Algorithmique semi-numérique rapide des séries de Tchebychev

Alexandre Benoit

École doctorale de l'École polytechnique

THÈSE

pour obtenir le grade de

Docteur de l'École polytechnique

Spécialité : **Informatique**

préparée à **INRIA**

présentée par

Alexandre Benoit

le 18 Juillet 2012

Algorithmique semi-numérique rapide des séries de Tchebychev

Jury

Joris van der Hoeven	CNRS & École polytechnique,	président du jury
Bruno Salvy	INRIA,	directeur de thèse
Moulay Barkatou	Université de Limoges,	rapporteur
Marko Petkovšek	Université de Ljubljana, Slovénie,	rapporteur
Nicolas Brisebarre	CNRS & ENS de Lyon,	examinateur
Mark van Hoeij	Université de l'état de Floride, USA,	examinateur
Alban Quadrat	INRIA & Supelec,	examinateur

Résumé/Abstract

Résumé

Une série de Tchebychev est un développement dans la base des polynômes de Tchebychev. Ces séries sont importantes en théorie de l'approximation. Contrairement aux séries de Taylor, l'algorithmique en calcul formel autour d'elles n'est pas très développée.

Cette thèse propose de nouveaux algorithmes pour ces séries. Une première partie présente des algorithmes rapides pour convertir une série de Tchebychev tronquée en une série de Taylor tronquée et réciproquement, et pour multiplier ou diviser deux séries de Tchebychev tronquées. Le reste de la thèse porte sur les séries de Tchebychev solutions d'une équation différentielle linéaire à coefficients polynomiaux. Dans cette classe, les coefficients des séries sont solutions d'une récurrence linéaire. Cette thèse montre comment calculer cette récurrence efficacement, puis comment l'utiliser pour obtenir un calcul approché efficace des coefficients malgré des instabilités numériques. Ces algorithmes mènent au calcul efficace d'une approximation sur un segment par un polynôme de degré fixé d'une fonction solution d'une équation différentielle linéaire. Enfin, le calcul des récurrences pour les coefficients de séries est généralisé au cas des séries de Fourier généralisées. L'ensemble est illustré d'exemples à partir de programmes développés durant cette thèse.

Abstract

A Chebyshev series is an expansion in the basis of Chebyshev polynomials of the first kind. These series are important in approximation theory. Unlike Taylor series, their algorithmic aspects are not very developed in computer algebra.

This thesis proposes new algorithms for these series. A first part gives fast algorithms that convert a truncated Chebyshev series into a truncated Taylor series and vice versa, and others that multiply or divide two truncated Chebyshev series. The rest of the thesis is devoted to Chebyshev series that are solutions of a linear differential equation with polynomial coefficients. In this class, the coefficients of series are solutions of a linear recurrence. This thesis shows how to compute this recurrence efficiently, then how to use it for the efficient computation of approximated coefficients despite numerical instabilities. These algorithms lead to an efficient computation of the approximation by polynomials of fixed degree on a segment for solutions of linear differential equations. Finally, the computation of recurrences for the coefficients of series is generalized to the case of generalized Fourier series. The document is illustrated by examples using implementations developed during this thesis.

Remerciements

Mes premiers remerciements vont à Bruno Salvy qui m'a guidé tout au long de ces quatre années de thèse malgré un emploi du temps très chargé. Il m'a fait découvrir le calcul formel lors de mon master et m'a tout de suite donné envie de travailler avec lui. Après ces quatre années, je ne regrette pas mon choix de l'avoir suivi. Je dois dire, si ce manuscrit a pu voir le jour c'est grâce aux temps qu'il a consacré à le relire afin de m'aider à l'améliorer. Je sais que ce ne fût pas toujours facile pour lui, notamment lors de la dernière ligne droite où il m'a impressionné par sa réactivité et sa puissance de travail.

Lors de ce même cours, j'ai fait la connaissance de Alin Bostan et Frédéric Chyzak que j'ai eu la chance de côtoyer tout au long de ma thèse. Je les remercie pour le temps qu'ils m'ont consacré pour répondre à mes questions et l'aide qu'ils m'ont apportée.

J'ai eu la chance d'effectuer cette thèse dans le même bureau que Marc Mezzarobba. Je garde des bons souvenirs du temps que l'on a passé ensemble pour parler de science (ou d'autres choses dès que notre directeur de thèse avait le dos tourné). Je le remercie aussi vivement pour toute l'aide qu'il m'a apportée grâce à sa connaissance profonde de l'informatique que ce soit au niveau pratique (même s'il râlait en présence d'une pomme croquée sur l'ordinateur) ou théorique.

Je tiens à remercier chaleureusement Moulay Barkatou et Marko Petkovšek qui m'ont fait l'honneur de rapporter cette thèse. Je remercie aussi Nicolas Brisebarre, Mark van Hoeij, Joris van der Hoeven et Alban Quadrat d'avoir accepté de participer à mon jury.

Lors de cette thèse, j'ai eu la chance d'évoluer dans trois équipes différentes ce qui m'a permis de rencontrer et de côtoyer de nombreux chercheurs en informatique. J'ai passé la plus grande partie de mon temps à Orsay dans le laboratoire commun MSR-INRIA dirigé par Jean-Jacques Lévy que je remercie pour avoir su créer des conditions parfaites de travail en s'efforçant de favoriser le bien-être des chercheurs.

Je remercie l'ensemble des personnes qui sont passées par l'équipe du DDMF, Alexis, Christoph, Stefan, Flavia, Élie, Pierre, Lucien et plus généralement celles que j'ai pu côtoyer au laboratoire commun MSR-INRIA, notamment, James, Jérémy, François, Cyril, Assia et Denis pour les discussions plus ou moins sérieuses que j'ai pu avoir avec eux.

De la même façon je remercie les personnes que j'ai côtoyées à Rocquencourt dans l'équipe Algo, Philippe Dumas, Shaoshi Chen, Nicolas Broutin, Pierre Nicodème, Cyril Banderier, Basile Morcrette, Jérémie Lumbroso, Carine Pivoteau et Nicolas Le Roux.

J'ai pu effectuer ma dernière année dans l'équipe POLSYS. Je tiens spécialement à

remercier Guénaël Renault qui m'a fait l'honneur de « squatter » **mon** bureau pendant cette année. Mohab Safey El Din m'a guidé lors de mon arrivée à Paris VI, il m'a notamment aidé à choisir les enseignements afin d'optimiser mon temps de recherche et finir ma thèse, je le remercie pour ça. Merci aussi à Jean-Charles Faugère pour m'avoir encouragé et soutenu lors de ma venue. Je remercie l'ensemble des autres membres de l'équipe, Daniel, Ludovic, Martin, Elias, Luk, Christopher, Aurélien, Louise, Pierre-Jean, Jules, Frédéric, Rina et plus généralement ceux du département CalSci, notamment Mourad, Christoph, Valérie, LSD et Stef pour nos échanges scientifiques, pédagogiques et surtout pour les échanges un peu moins sérieux.

Je remercie aussi les autres thésards, et maintenant pour la plupart docteur, avec qui j'ai eu des contacts privilégiés notamment Mioara Joldeş avec qui je suis sûr de finir par être co-auteur (notre article sera un jours envoyé, c'est sûr) mais aussi Mathieu Feuillet, Romain Cosset, Luca De Feo, Stéphane Jacob,Remez Romain Lebreton et Jérémy Berthomieu.

