.

Plan

La première section de ce rapport est consacrée à l'algorithmique dans des corps de fractions rationnelles de récurrences. Les opérations sur les opérateurs de récurrence sont rappelées ainsi que la complexité des algorithmes pour la multiplication, division, calcul de PGCD et PPCM. Cette dernière opération joue un rôle clé dans les calculs sur les fractions non commutatives qui sont utilisées par la suite. J'ai implanté en Maple ces calculs à partir de packages déjà existants, OreTools et OreAlgebra; j'ai aussi implanté la multiplication rapide de polynômes tordus de van der Hoeven [19] que je détaille dans cette section.

La deuxième section est dédiée aux développements d'opérateurs dans la base des polynômes de Tchebychev. Un de mes principaux apports dans cette section est de retravailler l'article de Lewanowicz [8], en refaisant les preuves de façon je l'espère beaucoup plus claire et beaucoup plus simple au moyen des corps de fractions rationnelles de Ore. Dans son article, Lewanowicz donne un algorithme qui calcule les récurrences minimales et rappelle l'algorithme de Paszkowski certainement plus simple mais qui n'obtient pas la minimalité. Rebillard dans sa thèse [16] donne un algorithme proche de celui de Paszkowski, et obtient dans certains

cas la récurrence minimale en donnant une méthode de factorisation de récurrence. Je donne dans cette section un algorithme qui est plus efficace que ceux de Lewanowicz, Paszkowski et Rebillard ; cet algorithme est en outre capable de donner une récurrence minimale dans tous les cas sans perdre en complexité.

1 Algorithmique des corps de fractions rationnelles de récurrences

Dans cette section, la première partie est consacrée aux anneaux des polynômes tordus. Après une brève introduction générale sur ces anneaux, les opérateurs de récurrences sont exclusivement étudiés. Les opérations sur ces opérateurs sont rappelées ainsi que leurs complexités. Dans la seconde partie la construction des corps des fractions rationnelles de récurrences qui sont nommés corps de Ore de récurrences est détaillée ; les opérations de base comme l'addition ou la multiplication y sont décrites. La théorie des corps de Ore de récurrences joue un rôle crucial dans la suite de mon travail, toutes les preuves d'algorithmes dans les sections suivantes utilisent cette notion.

Cette section ne comporte pas de nouveauté sauf les complexités des opérations dans les corps de Ore de récurrences de la partie 1.2.

1.1 L'anneau des opérateurs de récurrence

Je donne ici une approche très brève des opérateurs de récurrence. Sauf pour la partie sur la complexité, on peut trouver des détails plus approfondis dans le polycopié [2, page 195].

Définition d'un anneau de Ore

Soit \mathbb{K} un corps commutatif de caractéristique zéro, que nous supposons muni d'un endomorphisme injectif σ et d'une σ -dérivation δ , au sens où pour tout a et tout b de \mathbb{K} ,

$$\sigma(a + b) = \sigma(a) + \sigma(b), \quad \sigma(ab) = \sigma(a)\sigma(b), \quad \delta(ab) = \sigma(a)\delta(b) + \delta(a)b.$$

Pour une nouvelle variable ∂ , on appelle anneau de polynômes tordus l'algèbre sur \mathbb{K} engendrée par ∂ et les relations, pour tout a de \mathbb{K} ,

$$\partial a = \sigma(a)\partial + \delta(a).$$

On note cet anneau $\mathbb{K}\langle\partial; \sigma, \delta\rangle$.

Exemples

Deux exemples illustrent les polynômes de Ore :

- $\mathbb{K}(x)\langle\partial; I, D\rangle$, l'algèbre des opérateurs différentiels linéaires,
- $\mathbb{K}(n)\langle\partial; S, 0\rangle$, l'algèbre des différences finies, où S agit sur une suite u_n par $S \cdot u_n = u_{n+1}$.

Dans la suite de ce rapport, on ne s'intéresse qu'au cas des opérateurs aux différences finies. On note un opérateur de récurrence P d'ordre k :

$$P = \sum_{i=0}^k p_i(n)S^i, \tag{15}$$

où les p_i sont des éléments de $\mathbb{K}(n)$. P peut être vu comme un polynôme de degré k en S .

La multiplication de deux opérateurs de récurrences est effectuée par la commutation $S^i p(n) = p(n+i)S^i$ pour toute fraction rationnelle $p(n)$. Un algorithme de multiplication de deux opérateurs renvoie toujours un opérateur de la forme (15), normalisé avec les monômes en S à droite.

Division Euclidienne d'opérateurs de récurrence

Lorsque l'on multiplie deux opérateurs de récurrence, le degré du produit est la somme des degrés des deux opérateurs. Dans les anneaux de polynômes commutatifs, ce résultat permet de définir un stathme et rend l'anneau euclidien. Avec les opérateurs de récurrence, le même stathme peut être défini et une division euclidienne entre deux polynômes peut être construite. On en déduit que l'anneau $\mathbb{K}(n)\langle S \rangle$ est euclidien. Il existe deux façons de diviser deux opérateurs A et B , on peut les diviser à gauche ou à droite. Par exemple si $A := nS^2 - n$ et $B := S + 1$, en effectuant une division euclidienne à droite on a :

$$A = n(S - 1)B$$

en l'effectuant à gauche, on a :

$$A = B(S - 1) - 2,$$

B divise donc A à droite mais pas à gauche ce qui constitue la principale différence avec les divisions entre polynômes commutatifs. Je donne ici l'algorithme de division à droite entre A d'ordre k_A et B d'ordre k_B , c'est le même que dans le cas commutatif. La notation $\text{coeff}(B, d_B)$

Algorithme 1 Division euclidienne à droite

ENTRÉES: $A, B \in \mathbb{K}(n)\langle S \rangle$

SORTIES: $R, Q \in \mathbb{K}(n)\langle S \rangle$

$i := k_A - k_B, Q := 0, R := A$

tantque $i \neq 0$ **faire**

$Q := Q + \text{coeff}(R, i + k_B) S^i \frac{1}{\text{coeff}(B, k_B)}$

$R := R - \text{coeff}(R, i + k_B) S^i \frac{1}{\text{coeff}(B, k_B)} B$

$i := i - 1$

fin tantque

représente le coefficient de B de degré d_B en S . C'est une fraction rationnelle en n .

Algorithme d'Euclide

L'algorithme d'Euclide pour des opérateurs de récurrences découle de la division euclidienne, comme dans les anneaux commutatifs. Le dernier reste non nul est le PGCD des deux polynômes. Je décris ici l'algorithme d'Euclide pour le PGCD à droite. $A \text{ modd } B$ est le reste de la division euclidienne à droite de A par B . Pour la division à droite la sortie de cet algorithme est le PGCD à droite que je noterai pgcd_d . De la même façon que dans le cas commutatif, l'algorithme d'Euclide étendu à droite permet de calculer les coefficients de Bézout et un PPCM cette fois-ci à gauche que je noterai ppcm_g .

L'algorithme d'Euclide à gauche s'écrit de manière similaire, on en déduit le PGCD à gauche et le PPCM à droite.

Algorithme 2 Algorithme d'Euclide à droite

ENTRÉES: $A, B \in \mathbb{K}[n]\langle S \rangle$ **SORTIES:** $R_0 \in \mathbb{K}[n]\langle S \rangle$ $R_0 := B$ $R_1 := A \text{ modd } B$ **tantque** $R_1 \neq 0$ **faire** $R_2 := R_0$ $R_0 := R_1$ $R_1 := R_2 \text{ modd } R_0$ **fin tantque**

Complexité des algorithmes naïfs

Pour l'analyse de complexité des algorithmes, on se place dans un modèle où seules comptent les opérations arithmétiques sur le corps \mathbb{K} . Effectuer une addition, une multiplication ou une division entre deux éléments sur \mathbb{K} a donc un coût unitaire. La taille d'un objet est le nombre d'éléments sur le corps \mathbb{K} qu'il comporte. Par exemple si $A \in \mathbb{K}[n]\langle S \rangle$, alors sa taille sera bornée par le maximum des degrés en n plus 1 multiplié par l'ordre de l'opérateur en S .

On prendra comme entrée dans chaque algorithme des opérateurs de récurrence à coefficients polynomiaux en n .

Le premier algorithme à analyser est l'algorithme de multiplication naïf. Si on prend comme entrée deux polynômes A et B dans $\mathbb{K}[n]\langle S \rangle$, chacun de bidegré (k, d) en n et S , on a naïvement $k^2 d^2$ opérations arithmétiques, alors que la sortie de l'algorithme est le polynôme AB qui est de bidegré $(2k, 2d)$. En fait la complexité de cet algorithme est ici la même qu'avec la multiplication naïve de polynômes bivariés dans des anneaux commutatifs.

Des exemples classiques de complexités en calcul formel sont les suivants :

1. L'algorithme naïf pour multiplier deux polynômes de degré k effectue $O(k^2)$ opérations arithmétiques. Avec de l'algorithmique rapide, on sait descendre à une complexité quasi-linéaire en k , c'est-à-dire que le nombre d'opérations à effectuer est linéaire en k à des logarithmes près. On note $M(k)$ le nombre d'opérations arithmétiques pour multiplier deux polynômes de degré k [2, cours 2].
2. L'algorithme naïf pour multiplier deux matrices de tailles (k, k) effectue $O(k^3)$ opérations arithmétiques, on peut faire descendre cet exposant à un nombre strictement plus petit que 3, on appelle cette constante ω . On considérera toujours que la constante ω est strictement supérieure à 2. On considérera toujours que $M(k) \in O(k^\omega)$ [2, cours 3].

Proposition 1.1. *Si l'entrée de l'algorithme 1 est un couple d'opérateurs (A, B) dans $\mathbb{K}[n]\langle S \rangle$ de bidegré (k_A, d_A) pour A et (k_B, d_B) pour B , alors le nombre d'opérations arithmétiques pour calculer la division est de $O(k_A^3 \sup(d_A, d_B))$.*

Preuve Avant de calculer le nombre d'opérations, on détermine d'abord la taille des objets à l'étape i de l'algorithme. L'ordre des opérateurs R et Q se détermine facilement par la nature de l'algorithme, R est un opérateur d'ordre au plus $k_B + i$ et Q d'ordre $k_A - k_B$. Les opérateurs Q et R sont à coefficients des fractions rationnelles en n dont le numérateur et le dénominateur sont de mêmes degrés, ces degrés sont majorés par $(k_A - k_B - i) \sup(d_B, d_A)$. Le coût qui dominera à l'étape i sera la mise au même dénominateur, lors des calculs de Q et

R . Pour cette opération, on devra multiplier le dénominateur de $\text{coeff}(R, i + dB)S^i \frac{1}{\text{coeff}(B, dB)}$ par le numérateur de Q et de R . Pour R , on doit multiplier $k_B + i$ polynômes de degré $(k_A - k_B - i)d_B$ entre eux. On doit le faire $k_A - k_B$ fois, le nombre d'opérations à effectuer est donc un $O((k_A - k_B)^2 \sup(d_A, d_B)k_A)$. La complexité de la proposition s'en déduit. \square

Si les bidegrés de A et B sont $(2k, 2k)$ et (k, k) , on a donc un algorithme de complexité $O(k^4)$, pour un résultat de taille $O(k^3)$.

La multiplication rapide

Joris van der Hoeven [19] donne un algorithme rapide de multiplication de polynômes tordus. Il décrit brièvement la multiplication dans le cas d'opérateurs de récurrence, que je détaille dans ce rapport.

Le résultat du théorème suivant est important pour l'étude de complexité des algorithmes dans la section 2.

Théorème 1.2. *Soient A et B deux polynômes dans $\mathbb{K}[n]\langle S \rangle$ de bidegré (k, k) , le produit AB peut être calculé en $O(k^\omega)$ opérations.*

Preuve Le principe de l'algorithme de multiplication rapide est à base d'évaluation et interpolation. On considère ici les polynômes en n et S comme des opérateurs en n à coefficients polynomiaux en S qu'on fait agir sur des polynômes en S avec l'action suivante, $n^i(S^j) = (-j)^i S^j$. Si $A = \sum_{i=0}^k p_i(n)S^i$, on a :

$$A(S^j) = \sum_{i=0}^k S^i p_i(n-i)(S^j) = \sum_{i=0}^k S^i p_i(-i-j)S^j = \sum_{i=0}^k p_i(-i-j)S^{i+j}.$$

Il est facile de voir que l'opérateur A est une application linéaire de $\mathbb{K}[S]$ dans $\mathbb{K}[S]$. On peut donc le représenter par une matrice M_A à coefficients dans \mathbb{K} qui agit sur un vecteur colonne représentant un polynôme en S écrit dans la base $(1, S, \dots, S^i, \dots)$ de l'espace vectoriel $\mathbb{K}[S]$. Si on fait opérer A sur un polynôme de degré i , le polynôme obtenu est de degré $i+k$, on peut donc se contenter de restreindre l'application A à l'espace vectoriel $\mathbb{K}_i[S]$ des polynômes de degrés inférieurs ou égaux à i . L'opérateur A est de degré k , $A|_{\mathbb{K}_i[S]}$ est donc une application linéaire de $\mathbb{K}_i[S]$ dans $\mathbb{K}_{i+k}[S]$, $M_{A|_{\mathbb{K}_i[S]}}$ est alors une matrice de $i+1$ colonnes et $i+k+1$ lignes. Je noterai par la suite

$$M_{A|_{\mathbb{K}_i[S]}} = M_{A_i}.$$

Par exemple, pour évaluer A écrit comme précédemment sur le polynôme $l := a_0 + a_1 S + \dots + a_i S^i$, On écrit l comme vecteur colonne V_l sur la base $(1, \dots, S^i)$. On a

$$A := \sum_{i=0}^k p_i(n)S^i \text{ et } M_{A_i} := \begin{pmatrix} p_0(0) & & & & 0 \\ \vdots & p_0(-1) & & & \\ p_k(-k) & \vdots & \ddots & & \\ & p_k(-k-1) & & p_0(-k-1) & \\ & & \ddots & \vdots & \\ 0 & & & & p_k(-k-i) \end{pmatrix}. \quad (16)$$

Si on connaît le degré k en n de l'opérateur A , on peut reconstruire cet opérateur à partir de sa matrice M_{A_k} par interpolation des p_i .

On utilise ensuite une propriété des opérateurs linéaires :

$$\forall l \in \mathbb{K}_i[S], M_{AB_{i+k}} V_l := M_{A_{i+k}} M_{B_i} V_l.$$

Comme on connaît le degré en n des deux opérateurs A et B qui est de $2k$, on en déduit un algorithme de multiplication entre A et B :

Algorithme 3 Multiplication rapide d'opérateurs

ENTRÉES: $A, B \in \mathbb{K}[n]\langle S \rangle$

- 1 Construction de $M_{A_{2k}}$ et M_{B_k}
 - 2 Calcul $M_{A_{2k}} M_{B_k}$
 - 3 Calcul AB
-

L'étape 1 est l'évaluation de $3k + 2$ polynômes de degré k en au plus $3k + 1$ points, le coût asymptotique de cette étape est de $O(kM(k) \log(k))$ opérations arithmétiques [2, cours 9].