Un grand merci à Virginie Collette et Martine Thirion pour la gestion efficace de toute la partie administrative et des missions ainsi que pour les discussions qu'on a eues ensemble.

Cette thèse n'aurait pas vu le jour si mes parents ne m'avaient pas soutenu (financièrement et moralement) pendant toutes mes études. Merci pour cette aide indispensable. Merci à ma grand-mère de m'avoir hébergé durant cette thèse.

Surtout un grand merci à Marie-Laure qui m'a soutenu tout du long et surtout a géré, malheureusement trop souvent seule, la première année de Coline afin que je finisse ma rédaction. Malgré le temps que Coline m'a pris ces derniers mois, je tiens à la remercier pour tous les bons moments et le bonheur qu'elle m'apporte.

Je sais que les années passées au sein de l'équipe Algo et dans le domaine du calcul formel n'auraient pas été les mêmes sans l'apport scientifique et humain de Philippe Flajolet. J'ai eu la chance de le côtoyer pendant trois ans en partie dans « sa salle café Algo ». Je tiens à saluer sa mémoire et le remercier pour ce qu'il m'a apporté.

Table des matières

Chapitre 1

Introduction

Sommaire

1 Motivations et Résultats

L'objet de cette thèse est l'étude et la conception d'algorithmes rapides pour le développement de fonctions en séries de Tchebychev. Ces séries, de la forme $\sum_{n\in\mathbb{N}} c_n T_n(x)$ peuvent être vues, par la définition $T_n(\cos(\theta)) = \cos(n\theta)$, comme des séries de Fourier avec un changement de variable.

À l'instar des séries de Fourier, les séries de Tchebychev ont un intérêt pour la théorie de l'approximation. Il est en effet connu que le développement tronqué d'une fonction en série de Tchebychev est un polynôme proche du meilleur approximant polynomial, au sens de la norme uniforme, sur le segment $[-1, 1]$.

Malgré ces propriétés, ce développement n'est pas souvent utilisé pour approximer une fonction sur un segment. La raison principale est qu'il n'est pas facile de calculer les coefficients de Tchebychev. Quitte à calculer un développement, il est plus aisé de l'effectuer en série de Taylor ; même si le degré de troncature est plus élevé, les coefficients sont plus faciles à calculer. D'autres approximent avec des polynômes d'interpolation. Ils utilisent alors l'algorithme de Remez qui donne la meilleure approximation polynomiale pour un degré fixé ou interpolent en des points de Tchebychev afin d'obtenir une « bonne » approximation avec des propriétés semblables aux séries de Tchebychev. Ces méthodes demandent une évaluation multipoints de la fonction, ce qui n'est pas toujours facile.

Le propos de cette thèse est de réhabiliter les séries de Tchebychev en proposant des algorithmes efficaces afin de calculer ces coefficients et en montrant qu'utiliser des séries de Tchebychev tronquées est un des moyens les plus simples d'approximer une série. Dans cette thèse, ces algorithmes sont en grande partie étudiés dans le cas très fréquent où les séries de Tchebychev sont solutions d'équations différentielles linéaires à coefficients polynomiaux. En effet, dans ce cas, ces séries vérifient une belle propriété, semblable à ce qui se passe pour les séries de Taylor : les coefficients de la série vérifient une récurrence linéaire à coefficients polynomiaux. C'est en grande partie en exploitant cette propriété que je donnerai des algorithmes efficaces.

Concrètement, les exemples suivants exposent des problèmes pouvant être traités par les résultats de ma thèse.

Problème 1.1. Étant donnée une série de Tchebychev tronquée

$$c_0 T_0(x) + c_1 T_1(x) + \cdots + c_n T_n(x),$$

peut-on l'exprimer rapidement comme un polynôme en x ?

Dans le chapitre 3, j'expose deux algorithmes adaptés pour l'un de Pan [Pan98] et pour l'autre de Bostan, Salvy et Schost [BSS08] permettant d'effectuer ce calcul très rapidement. Je compare ces deux algorithmes et je montre que ce calcul s'effectue asymptotiquement aussi rapidement que multiplier deux polynômes de degré n.

Un algorithme inverse permettant d'exprimer un polynôme en x comme un polynôme dans la base des T_n est aussi exhibé avec la même complexité.

Problème 1.2. Étant donnés deux polynômes exprimés dans la base de Tchebychev, peut-on multiplier ou diviser rapidement ces polynômes sans effectuer de changement de base ?

Toujours dans le chapitre 3, j'expose un algorithme connu permettant d'effectuer la multiplication en un temps équivalent à deux multiplications de polynômes dans la base monomiale. Ensuite, je donne un nouvel algorithme de division rapide de polynômes exprimés dans la base de Tchebychev. Je montre que cette division peut être effectuée au même coût que lorsque elle est calculée dans la base monomiale.

Problème 1.3. Un résultat de Luke [Luk69] nous donne des jolies formules de développements en séries de Tchebychev. Par exemple

$$\sin(x) = 2\sum_{n=0}^{\infty}(-1)^n \mathrm{J}_{2n+1}(1)T_{2n+1}(x),$$

où J est la fonction de Bessel de première espèce. Comment prouver ou trouver de telles formules ?

Dans le chapitre 6, je montre comment obtenir automatiquement ce type de formules.

Problème 1.4. Étant donnée une série de Tchebychev

$$\sum_{n\in\mathbb{N}} c_n T_n(x),$$

solution d'une équation différentielle linéaire à coefficients polynomiaux, que peut-on dire des c_n ?

Dans le chapitre 5, je rappelle un résultat classique montrant que les c_n sont solutions d'une récurrence linéaire à coefficients polynomiaux. Des travaux antérieurs donnent des algorithmes pour le calcul de cette récurrence, je montre comment unifier ces algorithmes simplement et je propose un nouvel algorithme plus rapide pour la calculer.

Problème 1.5. Étant donnée une série de Tchebychev

$$\sum_{n\in\mathbb{N}} c_n T_n(x),$$

solution d'une équation différentielle linéaire à coefficients polynomiaux et analytique en 0. Comment calculer rapidement des approximations des N premiers c_n à une précision fixée ϵ ?

L'utilisation directe de la récurrence n'est pas une bonne solution pour calculer ces coefficients. Une première explication vient de la difficulté d'obtenir les conditions initiales qui sont souvent des nombres transcendants (exemple $\mathrm{J}_1(1)$ pour sin). Une seconde raison vient de la présence de solutions divergentes dans toutes récurrences annulant une suite de coefficients de Tchebychev, comme je le montre dans le chapitre 5.

Dans le chapitre 6, je donne un nouvel algorithme permettant d'effectuer ce calcul en un temps linéaire par rapport à N et $\log(\epsilon^{-1})$ en utilisant *intelligemment* cette récurrence.

Problème 1.6. Étant donnée une fonction solution d'une équation différentielle, comment calculer rapidement un « bon » approximant polynomial (au sens de la norme uniforme), de degré d, sur le segment $[-1,1]$? Comment évaluer la qualité de cet approximant ?

Dans le chapitre 7, je donne un algorithme prenant en entrée une équation différentielle et un degré d, et renvoyant un approximant polynomial, de degré d, de la solution de l'équation sur le segment $[-1,1]$ ainsi qu'une borne supérieure de l'erreur de l'approximation. Ce polynôme est un très bon approximant car il est proche du développement en série de Tchebychev, tronqué à l'ordre d, de la solution.

Les séries de Tchebychev sont des cas particuliers des séries de Fourier généralisées. Ces séries (définies formellement dans le chapitre 8) sont des développements de fonctions dans des bases de fonctions (comme les polynômes orthogonaux). Les exemples classiques sont les séries de Taylor (base de fonctions x^n), les séries de Fourier (base de fonctions $\exp(2\pi inx)$) et bien sûr les séries de Tchebychev (base de fonctions $T_n(x)$). Les développements de fonctions dans des bases de Hilbert donnent un nombre important de nouveaux exemples. Un chapitre de cette thèse donne un début de généralisation des résultats sur les séries de Tchebychev à une plus large classe de séries de Fourier généralisées. On pourra de cette manière répondre au problème suivant :

Problème 1.7. La fonction $-\frac{x}{2}\,\mathrm{J}_1(x)$, où J est la fonction de Bessel de première espèce se développe dans la base des fonctions de Bessel comme

$$-\frac{x}{2}\,\mathrm{J}_1(x) = \sum_{n\in\mathbb{Z}} (-1)^n |n|\,\mathrm{J}_{2n}(x).$$

Comment prouver cette formule ?

Dans le chapitre 8, je montrerai comment prouver simplement cette formule en calculant une récurrence vérifiée par les coefficients de ce développement.

2 Prérequis et notations

Parmi les objets centraux de cette thèse se trouvent les polynômes et les séries de Tchebychev. Ces objets sont assez importants pour qu'un chapitre (voir chapitre 2) leur soit consacré.

Une autre notion fondamentale de cette thèse est la notion de D-finitude que je vais introduire dans la suite. Comme je l'utiliserai dans de nombreux chapitres, je définis aussi le modèle de complexité choisi.

2.1 Fonctions D-finies et suites P-récursives

Les fonctions différentiellement finies (*D-finies*) sont les solutions d'équations différentielles linéaires à coefficients polynomiaux. Ces fonctions sont très courantes en mathématiques et physique. Par exemple, la plupart des fonctions spéciales sont D-finies. Environ 60%

des fonctions du livre de Abramowitz et Stegun [AS64] (un des ouvrages les plus cités en science) sont D-finies [Sal05].

Les suites P-récursives sont les solutions des récurrences linéaires à coefficients polynomiaux. Environ 25% de l'encyclopédie de Sloane [SP95, Inc] sont P-récursives [Sal05].

Ce qui relie ces deux notions est que la suite des coefficients de Taylor d'une série D-finie est P-récursive.

Tous ces objets partagent de nombreuses propriétés qui sont énoncées dans le cadre de la « D-finitude ». Ces propriétés nous permettent de traiter ces objets par les équations qui les annulent. Ces équations forment ainsi une structure de données adaptée pour cette classe. Par exemple la fonction exp peut être définie comme l'unique solution de l'équation

$$y' - y = 0 \quad \text{et} \quad y(0) = 1,$$

et la suite $n!$ peut être définie comme l'unique solution de :

$$u_{n+1} - (n+1)u_n = 0 \quad \text{et} \quad u_0 = 1.$$

On retrouve une analogie avec la représentation des nombres algébriques par des équations polynomiales. Comme pour les nombres algébriques, il existe des propriétés de clôture entre fonctions D-finies (et entre suites P-récursives) [Sta80]. L'addition, la multiplication, le produit d'Hadamard de deux fonctions D-finies (suites P-récursives) sont D-finies (P-récursives). Comme pour les nombres algébriques, des algorithmes permettent de calculer les équations qui annulent la somme, le produit, le produit d'Hadamard de deux fonctions D-finies (suites P-récursives) à partir des équations qui les annulent [SZ94]. Il est possible d'évaluer numériquement un nombre algébrique à partir de l'équation polynomiale qui l'annule. Le problème semblable pour les suites est le calcul du $n^{\text{ème}}$ terme à partir de la récurrence qui l'annule, qui est facile à traiter en déroulant la récurrence. "Le problème semblable pour les fonctions D-finies, l'évaluation numérique via une équation différentielle, se traite aussi aisément en évaluant la série tronquée solution de l'équation.

2.2 Modèle de complexité

Dans cette thèse, le modèle de complexité utilisé est la complexité *arithmétique*. Cette mesure modélise précisément le temps de calcul pour des opérations sur les entiers modulaires ou sur des flottants machine. Nous étendons cette mesure au cas où les objets manipulés ne sont pas des entiers, mais plus généralement des éléments d'un corps $\mathbb{K}$ donné et nous mesurons alors la complexité en nombre d'opérations arithmétiques dans $\mathbb{K}$. Une addition ou une multiplication dans ce corps ont donc le même coût qui est 1. Ce modèle de complexité ne coïncide pas toujours avec la complexité binaire. Néanmoins, comprendre la complexité arithmétique constitue généralement une première étape utile vers la compréhension de la complexité binaire.

Notation 1.8. On utilisera la notation de Landau classique $\mathcal{O}$ pour exprimer le coût asymptotique d'un algorithme. La notation $\tilde{\mathcal{O}}(d)$ est utilisée lorsque le coût d'un algorithme

est asymptotiqement linéaire en d à des log près, on dit alors que l'algorithme est de complexité quasi-linéaire.

Un algorithme est dit optimal si le nombre d'opérations est asymptotiquement linéaire en la taille de son entrée ou de sa sortie. Il est dit quasi-optimal s'il est optimal à des log près.

La complexité arithmétique en calcul formel a été beaucoup étudiée ces dernières décennies. Il est bien établi que le coût de la multiplication de polynômes est une référence de mesure de complexité commutative et que le coût de la multiplication de matrices est une référence de complexité non-commutative dans le sens que les complexités des opérations entre polynômes (respectivement entre matrices) peuvent généralement être exprimées en terme de coût de multiplication de polynômes (respectivement de matrices) et pour beaucoup d'entre eux en temps quasi-linéaire [AHU74, BP94a, BCS97, Pan01, vzGG03].

Dans cette thèse, on se ramènera souvent à ces complexités de base pour la multiplication. Il est bien connu que deux polynômes de degré $< d$ peuvent être multipliés en temps $\mathsf{M}(d) = \mathcal{O}(d \log d \log\log d)$ en utilisant des algorithmes basés sur la transformée de Fourier rapide (FFT) [CT65, SS71, CK91], et que deux matrices $r \times r$ peuvent être multipliées en temps $\mathcal{O}(r^\omega)$, avec $2 \leqslant \omega \leqslant 3$ [Str69, Pan84, CW90]. Actuellement, la plus petite borne supérieure de ω est due à Vassilevska Williams [Vas12], et est $\omega < 2.3727$, suivant les travaux de Coppersmith et Winograd [CW90] et Stothers [Sto10]. Trouver la meilleure borne supérieure est un des problèmes ouverts les plus importants en théorie de la complexité algébrique.

3 Plan de la thèse

Les chapitres de ce manuscrit ne sont pas indépendants les uns des autres. Un diagramme des dépendances est proposé en figure 1.1. Le plan de ce mémoire est organisé de la manière suivante :

Polynômes et séries de Tchebychev Ce chapitre rappelle des propriétés des polynômes de Tchebychev et surtout introduit et énonce des résultats sur les séries de Tchebychev, objet central de cette thèse et moins courant dans la littérature.