L'étape 3 est l'interpolation de $2k + 1$ polynômes de degré $2k + 1$, le coût asymptotique est de $O(kM(k) \log(k))$ opérations arithmétiques [2, cours 9].

L'étape 2 est le calcul dont le coût va dominer. $M_{A_{2k}}$ est une matrice de $3k + 1$ lignes et $2k + 1$ colonnes et M_{B_k} est une matrice de $2k + 1$ lignes et $k + 1$ colonnes. Le coût de la multiplication de ces deux matrices est donc de $O(k^\omega)$ opérations arithmétiques. □

PGCD et PPCM rapide

Theorème 1.3 (Grigor'ev [7]). *Soit P et $Q \in \mathbb{K}[n]\langle S \rangle$ de bidegré (k, k) , le PGCD entre les deux polynômes se calcule en $O(k^{\omega+1})$ opérations arithmétiques.*

Grigor'ev ramène le calcul du PGCD au calcul d'un déterminant de matrice de $2k$ lignes et $2k$ colonnes à coefficients des polynômes de degrés k . Un article plus récent de Giorgi *et alii* [6] ramène le calcul de ce déterminant à une complexité $O(k^{\omega+1})$.

Theorème 1.4 (Li et Nemes [11]). *Soit P et $Q \in \mathbb{K}[n]\langle S \rangle$ de bidegré (k, k) , le PPCM entre les deux polynômes se calcule en $O(k^{\omega+1})$ opérations arithmétiques. Les cofacteurs sont aussi calculés dans cette complexité.*

Ici encore les auteurs ramènent le calcul des cofacteurs d'un PPCM à un déterminant de matrice à coefficients polynomiaux, [6] permet de conclure.

1.2 Polynômes de Laurent

Comme dans le cas commutatif, on peut définir des polynômes de Laurent non commutatifs. Dans le cas des opérateurs de récurrences, cette définition a un sens car S^{-1} commute naturellement avec n par : $S^{-1}n = (n - 1)S^{-1}$.

Définition Soit B un opérateur de Laurent de récurrence :

$$B := \sum_{i=-k_1}^{k_2} p_i(n)S^i.$$

On appelle *ordre* de B , la différence $k_2 + k_1$.

Je noterai dans la suite l'anneau des polynômes de Laurent $\mathbb{K}(n)\langle S, S^{-1} \rangle$.

Théorème 1.5 ([2]). *L'anneau $\mathbb{K}(n)\langle S, S^{-1} \rangle$ est un anneau euclidien.*

Les algorithmes présentés sur $\mathbb{K}(n)\langle S \rangle$ sont tous adaptables avec des opérateurs de Laurent, les complexités restent les mêmes.

1.3 Corps des fractions rationnelles de récurrences

Définition et algorithmes pour les opérations de base dans les corps de Ore

Oystein Ore [12] montre l'existence du corps des fractions rationnelles d'un anneau de polynômes tordus lorsque le ppcm_d est défini comme ci-dessus.

Théorème 1.6 ([12]). *Soit $\mathbb{K}(n)\langle S, S^{-1} \rangle$, il existe un corps que je noterai $\mathbb{K}(n)(S, S^{-1})$ tel qu'un sous-anneau de $\mathbb{K}(n)(S, S^{-1})$ soit isomorphe à $\mathbb{K}(n)\langle S, S^{-1} \rangle$. Il vérifie aussi la propriété que tout sous-corps contenant ce sous-anneau soit $\mathbb{K}(n)(S, S^{-1})$ lui-même.*

Le corps $\mathbb{K}(n)(S, S^{-1})$ est le corps des fractions rationnelles de $\mathbb{K}(n)\langle S, S^{-1} \rangle$. Les éléments de $\mathbb{K}(n)(S, S^{-1})$ sont les couples $(A, B) \in \mathbb{K}(n)\langle S, S^{-1} \rangle \times \mathbb{K}(n)\langle S, S^{-1} \rangle^*$. Un couple (A, B) est appelé quotient de A et B .

Notation Soient $A, B \in \mathbb{K}(n)\langle S, S^{-1} \rangle$, le quotient de A et B est noté

$$\frac{A}{B} \text{ ou } B^{-1}A.$$

La relation d'équivalence entre deux quotients $B^{-1}A$ et $D^{-1}C$ est :

$$B^{-1}A = D^{-1}C \Leftrightarrow B_1A = D_1C$$

où B_1 et D_1 sont les cofacteurs du $\text{ppcm}_g(B, D)$, c'est-à-dire, $B_1B = D_1D$.

Exemple Les quotients

$$B^{-1}A := \frac{nS + n}{nS^2 - n} \text{ et } D^{-1}C := \frac{1}{S - 1} \quad (17)$$

sont équivalents. En effet, $B = n(S + 1)(S - 1)$, on a donc $B_1 = 1$ et $D_1 = n(S + 1)$, l'égalité $B_1A = D_1C$ est bien vérifiée.

Définition Les opérations sur ce corps sont définies par

$$\frac{A}{B} + \frac{C}{D} = \frac{D_1A + B_1C}{B_1B} \text{ et } \frac{A}{B} \cdot \frac{C}{D} = \frac{UC}{VB},$$

où $\text{ppcm}_g(D, A) = UD = VA$.

Les opérations sur $\mathbb{K}(n)(S, S^{-1})$ sont compatibles avec la relation d'équivalence.

Exemple Soient $B^{-1}A := \frac{S}{nS^2 - n}$ et $D^{-1}C := \frac{1}{(S+1)}$, l'addition et la multiplication de ces deux quotients sont :

$$\frac{A}{B} + \frac{C}{D} = \frac{(n+1)S - n}{nS^2 - 1} \text{ et } \frac{A}{B} \cdot \frac{C}{D} = \frac{S}{(n+1)S^3 + nS^2 - (n+1)S - n}$$

1.3.1 Fractions irréductibles

Comme dans le cas commutatif, on montre l'existence d'une représentation irréductible d'une fraction rationnelle.

Proposition 1.7. *Soit Q un quotient, il existe un unique couple A et B à inversible dans $\mathbb{K}(n)\langle S, S^{-1} \rangle$ près tel que $\text{pgcd}_g(A, B) = 1$ et $Q = \frac{A}{B}$. La représentation $\frac{A}{B}$ est appelé forme irréductible de Q .*

Preuve Soient deux couples A, B et C, D tels que $\frac{A}{B} = \frac{C}{D} = Q$ et $\text{pgcd}_g(A, B) = \text{pgcd}_g(C, D) = 1$. Soient $N = UC = VA$ le ppcm $_g$ de A et C et $M = VB = UD$ le ppcm $_g$ de B et D . On a alors la suite d'égalités suivante :

$$V = V\text{pgcd}_g(A, B) = \text{pgcd}_g(VA, VB) = \text{pgcd}_g(N, M) = \text{pgcd}_g(UC, UD) = U\text{pgcd}_g(C, D) = U.$$

De l'égalité des extrémités, on a bien $B = D$ et $A = C$ car l'anneau est intègre. \square

Exemple Dans l'exemple (1.3), $D^{-1}C$ est une fraction irréductible.