Algorithmes efficaces pour des polynômes exprimés dans la base des Tchebychev Dans ce chapitre, j'expose deux algorithmes rapides, de complexité quasi-linéaire, permettant d'exprimer un polynôme dans la base de Tchebychev à partir de sa représentation dans la base monomiale. Ces algorithmes ne sont pas nouveaux mais ma contribution est de les expliciter et d'étudier plus précisément leurs complexités pour pouvoir les comparer.

Je présente aussi un algorithme quasi-optimal de multiplication de polynômes et un algorithme quasi-optimal de division euclidienne entre deux polynômes exprimés dans la base de Tchebychev. Le terme « quasi-optimal » signifie ici que l'on a une complexité très proche de celle connue pour les mêmes opérations lorsque les polynômes sont exprimés

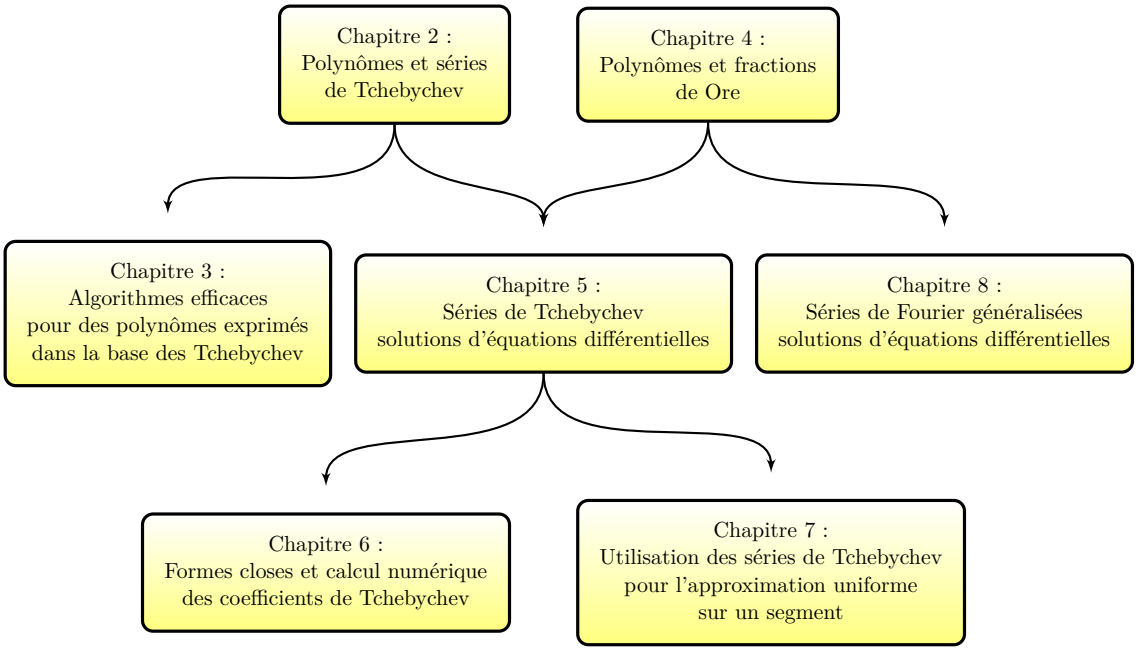


FIGURE 1.1 – Diagramme des dépendances des chapitres

dans la base monomiale. Les complexités obtenues sont meilleures que celles obtenues en effectuant ces mêmes opérations dans la base monomiale avec des conversions. L'algorithme de multiplication était déjà connu. L'algorithme de division rapide est par contre nouveau. Ce travail a été effectué avec Alin Bostan.

Polynômes et fractions de Ore Dans le chapitre 4, j'introduis les polynômes de Ore, objets classiques pour représenter les opérateurs différentiels ou de récurrence comme des polynômes. Ce chapitre donne aussi des résultats de complexité sur les opérations de multiplication et de division entre ces polynômes. Je présente un algorithme de multiplication rapide d'opérateurs, dû à van der Hoeven, dans le cas où les opérateurs sont des récurrences.

Une fois les polynômes de Ore présentés, j'expose la théorie des fractions d'opérateurs de récurrence qui est un objet incontournable de ma thèse. Ces objets sont très peu représentés et utilisés dans la littérature. Je présente donc des algorithmes permettant d'effectuer les opérations « de base » entre fractions.

Séries de Tchebychev solutions d'équations différentielles Lorsqu'une série de Tchebychev est solution d'une équation différentielle linéaire à coefficients polynomiaux, ses coefficients satisfont une récurrence linéaire. Dans ce chapitre, j'interprète cette récurrence comme le numérateur d'une fraction d'opérateurs de récurrence. Cette interprétation me

donne un point de vue simple des algorithmes existants pour calculer cette récurrence, me permet d'analyser leurs complexités, et d'en concevoir un plus rapide pour des ordres grands.

L'idée de cet algorithme rapide est d'utiliser le paradigme *diviser pour régner* avec une multiplication d'opérateurs de récurrence rapide (celle présentée dans le chapitre 4).

Ce chapitre est largement extrait d'un article écrit avec Bruno Salvy [BS09].

Formes closes et calcul numérique des coefficients de Tchebychev Ce chapitre expose une méthode permettant de représenter chaque coefficient de Tchebychev d'une fonction f comme l'évaluation d'une série entière en 1.

De cette représentation, je déduis des nouvelles preuves de formules donnant des formes closes pour les coefficients de Tchebychev de fonctions hypergéométriques.

Je donne aussi un nouvel algorithme de calcul d'approximations à précision arbitraire des coefficients de Tchebychev de fonctions D-finies en temps linéaire. Cet algorithme prend en entrée une équation différentielle (non singulière dans le disque unité) munie de conditions initiales en 0, un entier N représentant le nombre de coefficients à calculer et une erreur ϵ. Il rend la liste des approximations rationnelles des N premiers coefficients de Tchebychev avec une erreur ϵ en $\mathcal{O}(N + \log(\epsilon^{-1}))$ opérations arithmétiques, ce qui est la complexité asymptotique optimale pour ce problème.

Pour obtenir cet algorithme j'utilise la récurrence vérifiée par ces coefficients et calculée dans le chapitre 5. On voit dans ce chapitre que l'utilisation directe de cette récurrence, comme on l'utilise pour calculer les coefficients de Taylor, n'est pas bonne pour obtenir rapidement une approximation des coefficients de Tchebychev. L'utilisation que je propose est originale et plus astucieuse.

Utilisation des séries de Tchebychev pour l'approximation uniforme sur un segment Un large éventail de méthodes numériques existe pour calculer des approximations polynomiales de solution d'équation différentielle basées sur les séries de Tchebychev ou sur l'interpolation polynomiale en des points de Tchebychev. Dans ce chapitre je considère l'application de telles méthodes dans le contexte du calcul rigoureux, où nous avons besoin d'un moyen pour obtenir des garanties sur l'exactitude du résultat, en prenant en considération à la fois les erreurs de troncature et d'arrondi.

Étant donné un degré d et une fonction (définie comme solution d'une équation différentielle), la méthode proposée ici calcule un polynôme p_d de degré d et une borne fine B telle que $\|f - p_d\|_\infty < B$ en deux étapes.

- La première étape est le calcul d'un polynôme « candidat » pour approximer f. Afin d'obtenir une bonne approximation, on calcule un polynôme que l'on *espère* proche de la série de Tchebychev tronquée admettant de bonnes propriétés d'approximation.
- La seconde étape consiste au calcul des bornes. Cette étape est indépendante de la première. L'algorithme que l'on propose prend en entrée un polynôme et rend une borne de $\|f - p_d\|_\infty$. En utilisant le polynôme calculé dans la première partie, on obtient une bonne borne.

Afin d'obtenir ce polynôme d'approximation on utilise ici encore la récurrence vérifiée par les coefficients de Tchebychev. On n'utilise pas l'algorithme du chapitre 6 pour calculer les coefficients mais un nouvel algorithme plus simple et plus général mais moins fort dans la mesure où il ne garantit pas le résultat. La différence essentielle vient du fait que l'algorithme du chapitre précédent calcule les coefficients de Tchebychev avec une précision donnée *a priori* par l'utilisateur alors que la méthode de ce chapitre calcule une borne sur la « finesse » de l'approximation *a postériori*.

Ce chapitre représente une version préliminaire d'un travail en commun avec Mioara Joldeş et Marc Mezzarobba [BJM].

Séries de Fourier généralisées solutions d'équations différentielles Lorsque les séries de Fourier généralisées sont solutions d'équations différentielles linéaires et sous certaines conditions, les coefficients de ces séries vérifient une récurrence linéaire à coefficients polynomiaux.

Dans ce chapitre, je donne des conditions suffisantes pour que les coefficients vérifient une telle récurrence. Je présente aussi un nouvel algorithme qui permet de calculer cette récurrence. Cet algorithme est une généralisation de celui présenté dans le chapitre 5. Ici encore, c'est l'utilisation originale des fractions d'opérateurs de récurrence qui nous permet d'obtenir ces récurrences simplement. Ce travail est commun avec Bruno Salvy.

Pour aller un peu plus loin Un de mes projets futurs est d'obtenir des algorithmes rapides pour l'arithmétique de base entre les fractions d'opérateurs de récurrence. L'opération fondamentale de cette arithmétique est le calcul de ppcm à gauche entre deux opérateurs de récurrence. Des travaux récents [vdH11, BCLS12] ont donné des algorithmes rapides pour cette opération dans le cas très proche où les opérateurs sont différentiels.

Un des outils essentiels de ces algorithmes est la multiplication d'opérateurs différentiels rapides lorsque les degrés des coefficients polynomiaux des opérateurs sont grands par rapport à l'ordre. Il existe un algorithme de multiplication dû à van der Hoeven [vdH02] qui est optimal lorsque les degrés et les ordres des opérateurs sont de tailles semblables. Une collaboration avec Alin Bostan et Joris van der Hoeven [BBvdH12] a donné un nouvel algorithme permettant d'effectuer cette multiplication rapidement même lorsque les degrés et les ordres ne sont pas proches. On obtient de cette manière une complexité quasi-optimale pour ce problème.

Ce travail étant un peu dans la suite de ma thèse, j'ai décidé de le placer tel quel dans l'annexe B et non d'en faire un chapitre.

4 Contributions logicielles

Lors de cette thèse, j'ai développé du code en rapport avec mes résultats. La plupart de mon code a été développé en Maple. Ce code a en partie été intégré dans The Dynamic Dictionary of Mathematical Functions (DDMF), un site web interactif, inspiré du Handbook

of Mathematical Functions d'Abramowitz et Stegun [AS64], et qui se veut une vitrine de l'approche par la D-finitude des fonctions spéciales. Ce site web est développé au sein du projet du même nom au Centre de Recherche Commun INRIA-Microsoft Research.

Package : Chebyshev Ce package Maple reprend les algorithmes développés dans les chapitres 5 et 6. Ce code est intégré dans la section Chebyshev Expansion du DDMF.

La figure 4 représente la page web de la fonction erreur dans la version 1.7.2 du DDMF `http://ddmf.msr-inria.inria.fr`. Cette page est centrée sur la section 8 qui est le développement en série de Tchebychev de la fonction erreur. Les formules présentes sont calculées automatiquement à partir de l'équation différentielle et des conditions initiales de la section 1 avec ce package.

La récurrence vérifiée par les coefficients de Tchebychev est calculée par l'algorithme de la section 5. Dans cette section, on trouve aussi les coefficients de Tchebychev de deux façons différentes. La première est l'expression des coefficients sous forme close. Ces formes closes sont calculées automatiquement à partir de l'équation différentielle par la méthode présentée dans le chapitre 6. La seconde est la représentation des coefficients par des approximations numériques. C'est l'algorithme du calcul des coefficients du chapitre 6 qui est utilisé. Le DDMF étant interactif, il est possible de demander plus de termes du développement. L'algorithme qui calcule ces approximations de coefficients utilise la récurrence de Tchebychev, le calcul de plusieurs termes en plus s'effectue donc très rapidement. Il en coûtera moins d'une seconde de calculer les 20 000 premiers coefficients de Tchebychev de la fonction erf.

The Dynamic Dictionary of Mathematical Functions (DDMF) Pour afficher les formes closes des coefficients de Tchebychev, j'ai vite été limité par la forme des solutions renvoyées par Maple. Les formules obtenues par Maple, notamment par la fonction rsolve, ne sont pas toujours aussi « jolies » que les formules que l'on trouve dans les livres comme [AS64]. Souvent des simplifications (même en appliquant simplify) qui rendent la formule plus jolie pour un lecteur ne sont pas effectuées. Un exemple assez concret, la simplification de terme par la formule de duplication pour la fonction gamma

$$\Gamma(n)\Gamma(n+1/2) = 2^{1-2n}\sqrt{\pi}(2n-1)!,$$

n'est jamais utilisée par défaut en Maple.

Pour le DDMF, on a besoin de construire automatiquement à partir des équations différentielles des jolies formules pour les développements en séries des solutions. Le code que j'ai développé dans un package Maple nommé Hypergeometric permet d'obtenir ce genre de formules proches de celles que l'on trouve dans les formulaires.

Toutes les formes closes des développements en séries du DDMF sont calculées à partir de ce package. Comme la plupart des coefficients des développements de Taylor

The Special Function $\mathrm{erf}\,(x)$

[-] 1. Differential equation

rendering link

The function $\mathrm{erf}\,(x)$ satisfies

$$2\left(\frac{d}{dx}y\,(x)\right)x+\frac{d^2}{dx^2}y\,(x)=0$$

with initial values $y\,(0)=0$, $(y')\,(0)=2\,\frac{1}{\sqrt{\pi}}$.