Lewanowicz [8] calcule les récurrences de Tchebychev, d'ordres minimaux. Je montre dans la section suivante le rapport entre ces récurrences minimales et des fractions irréductibles. Le lemme et le corollaire suivant sont les points principaux des preuves de minimalité de l'algorithme de Lewanowicz que je développe dans les sections 2.3 et 2.4.

Lemme 1.8. *Soient $Q = B^{-1}A$ une fraction irréductible et P un polynôme en S à coefficients dans $\mathbb{K}(n)$, alors $PQ = V^{-1}UA$ où $\text{ppcm}_g(P, B) = UB = VP$. Cette fraction est irréductible.*

Preuve On a $\text{pgcd}_d(U, V) = 1$ par définition d'un ppcm (sinon UB serait divisible à gauche par $\text{pgcd}_d(U, V)$ et ne serait donc plus le ppcm).

Soit g tel que g divise à gauche V et UA . Le polynôme g divise donc à gauche $VP = UB$. On a donc $g|_g\text{pgcd}_g(UA, UB) = U\text{pgcd}_g(B, A) = U$, d'où l'égalité $g = 1$. \square

Corollaire 1.9. *Soient $Q = B^{-1}A$ une fraction irréductible et P un polynôme en S à coefficients dans $\mathbb{K}(n)$, alors $QP^{-1} = (VB)^{-1}U$ où $\text{ppcm}_g(P, A) = VA = UP$ est une fraction irréductible.*

Complexités des opérations de base

Theorème 1.10. *Soient $B^{-1}A$ et $C^{-1}D$ deux fractions de Ore, où A, B, C et $D \in \mathbb{K}[n]\langle S \rangle$ de bidegrés (k, k) . L'addition et la multiplication de ces deux fractions peut être effectuée en $O(k^{\omega+1})$ opérations.*

La démonstration de ce théorème se déduit facilement du théorème 1.4.

Pour la forme irréductible, le PGCD s'effectue aussi en $O(k^{\omega+1})$ opérations, on peut donc vérifier si une fraction est irréductible en cette complexité. Par contre pour rendre une fraction irréductible, on doit effectuer une division euclidienne, opération qui est plus coûteuse.

Fractions de récurrences

On définit maintenant l'action de $\mathbb{K}(n)\langle S, S^{-1} \rangle$ sur l'ensemble des suites. Si on se donne une fraction rationnelle de récurrence $Q = B^{-1}A$ et une suite $(u_n)_{n \in \mathbb{Z}}$ à coefficients dans \mathbb{K} , l'idée naturelle est de définir une action par l'application $Q \cdot u_n = v_n$, où v_n est définie par $A \cdot u_n = B \cdot v_n$. Cette opération n'est pas bien définie, notamment car v_n n'est pas unique. On veut aussi que l'action soit compatible avec la multiplication de fractions rationnelles.

On fait donc agir les fractions rationnelles de récurrences sur une classe de suite à coefficients sur un corps \mathbb{K} . On définit cette classe de suites par la relation d'équivalence suivante.

Définition Soient $(u_n)_{n \in \mathbb{Z}}$ et $(v_n)_{n \in \mathbb{Z}}$ deux suites à coefficients dans \mathbb{K} et $P \in \mathbb{K}(n)\langle S, S^{-1} \rangle$, on définit la relation d'équivalence \sim_P par

$$u_n \sim_P v_n \text{ si } P \cdot (u_n - v_n) = 0.$$

Si u_n et v_n sont équivalents à v_n , on dira que u_n est dans la classe de v_n modulo P .

On définit maintenant l'action de Q sur une classe de suites.

Définition Soient $Q := B^{-1}A$ une fraction rationnelle de récurrence, $P \in \mathbb{K}[n]\langle S, S^{-1} \rangle$ et une classe u modulo P . Soient A_1 et P_1 tels que $A_1A = P_1P$. On a $Q \cdot u := v$, tel que v est la classe modulo A_1B des suites vérifiant $P_1 \cdot u_n = A_1B \cdot v_n$.

Si on prend $P = 1$, l'action définie ci-dessus correspond à l'idée naturelle du début du paragraphe.

Exemple Si on prend $P = 1$, et $Q := B^{-1}A = (S + 1)^{-1}((n - 2)S^2 - n)$, si $Q \cdot u_n = 0$ alors on a $A \cdot u_n = B \cdot 0 = 0$. La suite u_n vérifie l'égalité $(n - 2)u_{n+2} - nu_n = 0$.

Définition Soit $Q = B^{-1}A$ une fraction de récurrence, on appelle A un opérateur de récurrence associé à Q .

Cette définition correspond à l'exemple précédent si $Q \cdot u_n = 0$ alors $A \cdot u_n = 0$.

La proposition suivante se déduit tout de suite de la notion de fraction irréductible.

Theorème 1.11. *Soit Q une fraction de récurrence. L'opérateur de récurrence d'ordre minimale associé à Q est le numérateur de la représentation irréductible de Q .*

Exemple De l'exemple précédent, une récurrence associée à $Q = B^{-1}A$ est $(n - 2)u_{n+2} - nu_n = 0$. Mais on a aussi $Q = (S + 1)^{-1}(S + 1)(S - 1)n = (S - 1)n = (n + 1)S - n$, donc $(n + 1)u_{n+1} - n$ est la récurrence minimale associée à Q .

2 Développements en séries de Tchebychev

Cette section traite des récurrences de Tchebychev. Charles William Clenshaw [4] calcule numériquement les coefficients de Tchebychev qu'il compare avec d'autres algorithmes de calcul de coefficients de séries de Fourier. Il ne calcule pas les récurrences des coefficients de Tchebychev mais donne les idées principales des algorithmes de récurrences qui viendront après.

Dans cette section on développe des algorithmes qui prennent en entrée une équation différentielle linéaire et qui renvoient en sortie un opérateur de récurrence qui annule les coefficients des séries développées dans la base des polynômes de Tchebychev. J'énumère les différents algorithmes pour le calcul de cette récurrence d'une manière chronologique.

2.1 La méthode de Clenshaw

Dans la suite de cette section deux récurrences sur les polynômes de Tchebychev sont utilisés :

$$2xT_n(x) = T_{n+1}(x) + T_{n-1}(x) \text{ et } (1 - x^2)T'_n(x) = +2nT_{n+1}(x) - 2nT_{n-1}(x).$$

De la première égalité, Clenshaw déduit une récurrence sur les coefficients de la série de Tchebychev. Si $f := \sum_{n \geq 0} c_n T_n(x)$, on a

$$xf(x) = \sum_{n \geq 0} \left(\frac{1}{2}c_{n+1} + \frac{1}{2}c_{n-1} \right) T_n(x).$$

De cette égalité, on peut définir l'opérateur $X \in \mathbb{K}(n)\langle S, S^{-1} \rangle$:

$$X := \frac{1}{2}(S + S^{-1}). \quad (18)$$

La seconde égalité de récurrence ne donne pas directement un opérateur de Laurent pour la dérivée. Clenshaw utilise une autre égalité de récurrence [18]

$$2 \int T_n(x) = \frac{1}{n+1}T_{n+1} - \frac{1}{n-1}T_{n-1}.$$

De cet égalité, on peut définir un opérateur de récurrence de primitive :

$$P := \frac{S^{-1} - S}{2n}. \quad (19)$$

Les corps de fractions rationnelles de récurrence donnent une autre façon de voir les choses. Avec l'égalité sur la dérivée, on peut définir un opérateur sur $\mathbb{K}[n](S, S^{-1})$ pour la dérivée :

$$D := (1 - X^2)^{-1}(-S^2 + 1)n. \quad (20)$$

Une représentation irréductible de cet opérateur est :

$$D = \frac{2n}{S^{-1} - S}. \quad (21)$$

On retrouve l'inverse de l'opérateur P donné par Clenshaw. L'opérateur de dérivée est donc l'inverse d'un polynôme. Cette propriété sera largement utilisée dans les algorithmes qui vont suivre.