[+] 2. Plot

[+] 3. Numerical Evaluation

[+] 4. Symmetry

[+] 5. Taylor Expansion at 0

[+] 6. Local Expansions at Singularities and at Infinity

[+] 7. Hypergeometric Representation

[-] 8. Chebyshev Expansion over $[-1,1]$

- Chebyshev expansion:

$$\mathrm{erf}\,(x)=\sum_{n=0}^{\infty}2\quad\frac{4^{-n}\,(-1)^n\,{}_1F_1(1/2+n;\,2\,n+2;-1)T_{2\,n+1}\,(x)}{\sqrt{\pi}\,(2\,n+1)\,n!}.$$

- First terms and polynomial approximation:

$$\mathrm{erf}\,(x)=0.904347\,T_1\,(x)-0.0661130\,T_3\,(x)+0.00472936\,T_5\,(x)+\ldots$$

$$\mathrm{erf}\,(x)\approx 1.12633280\,x-0.35903920\,x^3+0.07566976\,x^5.$$

order = 6 Envoyer

- The coefficients c_n in the Chebyshev expansion $\mathrm{erf}\,(x)=\sum_{n=0}^{\infty}c_nT_n(x)$ satisfy the recurrence

$$\left(n^2+3\,n\right)c\,(n)+\left(2\,n^3+12\,n^2+24\,n+16\right)c\,(n+2)+\left(-n^2-5\,n-4\right)c\,(n+4)=0.$$

- Approximations by successive truncations of the Chebyshev series on $[-2,2]$:

FIGURE 1.2 – Page de la fonction erreur dans le DDMF.

du DDMF sont hypergéométriques, il est immédiat d'obtenir les fonctions sous forme de fonction hypergéométrique une fois les formes closes calculées. La section « Hypergeometric representation » du DDMF est aussi calculée à partir de ce code.

Ce package ne se veut en aucun cas un nouveau programme pour résoudre des récurrences mais simplement pour obtenir de jolies formules permettant d'exprimer les solutions de récurrence à partir de code déjà existant.

Les exemples suivants illustrent mon code en comparant les formules obtenues par Maple et les formules que j'obtiens avec mon package.

Exemple 1.9 (Fonction erf)**.** À partir de l'équation différentielle et des conditions initiales vérifiées par erf, il n'est pas difficile de voir que le développement de Taylor en 0 de cette fonction est de la forme

$$\sum_{n=0}^{\infty} u_n x^n,$$

où u_n vérifie la récurrence :

$$2nu_n + \left(n^2 + 3n + 2\right) u_{n+2} = 0 \quad \text{avec} \quad u_0 = 0 \quad \text{et} \quad u_1 = \frac{2}{\sqrt{\pi}}.$$

On peut donner une forme close de cette fonction en utilisant rsolve de la façon suivante :

```
> rec, ic;
```

$$\left\{2\,n\,u(n) + \left(n^2 + 3\,n + 2\right) u(n+2)\right\}, \left\{u(0) = 0, u(1) = \frac{2}{\sqrt{\pi}}\right\}$$

```
> rsolve({op(rec1), op(ic)}, u(n));
```

$$\begin{cases} 0 & n::even \\ \dfrac{2\,(-1)^{\frac{1}{2}n - \frac{1}{2}}}{n\,\Gamma\left(\frac{1}{2}n + \frac{1}{2}\right)\sqrt{\pi}} & n::odd \end{cases}$$

```
> % assuming n::odd:
  Sum(%*x^(2*n+1), n=0..infinity);
```

$$\sum_{n=0}^{\infty} \frac{2\,(-1)^{\frac{1}{2}n - \frac{1}{2}}\, x^{2n+1}}{n\,\Gamma\left(\frac{1}{2}n + \frac{1}{2}\right)\sqrt{\pi}}$$

On remarque que l'on peut rendre cette formule plus jolie en simplifiant la fonction gamma en utilisant la formule de duplication. C'est ce que fait mon code.

```
> sol := generalTerms(rec1, ic, n ,x):
  sol[u];
  sol[u][general];
```

$$Record\left(poly = 0, general = [Record(\ldots), Record(\ldots)], hyperSeries = \frac{2\,x\,\text{hypergeom}\left(\left[\frac{1}{2}\right], \left[\frac{3}{2}\right], -x^2\right)}{\sqrt{\pi}}, isHyper = true\right)$$

$$\left[Record(cform = 0, period = 2\,n), Record\left(cform = \frac{2\,(-1)^n}{\sqrt{\pi}\;(2\,n+1)\,n!}, period = 2\,n+1\right)\right]$$

```
> Sum(add(sol1[cform]*x^sol1[period], sol1 in sol[u][general]), n=0..
  infinity);
```

$$\sum_{n=0}^{\infty} \frac{2\,(-1)^n\,x^{2\,n+1}}{\sqrt{\pi}\;(2\,n+1)\,n!}$$

```
> sol[u][hyperSeries];
```

$$\frac{2\,x\,\text{hypergeom}\left(\left[\frac{1}{2}\right], \left[\frac{3}{2}\right], -x^2\right)}{\sqrt{\pi}}$$

La procédure generalTerms que j'ai écrite renvoie un Record. Le champ general donne les formes closes des coefficients de la solution. Le champ cform est la forme close du coefficient associé au monôme d'exposant period dans la série génératrice. En sommant le tout, on obtient une formule pour la série de Taylor. Cette formule est la même que celle retrouvée dans les formulaires classiques sur les fonctions.

La fonction generalTerms renvoie aussi la forme hypergéométrique de la série génératrice lorsqu'elle existe.

Exemple 1.10. Les coefficients de Taylor de la fonction d'Airy Ai(x) vérifient la récurrence :

$$-u_n + (n^2 + 5n + 6)u_{n+3} = 0.$$

Avec mon code on obtient le développement :

```
> sol := generalTerms(rec1, ic, n ,x):
  sol[u]:
  sol[u][general];
```

$$\left[Record\left(cform = \frac{1}{3}\,\frac{3^{1/3}}{9^n\,\Gamma\left(n+\frac{2}{3}\right)n!},\ period = 3\,n\right),\ Record\left(cform = -\frac{1}{9}\,\frac{3^{2/3}}{9^n\,\Gamma\left(n+\frac{4}{3}\right)n!},\ period = 3\,n+1\right),\ Record(cform = 0,\ period = 3\,n+2)\right]$$

```
> Sum(add(sol1[cform]*x^sol1[period], sol1 in sol[u][general]), n=0..
  infinity);
```

$$\sum_{n=0}^{\infty}\left(\frac{1}{3}\,\frac{3^{1/3}\,x^{3\,n}}{9^n\,\Gamma\left(n+\frac{2}{3}\right)n!} - \frac{1}{9}\,\frac{3^{2/3}\,x^{3\,n+1}}{9^n\,\Gamma\left(n+\frac{4}{3}\right)n!}\right)$$