2.2 L'algorithme de Paszkowski

Lewanowicz [8] rappelle l'algorithme de Paszkowski [14] pour calculer la récurrence de Tchebychev en signalant qu'elle n'est pas d'ordre minimal.

La base de cet algorithme dans le cas des polynômes de Tchebychev est que l'opérateur D est l'inverse d'un polynôme et que l'on peut exprimer ses puissances de façon simple.

Lemme 2.1. *Pour tout $i \in \mathbb{N}^*$,*

$$D^{-i} = \frac{1}{2n} \left(\prod_{k=1}^{i-1} \frac{1}{(n-k)(n+k)} \right) \left((n+1)_{(i-1)} S^{-i} + (n-1)_{-(i-1)} S^i \right. \\ \left. + \sum_{k=1}^{i-1} (-1)^k \binom{i}{k} (n-i+2k)(n+k+1)_{(i-1-k)} (n-i+k-1)_{-(k-1)} S^{-i+2k} \right) \quad (22)$$

où (i) est le symbole de Pochhammer et $n_{-(i)} = n(n-1)\dots(n-i+1)$.

La preuve est technique mais simple, il suffit de le démontrer par récurrence sur i .

Paszkowski utilise cette formule pour définir l'opérateur $B_i = (n-i)_{2i+1} D^{-i}$. Il donne un algorithme direct pour le calcul des polynômes $q_i(x)$ tel que

$$\sum_{i=0}^k p_i(x) \left(\frac{d}{dx} \right)^i = \sum_{i=0}^k \left(\frac{d}{dx} \right)^i q_i(x). \quad (23)$$

J'utiliserai pour la suite un algorithme quasi-optimal, pour cette opération tirée d'un article en préparation [3] qui en ramène la complexité à $O((k+d)M(k))$ opérations.

On obtient un opérateur $\sum_{i=0}^k D^i q_i(X)$. Connaissant les fractions de récurrence sur les corps de Ore, pour en déduire la récurrence associée, il suffit de calculer le numérateur de cette fraction rationnelle. En mettant au même dénominateur, on obtient :

$$\sum_{i=0}^k D^i q_i(X) = B_k^{-1} \left((n-k)_{2k+1} q_k(X) + \sum_{i=1}^k (n+k-i)_{(i)} (n-k+i)_{-(i)} B_{k-i} q_i(X) \right).$$

L'algorithme de Paszkowski calcule ce numérateur.

Algorithme 4 Paszkowski Tchebychev Récurrence

ENTRÉES: seq($q_i, i = 0..k$)

SORTIES: L tel que $L(c_n) = 0$

$L := nq_k(X)$

pour tout i de 1 à k **faire**

$L := (n+i)(n-i)L + B_i q_{k-i}(X)$

fin pour

Theorème 2.2. *L'algorithme de Paszkowski effectue $O(dk^3)$ opérations, où d est le maximum des degrés des p_i .*

Preuve À l'étape i , le calcul qui domine en complexité est la multiplication de B_i par q_{k-i} . B_i est un polynôme sur $\mathbb{K}[n]\langle S, S^{-1} \rangle$ de bidegré $(2i, 2i)$ en n et S , et q_k est un polynôme de bidegré $(0, 2d)$ en n et S . Par une multiplication naïve, le produit requiert en $O(di^2)$ opérations arithmétiques. Ce calcul s'effectue à k reprises, la complexité est donc en $O(dk^3)$ opérations. \square

On peut montrer que dans certains cas, on peut avoir la minimalité de l'ordre de la récurrence.

Theorème 2.3. *Si $q_k(1)q_k(-1) \neq 0$ alors l'opérateur de récurrence calculé par l'algorithme précédent est l'opérateur d'ordre minimal.*

La démonstration de ce théorème se trouve en section 2.4.

2.3 L'algorithme de Lewanowicz

Lewanowicz [8] présente son algorithme qui permet de calculer la récurrence minimale dans tous les cas. Il calcule aussi le numérateur de la fraction de récurrence. La différence avec la sortie de l'algorithme 4 est que la récurrence calculée est le numérateur de la fraction irréductible. Il prouve que cette fraction est irréductible sans utiliser les propriétés des corps de Ore.

Dans ce paragraphe je donne deux algorithmes, celui présenté dans [8] et un algorithme équivalent où la preuve de la minimalité est plus simple à voir. Je démontrerai la minimalité pour le second algorithme.

L'algorithme donné par Lewanowicz dans un vocabulaire différent est le suivant :

Algorithme 5 Lewanowicz Tchebychev Récurrence

ENTRÉES: $\text{seq}(p_i, i = 0..k)$

SORTIES: L tel que $L(c_n) = 0$

$L := p_k(X)$

$P = 1$

pour tout i de $k - 1$ à 0 **faire**

$[U, V] := \text{cofacteur}(\text{ppcm}(B_1, L))$ ($UB_1 = VL$)

$P := VP$

$L := Un + Pp_i$

fin pour

Un algorithme équivalent utilisant les corps de Ore est :

Algorithme 6 Lewanowicz Tchebychev Récurrence version ORE

ENTRÉES: $\text{seq}(p_i, i = 0..k)$

SORTIES: L tel que $L(c_n) = 0$

$L = p_k(X)$

pour tout i de $k - 1$ à 0 **faire**

$L := LD + p_i(X)$

fin pour

retourne (numérateur de L)

Ici D^{-1} est représenté comme l'inverse d'un polynôme de Laurent, c'est-à-dire que j'ai pris sa représentation irréductible.

Lemme 2.4. *Les algorithmes 5 et 6 effectuent les mêmes opérations sur le corps $\mathbb{K}(n)$.*

Preuve À l'étape i , l'algorithme 5 fait une addition comme décrite dans 1.3 page 13. Cette addition effectue les mêmes opérations que l'addition de fractions effectuée dans l'algorithme 6. □

J'ai énoncé ce second algorithme pour simplifier la preuve du théorème suivant.

Theorème 2.5. *Les algorithmes 5 et 6 calculent la récurrence de Tchebychev d'ordre minimal.*

Preuve La correction est facile à voir avec l'algorithme 6, l'algorithme calcule bien la récurrence vérifiée par les coefficients de la série de Tchebychev.

On va montrer par récurrence sur i que L_i est irréductible à l'étape i de l'algorithme.

À l'ordre k , il est clair que $L_k = p_k(X)$ est irréductible.

Supposons qu'à l'itération $i+1$, L_{i+1} soit irréductible. D est l'inverse d'un polynôme, $L_{i+1}D$ est par le lemme 1.8 irréductible. Soient B et A dans $\mathbb{K}(n)\langle S, S^{-1} \rangle$ tel que $L_{i+1}D = B^{-1}A$, on a $L_i = B^{-1}(A + Bp_{k-i}(X))$. La fraction rationnelle L_i est bien irréductible. \square

Theorème 2.6. *L'algorithme 5 effectue $O(k^3d)$ opérations où d est le maximum des degrés des p_i .*

Preuve Le polynôme $B_1 = (S^{-1} - S)$ est de degré constant et ne dépend donc pas de k , le calcul du ppcm dépend donc de la taille de L_i et s'effectue en temps linéaire par rapport à cette taille. Le ppcm se calcule en $O(k^2)$ opérations.