Exemple 1.11. Le dernier exemple est le développement en 0 de la fonction dilog . Cette fonction est singulière en 0. À partir de l'équation différentielle et des conditions initiales, on peut montrer qu'elle se développe comme

$$\frac{1}{6}\pi^2 \operatorname{dilog}(x) := \sum_{n=0}^{\infty} u_n x^n + \sum_{n=0}^{\infty} (v_n + w_n \ln(x))\, x^n,$$

où les suites u_n, v_n, w_n vérifient les récurrences :

$$\begin{aligned}
&n^2(n+1)u_{n+1} - n^3 u_n = 0\\
&n^2(n+1)v_{n+1} - n(2+3n)w_{n+1} + n^3 v_n + 3n^2 w_n = 0\\
&n^2(n+1)w_{n+1} - n^3 w_n = 0,
\end{aligned}$$

munies des conditions initiales :

$$u_0 = 0, u_1 = 1, v_0 = 0, v_1 = 0, v_2 = 1/4, w_0 = 0, w_1 = 1.$$

On est dans un cas particulier où on a plusieurs récurrences et où elles sont liées entre elles. Mon code est fait pour gérer ce genre de cas comme le montre cet exemple.

```
> sol := generalTerms(rec, ic, n ,x):
  sol[u[0]][general],sol[u[1]][general], sol[u[2]][general];
```

$$\left[Record\left(cform = \frac{1}{n+1}, period = n+1\right)\right], \left[Record\left(cform = \frac{n+1}{(n+2)^2}, period = n+2\right)\right],$$
$$\left[Record\left(cform = \frac{1}{n+1}, period = n+1\right)\right]$$

```
> Sum(add(sol1[cform]*x^sol1[period], sol1 in sol[u[0]][general])+add
  (sol1[cform]*x^sol1[period], sol1 in sol[u[1]][general]), n=0..
  infinity)+ln(x)*Sum(add(sol1[cform]*x^sol1[period], sol1 in sol[u
  [2]][general]), n=0..infinity);
```

$$\sum_{n=0}^{\infty}\left(\frac{x^{n+1}}{n+1}+\frac{(n+1)\,x^{n+2}}{(n+2)^2}\right)+\ln(x)\left(\sum_{n=0}^{\infty}\frac{x^{n+1}}{n+1}\right)$$

Pour intégrer mes programmes Maple sur la page web du DDMF, j'ai implémenté du code en DynaMoW, un langage développé par Frédéric Chyzak et Alexis Darasse pour le DDMF [CD11]. Ce langage est implémenté comme une extension de OCaml. Il sert d'interface avec des logiciels de calcul formel, essentiellement Maple pour le moment. Il permet aussi d'assembler des pages web en donnant la possibilité d'insérer les formules renvoyées par le logiciel de calcul formel. Ces formules sont aussi manipulables par ce langage.

Mon travail sur le DDMF a donné lieu à une publication avec les autres membres du projet [BCD+10]. Ce travail est mis en annexe A tel quel.

Code en cours de développement Parallèlement à notre travail avec Mioara Joldeş et Marc Mezzarobba (voir chapitre 7), nous avons commencé à développer du code qui prend en entrée une équation différentielle, des conditions initiales et un degré d et renvoie une approximation polynomiale de degré d de la solution de l'équation différentielle et une borne d'erreur de cette approximation. Ce code nous donne des résultats expérimentaux de notre méthode très encourageants comme on peut le voir dans le chapitre 7.

Cette méthode est pour le moment programmée en Maple. Nous aimerions par la suite la coder en **C** pour pouvoir utiliser une meilleure bibliothèque d'arithmétique des intervalles.

Je suis aussi en train d'écrire le package GFSRecurrence qui permet de calculer les récurrences vérifiées par les coefficients des séries de Fourier généralisés en utilisant les méthodes du chapitre 8 (des exemples de mon code sont donnés dans ce chapitre).

Ces méthodes s'appuient fortement sur l'arithmétique des fractions d'opérations de récurrence. Afin d'obtenir une arithmétique simple à manipuler, j'ai surchargé les opérateurs de base dans le package OreField. Des exemples de son utilisation sont donnés tout au long du chapitre 4.

Chapitre 2

Polynômes et séries de Tchebychev

Résumé

Dans ce chapitre, une présentation des séries de Tchebychev est donnée. Avant de définir les séries de Tchebychev, des propriétés des polynômes de Tchebychev sont rappelées.

Sommaire

Les résultats de ce chapitre peuvent se trouver dans les ouvrages consacrés en grande partie ou intégralement aux polynômes et aux séries de Tchebychev [Sny66, AS64, Sze75, NU88, Riv90, Boy01, MH03].

1 Polynômes de Tchebychev

1.1 Définition et propriétés de base

Les polynômes de Tchebychev[1] forment l'une des familles de polynômes les plus populaires, notamment pour ses applications en approximation. Une remarque attribuée à de nombreux mathématiciens et numériciens, citée au début de [MH03], témoigne de cette importance :

"*Chebyshev polynomials are everywhere dense in numerical analysis.*"

Il y a différentes sortes de polynômes de Tchebychev. Dans cette thèse, seuls les polynômes de Tchebychev de première espèce, notés T_n, seront évoqués. Le terme « polynôme de Tchebychev » est réservé à cette famille. Au passage, beaucoup d'articles et de livres utilisent ce terme en se référant uniquement aux polynômes de Tchebychev de première espèce, cette appellation n'est donc pas déraisonnable.

Il y a plusieurs façons de définir des polynômes de Tchebychev. On utilise ici une des définitions les plus simples et les plus répandues [Sze75, NU88, Riv90, Boy01, MH03].

Définition 2.1. Le polynôme de Tchebychev $T_n(x)$ avec $n \in \mathbb{Z}$ est défini par la relation

$$T_n(x) = \cos(n\theta) \quad \text{où} \quad x = \cos(\theta). \tag{2.1}$$

Remarque 2.2. Par parité de la fonction cos, pour tout $n \in \mathbb{Z}$, les polynômes T_n vérifient

$$T_n(x) = T_{-n}(x). \tag{2.2}$$

Remarque 2.3. Une autre conséquence qui sera importante dans la suite du manuscrit est la relation

$$T_n\left(\frac{x + x^{-1}}{2}\right) = \frac{x^n + x^{-n}}{2}. \tag{2.3}$$

De la définition 2.1, on déduit la propriété suivante qui prouve que T_n est bien un polynôme de degré n.

Propriété 2.4. *Les polynômes de Tchebychev vérifient la relation de récurrence à trois termes*

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \tag{2.4}$$

1. Tchebychev, quand il écrivait en français, transcrivait lui-même son nom du cyrillique comme « Tchebycheff », mais apparemment une transcription plus moderne de ce nom est Tchebychev. En anglais, la version la plus courante de son nom est « Chebyshev ».

avec les conditions initiales :

$$T_0(x) = 1 \quad et \quad T_1(x) = x. \tag{2.5}$$

Cette propriété est une conséquence immédiate de l'identité trigonométrique :

$$\cos(n+1)\theta + \cos(n-1)\theta = 2\cos\theta\cos n\theta.$$

En partant de la définition (2.1), on retrouve bien la récurrence à trois termes (2.4). Il est aussi classique de définir les polynômes T_n par cette équation.

Exemple 2.5. Une application immédiate de la propriété 2.4 est le calcul explicite des premiers polynômes de Tchebychev.

$$\begin{aligned}
T_0(x) &= 1,\\
T_1(x) &= x,\\
T_2(x) &= -1 + 2x^2,\\
T_3(x) &= 4x^3 - 3x,\\
T_4(x) &= 1 + 8x^4 - 8x^2,\\
T_5(x) &= 16x^5 - 20x^3 + 5x,\\
&\vdots
\end{aligned}$$

Favard [Fav35] a montré comment, à partir de cette récurrence, définir une mesure afin d'obtenir un produit scalaire qui rend les polynômes T_n orthogonaux entre eux. Le produit scalaire obtenu est le suivant :

$$\frac{2}{\pi}\int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\mathrm{d}x = \begin{cases} 1 & \text{si} \quad n = m \neq 0\\ 2 & \text{si} \quad n = m = 0\\ 0 & \text{sinon} \end{cases} \tag{2.6}$$

À partir de ce produit scalaire et en effectuant plusieurs intégrations par parties [MH03, §4.5], on obtient

$$T_n(x) = \frac{(-1)^n 2^n n!}{(2n)!}\sqrt{1-x^2}\frac{\mathrm{d}^n}{\mathrm{d}x^n}(1-x^2)^{n-\frac{1}{2}}. \tag{2.7}$$

Cette formule est la formule de *Rodrigues* pour les polynômes de Tchebychev (voir [AO05, §6] pour l'histoire intéressante de cette formule).