Le calcul le plus coûteux sera encore ici la multiplication de B par $p_i(X)$ et pour les mêmes raisons que l'algorithme 4, on en déduit la complexité de $O(k^3d)$ opérations. \square

2.4 Equivalence entre les algorithmes de Paszkowski et Lewanowicz

Le résultat principal de cette partie est le théorème suivant. Ma contribution est de donner une preuve plus simple que celle de Lewanowicz [8] en utilisant la théorie des polynômes de Ore.

Theorème 2.7 (Lewanowicz [8]). *Si $p_k(1)p_k(-1) \neq 0$, alors la récurrence calculée par l'algorithme de Lewanowicz est du même ordre que celle calculée par l'algorithme de Paszkowski.*

Pour démontrer ce théorème nous avons d'abord besoin de prouver plusieurs propriétés. Je rappelle que

$$X := \frac{S + S^{-1}}{2} \text{ et } D := \frac{2n}{S^{-1} - S}.$$

$\mathbb{K}[X]$ est donc considéré ici comme un sous anneau de l'anneau commutatif $\mathbb{K}\langle S, S^{-1} \rangle$.

Lemme 2.8. *Soit p un polynôme dans $\mathbb{K}[X]$. Si $p(1)p(-1) \neq 0$, alors $\text{pgcd}(p(X), S^{-1} - S) = 1$.*

Preuve Je vais démontrer ce résultat par contraposé.

On suppose que $\text{pgcd}(p(X), S^{-1} - S) \neq 1$. On remarque que $S^{-1} - S = (1 - S)(1 + S)S^{-2}$. Les polynômes $1 + S$ et $1 - S$ sont premier entre eux, deux cas sont donc possibles, soit $(1 + S)$ divise p , soit $(1 - S)$ divise p .

Étudier le premier cas suffit, le second est similaire. On écrit p dans la base de l'espace vectoriel $\mathbb{K}[X]$, $1, (1 + X), (1 + X)^2, \dots, (1 + X)^n$:

$$p(X) = a_0 + a_1(1 + X) + \dots + a_n(1 + X)^n.$$

Remplacer X par sa valeur mène à :

$$p(S, S^{-1}) = a_0 + a_1(1 + S)^2 S^{-1} + a_2(1 + S)^4 S^{-2} + \dots + a_n(1 + S)^{2n} S^{-n}.$$

Comme p est divisible par $1 + S$, $a_0 = 0$, ce qui entraîne

$$p(X) = a_1(1 + X) + \dots + a_n(1 + X)^n, \tag{24}$$

d'où découle $p(-1) = 0$.

\square

Lemme 2.9. *Pour tout polynôme $p(X) \in \mathbb{K}[X]$, il existe un polynôme $q(X)$ tel que $p(X)n = np(X) + q(X)(S^{-1} - S)$.*

Preuve La première remarque que l'on peut faire est que l'on a $Xn = (n-1)S^{-1} + (n+1)S = nX + (S - S^{-1})$.

Par récurrence sur i , on montre que $X^i n = nX^i - iX^{i-1}(S^{-1} - S)$. Pour $i = 1$, l'hypothèse de récurrence est vérifiée par la remarque précédente. Supposons l'hypothèse de récurrence vraie pour i . On a alors

$$\begin{aligned} X^{i+1}n &= X(nX^i - iX^{i-1}(S^{-1} - S)), \\ &= nX^{i+1} + (S^{-1} - S)X^i - iX^i(S^{-1} - S), \\ &= nX^{i+1} - (i+1)X^i(S^{-1} - S). \end{aligned}$$

Par linéarité, on en déduit le lemme. \square

Lemme 2.10. *Pour tout polynôme p dans $\mathbb{K}[X]$, on a pour tout k , $pD^k = D^k(p + D^{-1}Q)$, où Q est un polynôme dans $\mathbb{K}[X]$.*

Si de plus p est premier avec $S^{-1} - S$, alors cette représentation est irréductible.

Preuve Soit $p \in \mathbb{K}[X]$. Soit $p_0 = p$, on définit p_i tel que $p_{i-1}(X)n = np_{i-1}(X) + p_i(X)(S^{-1} - S)$.

On va montrer par récurrence sur k que

$$pD^k = D^k \left(p + D^{-1} \left(\sum_{i=1}^k (k+1-i)D^{-i+1}p_i \right) \right), \quad (25)$$

et que si p est premier avec $(S^{-1} - S)$ alors cette fraction est irréductible.

Pour $k = 0$, l'hypothèse est vérifiée. Supposons l'hypothèse vérifiée pour un $k \in \mathbb{Z}$, on a alors

$$pD^{k+1} = D^k \left(p + D^{-1} \left(\sum_{i=1}^k (k+1-i)D^{-i+1}p_i \right) \right) D. \quad (26)$$

Les polynômes p et p_i commutent avec $S^{-1} - S$, par l'égalité $D := (-S + S^{-1})2n$, on a :

$$pD^{k+1} = D^k \left((S^{-1} - S)^{-1}pn + D^{-1} \left(\sum_{i=1}^k (k+1-i)D^{-i+1}(S^{-1} - S)^{-1}p_i n \right) \right). \quad (27)$$

En appliquant le lemme 2.9, on obtient :

$$pD^{k+1} = D^{k+1} \left(p + D^{-1} \left(\sum_{i=1}^{k+1} (k+2-i)D^{-i+1}p_i \right) \right) \quad (28)$$

Si on suppose de plus que p est premier avec $S^{-1} - S$, par hypothèse de récurrence, $D^k p$ est une fraction irréductible.

Par le lemme 1.8, comme p est premier avec $S^{-1} - S$, le passage de l'équation (26) à (27), rend pD^{k+1} irréductible à l'équation (27). Dans la suite des calculs le degré du dénominateur ne change pas, donc la fraction reste irréductible. \square

Preuve du théorème 2.7 Si $p_k(1)p(-1) \neq 0$, alors p_k est premier avec $S^{-1} - S$. D'après le lemme 2.10, $D^k p_k$ est irréductible, mais on a aussi pour tout i plus grand que k , $p_i D^i = D^i(p_i + D^{-1}Q_i)$.

On a donc

$$\sum_{i=0}^k p_i(x) D^i = D^k (p_k + D^{-1}Q), \quad (29)$$

où Q est un polynôme, et cette fraction est irréductible. Le numérateur de cette fraction est une récurrence du même ordre que celle calculée par l'algorithme de Paszkowski (B_k à le même degré en S que D^k).

On vient donc de prouver que la récurrence calculée par l'algorithme de Paszkowski est d'ordre minimale. Ceci prouve le théorème 2.7. □

2.5 L'algorithme de Rebillard

Rebillard [16] donne une idée d'algorithme pour le calcul des récurrences de Tchebychev. La récurrence qu'il propose de calculer est très semblable à celle de Paszkowski, c'est la suivante :

$$\sum_{i=0}^k D^{k-i} q_i(X), \quad (30)$$

les $q_i(X)$ sont les mêmes polynômes que ceux définis à la ligne (23).

Contrairement à Paszkowski, il ne calcule pas les q_i dans le monde différentiel, mais il transforme l'opérateur de récurrence $p_i(X)$ en l'opérateur $q_i(x)$ après avoir fait la remarque , que pour tout k , on a :

$$D^{-k} X D^k = \frac{(n+k)S + (n-k)S^{-1}}{2n}. \quad (31)$$

De cette remarque, il définit l'opérateur :

$$X_k = \frac{(n+k)S + (n-k)S^{-1}}{2n} \quad (32)$$

La récurrence qu'il en déduit est la suivante :

$$\sum_{i=0}^k X_k D^{k-i} \quad (33)$$

Par définition de X_k , on retrouve bien la récurrence de l'équation (30).

De ces égalités je déduis un algorithme que j'appellerai l'algorithme de Rebillard.

Algorithme 7 Rebillard Tchebychev Récurrence version ORE

ENTRÉES: seq($p_i, i = 0 \dots k$)

SORTIES: L tel que $L(c_n) = 0$

$L = p_0(X_k)$

pour tout i de 1 à k **faire**

$L := LD^{-1} + p_i(X_k)$

fin pour

La correction de cet algorithme se vérifie immédiatement.

En nombre d'opérations, travailler avec l'opérateur X_k n'est pas idéal. En effet jusqu'à présent le calcul des $q_i(X)$, s'effectuer rapidement, car les $q_i(X)$ étaient des polynômes dans l'anneau commutatif $\mathbb{K}[n]\langle S, S^{-1} \rangle$. Ici c'est le calcul des $p_i(x)$ qui va dominer.

L'algorithme naïf qui est de calculer les $p_i(X_k)$ à chaque étape donne $O(d^3)$ opérations, où d est le degré de p_i . On pourrait aussi calculer les p_i avec à l'aide d'un algorithme du type diviser pour régner et d'une multiplication rapide d'opérateurs, ce qui ramènerait la complexité à $O(d^\omega)$ opérations. On doit effectuer cette opération k fois, la complexité non naïve serait de $O(d^{\omega+1})$ opérations.

2.6 Nouveaux algorithmes

Dans cette section, je présente deux nouveaux algorithmes plus rapides que ceux proposés ci dessus. Avant de donner ces algorithmes, je donne une amélioration aux algorithmes de Lewanowicz et Paszkowski. Je finis cette section avec un algorithme permettant de calculer la récurrence d'ordre minimale de l'algorithme de Lewanowicz dans tous les cas en gardant les mêmes complexités.

Amélioration des algorithmes de Paszkowski et Lewanowicz

Algorithme 8 Multiplication rapide

ENTRÉES: $P \in \mathbb{K}[n]\langle S, S^{-1} \rangle$ et $Q \in \mathbb{K}\langle S, S^{-1} \rangle$

SORTIES: PQ

Calcul des polynômes $a_i(S, S^{-1})$ tel que $P := \sum_i n^i a_i(S, S^{-1})$

pour tout i faire

$a_i(S, S^{-1}) := a_i(S, S^{-1})Q$

fin pour

$PQ := \sum_i n^i a_i(S, S^{-1})$

Lemme 2.11. *Si P est un polynôme de bidegré (k, k) et Q est un polynôme de degré d , la multiplication de P et Q se calcule en $O(kM(\sup(k, d)))$ opérations.*

La preuve de ce lemme se déduit immédiatement de l'algorithme précédent.

Corollaire 2.12. *L'algorithme 4 s'effectue en $O(k^2M(\sup(k, d)))$ opérations sur le corps k .*

La preuve de ce corollaire est une conséquence du lemme précédent et du lemme 2.2.

Hörner

Voici un nouvel algorithme pour calculer la récurrence.

Algorithme 9 Paszkowski Horner

ENTRÉES: seq($q_i, i = 0 \dots k$)
SORTIES: L tel que $L(c_n) = 0$
 $L := q_0(X)$
pour tout i de 1 à k **faire**
 $L := D^{-1}L + q_i(X)$
fin pour

Theorème 2.13. *L'algorithme 9 renvoie bien la récurrence de Tchebychev. Il effectue $O((d+k)k^2)$ opérations.*

Preuve La correction de cet algorithme est facile à voir car il retourne l'opérateur

$$\sum_{i=0}^k D^{k-i} q_i(X).$$

Pour le calcul de la complexité, on remarque que la multiplication de D^{-1} par L est linéaire en la taille de L , en effet D^{-1} est un polynôme en S de degré 2.

À l'étape i , le degré de L et de $2(i+d)$ en S . Les coefficients de L sont des fractions rationnelles en n dont le numérateur et le dénominateur ont un degré $2i$. La multiplication coûtera donc $O((d+i)i)$ opérations arithmétiques. Les autres opérations effectuées à chaque étape ne seront pas plus que linéaires en la taille de L (addition de polynômes et mise au même dénominateur en n).

On effectue k étapes, la complexité de cet algorithme s'en déduit. □

Diviser pour régner

Voici maintenant un nouvel algorithme du type diviser pour régner. Cet algorithme me permet d'utiliser la multiplication rapide d'opérateurs.

Pour simplifier, je vais supposer ici que l'équation différentielle est d'ordre k , une puissance de 2. L'opérateur $*$ est l'opérateur de multiplication de deux polynômes de Ore par l'algorithme de van der Hoeven présenté page 12.

Algorithme 10 Paszkowski Diviser pour régner

ENTRÉES: seq($q'_i, i = 0 \dots k$)
SORTIES: L, P
si $M=1$ **alors**
 Retourne ($q_i(X), D^{-1}$)
finsi
 $L_1, P :=$ Algorithme 10(seq($q'_i, i = 0..M/2 - 1$))
 $L_2 :=$ Algorithme 10(seq($q'_i, i = M/2..M$))
 $L := L_1 + P * L_2$
 $P := P * P$

Theorème 2.14. *Cette algorithme renvoie bien la récurrence vérifiée par les coefficients de la série de Tchebychev. Il effectue $O((k+d)k^{\omega-1})$ opérations.*

Preuve L'algorithme 10 renvoie le même opérateur que l'algorithme 9, il calcule donc bien la bonne récurrence.

Soit $C(k, d)$, le coût de l'algorithme pour une équation différentielle d'ordre k à coefficients polynomiaux de degrés d en x . L est un opérateur dans $\mathbb{K}(n)\langle S, S^{-1} \rangle$, d'ordre $2k + 2d$, dont les coefficients sont des fractions rationnelles en n avec des degrés de numérateur et de dénominateur majorés par $2k$. Pour l'opérateur P , on a l'égalité $P = D^{-k/2}$, dont on connaît la forme par l'égalité (22) page 17, on a donc un opérateur d'ordre k en S et dont les coefficients sont des fractions rationnelles en n avec des degrés de numérateur et de dénominateur majorés par $2k$.

De la taille de ces objets, on peut déduire la complexité des opérations calculées aux étapes (6) et (7).

L'algorithme 3 page 12 prend en entrée des opérateurs à coefficients des polynômes en n , mais il est adaptable lorsque les coefficients sont des fractions rationnelles en n , car on sait évaluer et interpoler des fractions rationnelles. Le nombre de points à évaluer sera linéaire en le maximum des degrés entre le numérateur et le dénominateur. Le calcul dominant est donc toujours la multiplication de matrices, et la complexité reste la même.

La mise au carré de P se calcule donc en k^ω opérations. Pour la multiplication de P par L_2 , il ne faut pas appliquer l'algorithme 3 tout de suite mais d'abord découper l'opérateur L_2 comme $(d + k)/k$ opérateurs d'ordre k , c'est-à-dire :

$$L_2 := L_{(2,0)} + L_{(2,1)}S^{d/k} + \dots + L_{(2,(k+d)/d)}S^d.$$

On multiplie ensuite chaque $L_{(2,i)}$ par P , ce qui se calcule en $O((d + k)k^{\omega-1})$ opérations. On a donc :

$$C(k, d) = 2C(k/2, d) + (k + d)k^{\omega-1}$$

On conclut par le lemme diviser pour régner [2, lemme 1 page 10] qui nous donne la complexité souhaitée. □

La minimalité de l'algorithme de Lewanowicz sans changer les complexités

La minimalité de l'algorithme de Lewanowicz repose sur le lemme 1.8 page 14 qui ne s'applique qu'en effectuant les mêmes opérations que l'algorithme 5. En utilisant des algorithmes rapides, on perd cette minimalité.

Dans cette sous-section, je propose un algorithme qui transforme la famille des polynômes $(q_i, i = 0 \dots k)$ en une famille de polynômes $(q'_i, i = 0 \dots k')$ (q'_k peut être une fraction de Ore), telle que $k \geq k'$ et :

$$\sum_{i=0}^k D^i q_i(X) = \sum_{i=0}^{k'} D^i q'_i(X), \tag{34}$$

et que $\sum_{i=0}^{k'} D^i q'_i(X)$ soit une fraction irréductible.

Algorithme 11 Paszkowski Minimal

ENTRÉES: $\text{seq}(q_i), i = 0 \dots k$ **SORTIES:** $\text{seq}(q_i), i = 0 \dots k'$ $G := \text{pgcd}(D^{-1}, q_k)$ **tantque** $G \neq 1$ **faire** $q_{k-1}(X) := Dq_k(X) + q_{k-1}$ $k := k - 1$ **si** $D^{-1} \neq G$ **alors**Retourne($\text{seq}(q_i), i = 0..k$)**sinon** $G := \text{pgcd}(D^{-1}, q_k)$ **finsi****fin tantque**Retourne ($\text{seq}(q_i), i = 0..k$)

La sortie de cet algorithme vérifie bien l'équation (34).

Par le lemme 2.10, la fraction rationnelle $\sum_{i=0}^{k'} D^i q'_i(X)$ est irréductible.

Il suffit alors de calculer l'opérateur $\sum_{i=0}^{k'} D^{-(k-i)} q'_i(X)$ avec les algorithmes rapides présentés ci-dessus.

Conclusion

Ce stage a permis de développer la structure des corps des fractions rationnelles non commutatives, qui paraît être idéale pour le calcul de récurrences des coefficients d'une fonction D-finie dans une base de fonctions spéciales. Le calcul des récurrences de Tchebychev est maintenant compris. Lors de ce travail, les algorithmes existants ont été revisités et leurs corrections ont été prouvées. L'analyse de complexité a aussi été effectuée, ce qui a permis de développer de nouveaux algorithmes plus rapides en utilisant des résultats récents de complexité sur les opérations entre opérateurs de récurrences.

La famille des polynômes de Tchebychev sont un cas particulier de la famille des polynômes de Jacobi. Lewanowicz a généralisé sa méthode à l'ensemble des polynômes de Jacobi [9, 10]. De la même façon, les nouveaux algorithmes présentés dans ce rapport se généralisent à la famille des polynômes de Jacobi. L'utilisation des corps de fractions rationnelles non commutatifs permet de généraliser les algorithmes à de nombreuses autres familles de fonctions, comme les polynômes de Laguerre, Hermite ou les fonctions de Bessel.

Une prochaine étape serait donc de travailler sur d'autres familles de fonctions, de déterminer les fonctions ayant les propriétés nécessaires pour des développements en séries, et de trouver des algorithmes rapides pour l'ensembles de ces fonctions. Il faudra pour cela effectuer un travail approfondi sur les opérations rapides dans les corps des fractions rationnelles de récurrences.

Remerciements

Je tiens à remercier chaleureusement mon directeur de stage Bruno Salvy pour son aide, et avec lui les autres membres de l'équipe de calcul formel du laboratoire commun : Alin Bostan, Frédéric Chyzak et Marc Mezzarobba.

Je remercie aussi vivement Stéphane Le Roux pour sa relecture ainsi que l'ensemble des membres du projet algorithms et du centre de recherche commun INRIA-Microsoft Research pour leur chaleureux accueil.

Références

- [1] *Handbook of mathematical functions, with formulas, graphs, and mathematical tables.* Edited by Milton Abramowitz and Irene A. Stegun. Dover Publications Inc., New York, 1966.
- [2] A. Bostan, F. Chyzak, M. Giusti, B. Salvy, and E. Schost. *Algorithme en calcul formel et automatique.* <http://algo.inria.fr/salvy/mpri/Poly.pdf>.
- [3] A. Bostan, F. Chyzak, and N. Le Roux. Skew-polynomial products by evaluation and interpolation. In preparation.
- [4] C. W. Clenshaw. The numerical solution of linear differential equations in Chebyshev series. *Proc. Cambridge Philos. Soc.*, 53 :134–149, 1957.
- [5] A. Gil, J. Segura, and N. M. Temme. *Numerical methods for special functions.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.
- [6] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 135–142 (electronic), New York, 2003. ACM.
- [7] D. Y. Grigor'ev. Complexity of factoring and calculating the GCD of linear ordinary differential operators. *J. Symbolic Comput.*, 10(1) :7–37, 1990.
- [8] S. Lewanowicz. Construction of a recurrence relation of the lowest order for coefficients of the Gegenbauer series. *Zastos. Mat.*, 15(3) :345–396, 1976.
- [9] S. Lewanowicz. Construction of the lowest-order recurrence relation for the Jacobi coefficients. *Zastos. Mat.*, 17(4) :655–675, 1983.
- [10] S. Lewanowicz. Quick construction of recurrence relations for the Jacobi coefficients. *J. Comput. Appl. Math.*, 43(3) :355–372, 1992.
- [11] Z. Li and I. Nemes. A modular algorithm for computing greatest common right divisors of Ore polynomials. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (Kihei, HI)*, pages 282–289 (electronic), New York, 1997. ACM.
- [12] O. Ore. Linear equations in non-commutative fields. *Ann. of Math. (2)*, 32(3) :463–477, 1931.
- [13] O. Ore. Theory of non-commutative polynomials. *Ann. of Math. (2)*, 34(3) :480–508, 1933.
- [14] S. Paszkowski. *Zastosowania numeryczne wielomianów i szeregów Czebyszewa.* Państwowe Wydawnictwo Naukowe, Warsaw, 1975. Podstawowe Algorytmy Numeryczne. [Fundamental Numerical Algorithms].
- [15] M. Petkovšek, H. S. Wilf, and D. Zeilberger. *A = B.* A K Peters Ltd., Wellesley, MA, 1996. With a foreword by Donald E. Knuth, With a separately available computer disk.
- [16] L. Rebillard. *Etude théorique et algorithmique des séries de Chebychev, solutions d'équations différentielles holonomes.* PhD thesis, Institut national polytechnique de Grenoble, 1998.

- [17] B. Salvy and P. Zimmermann. Gfun : a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, 20(2) :163–177, 1994.
- [18] G. Szegő. *Orthogonal polynomials*. American Mathematical Society, Providence, R.I., fourth edition, 1975. American Mathematical Society, Colloquium Publications, Vol. XXIII.
- [19] J. van der Hoeven. FFT-like multiplication of linear differential operators. *J. Symbolic Comput.*, 33(1) :123–127, 2002.