De cette formule, on peut vérifier par substitution que le polynôme T_n est solution de l'équation différentielle.

$$(1-x^2)y'' - xy' + n^2 y = 0. \tag{2.8}$$

Munie de conditions initiales (par exemple $y(0) = \sin\frac{\pi(n+1)}{2}$ et $y'(0) = -n\sin\frac{\pi(n+2)}{2}$), on retrouve une nouvelle définition équivalente à (2.1). C'est d'ailleurs par l'équation différentielle que les polynômes T_n sont définis dans [Sny66].

Cette équation entraîne que les polynômes T_n sont des cas particuliers des polynômes de Gegenbauer (ou « ultrasphériques ») C_n^α lorsque $\alpha = 1/2$ (à une multiplication par un polynôme en n près). Les polynômes de Gegenbauer étant eux-mêmes des cas particuliers des polynômes de Jacobi $P_n^{a,b}$ lorsque $a = \alpha - 1/2$ et $b = \alpha - 1/2$ (voir [KS98] pour une classification plus complète). Plus généralement l'ensemble de ces polynômes appartiennent à la classe des *polynômes orthogonaux classiques*.

Une autre propriété dérivée du produit scalaire est résumée ci-dessous.

Propriété 2.6. *La famille T_n forme une base de l'espace de Hilbert $L_2(w)$, où w est la mesure du produit scalaire (c'est-à-dire $w = \frac{1}{\sqrt{1-x^2}}$ sur $[-1, 1]$).*

Une autre propriété fondamentale dans cette thèse sur les polynômes de Tchebychev est déduite aisément de (2.1).

Propriété 2.7. *Les polynômes de Tchebychev vérifient l'équation*

$$2(1-x^2)T_n'(x) = -nT_{n+1}(x) + nT_{n-1}(x). \tag{2.9}$$

Cette relation s'obtient par la suite d'égalités suivantes :

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}T_n(x) &= \frac{\mathrm{d}\cos n\theta}{\mathrm{d}\theta} \Big/ \frac{\mathrm{d}\cos\theta}{\mathrm{d}\theta} \\ &= \frac{n\sin n\theta}{\sin\theta} \\ &= \frac{\frac{1}{2}n\left(T_{n-1}(x) - T_{n+1}(x)\right)}{1-x^2}.\end{aligned}$$

1.2 Développement en série

Les polynômes de Tchebychev se développent en série de Taylor en 0 avec la formule

$$T_j(x) = \frac{j}{2}\sum_{k=0}^{\left\lfloor\frac{j}{2}\right\rfloor}(-1)^k\frac{(j-k-1)!}{k!(j-2k)!}(2x)^{j-2k} \quad \text{pour } j \geqslant 1. \tag{2.10}$$

Cette formule se vérifie par une récurrence assez simple en utilisant (2.4). Elle nous donne explicitement les polynômes T_n en fonction de x, ce qui nous permet de calculer le $n^{\text{ème}}$ coefficient en $\mathcal{O}(n)$ opérations arithmétiques. Il en faudrait $\mathcal{O}(n^2)$ en utilisant (2.4).

Une autre application de cette formule est l'écriture sous forme hypergéométrique des polynômes de Tchebychev :

$$\begin{aligned}T_{2n}(x) &= (-1)^n {}_2F_1\left(\left.\begin{matrix}n,-n\\ \frac{1}{2}\end{matrix}\right| x^2\right), \\ T_{2n+1}(x) &= (-1)^n(2n+1)x\,{}_2F_1\left(\left.\begin{matrix}n+1,-n\\ \frac{3}{2}\end{matrix}\right| x^2\right),\end{aligned} \tag{2.11}$$

où ${}_2F_1$ est une série hypergéométrique, cas particulier de la définition ci-dessous.

Définition 2.8. La série hypergéométrique généralisée ${}_pF_q$ est définie par :

$$ {}_pF_q\left(\left.\begin{matrix} a_1;\cdots;a_p \\ b_1;\ldots;b_q \end{matrix}\right| x\right) = \sum_{n\in\mathbb{N}} \frac{(a_1)_n\cdots(a_p)_n}{(b_1)_n\cdots(b_q)_n}\frac{x^n}{n!}, \tag{2.12} $$

où $(\cdot)_n$ désigne le symbole de Pochhammer, c'est-à-dire :

$$ (a)_n := \frac{(a+n-1)!}{(a-1)!} = a(a+1)\cdots(a+n-1). \tag{2.13} $$

Cette représentation sous forme hypergéométrique n'est pas la plus classique. Cela est surement dû à la présence de deux formules afin de différencier les indices pairs et impairs. La formule la plus fréquente dans la littérature, qui se déduit facilement de l'équation (2.9) après un changement de variable, est :

$$ T_n(x) = {}_2F_1\left(\left.\begin{matrix} n,-n \\ \frac{1}{2} \end{matrix}\right| -\frac{x-1}{2}\right). \tag{2.14} $$

De la récurrence (2.4), on déduit aussi que la série génératrice (indexée sur $\mathbb{Z}$) des polynômes de Tchebychev vérifie la formule simple

$$ \sum_{n\in\mathbb{Z}} T_n(x)z^{|n|} = \frac{1}{2}\frac{1-z^2}{1-2xz+z^2}. \tag{2.15} $$

1.3 Quelques propriétés d'approximation

Dans cette section, on ne souhaite pas être exhaustif sur les propriétés d'approximation des polynômes de Tchebychev. On souhaite juste donner une application simple de ces propriétés et la relation entre le problème « minimax » et les polynômes de Tchebychev.

Problème 2.9. Étant donnés une fonction f continue sur $[-1,1]$ et un degré n, le problème « minimax » consiste à calculer un polynôme p_n de degré inférieur ou égal à n tel que :

$$ \|f-p_n\|_\infty = \min_{p\in\mathbb{R}_{n+1}[x]} \max_{-1\leqslant x\leqslant 1} |f(x)-p_n(x)|, $$

où $\mathbb{R}_{n+1}[x]$ est l'ensemble des polynômes de degré strictement plus petit que $n+1$ et $\|\cdot\|_\infty$ est la norme uniforme sur $[-1,1]$.

Un théorème attribué à Tchebychev ou à Borel (cela dépend des auteurs voir [MH03]) nous donne des informations sur le polynôme répondant au problème.

Théorème 2.10. *Pour toute fonction continue sur $[-1,1]$, il existe un unique polynôme p_n de degré plus petit que n qui répond au problème minimax, et il est uniquement caractérisé par la « propriété d'alternance » : il existe au moins $n+2$ points dans $[-1,1]$ pour lesquels $f(x)-p_n(x)$ atteint sa valeur maximum avec des signes qui alternent.*

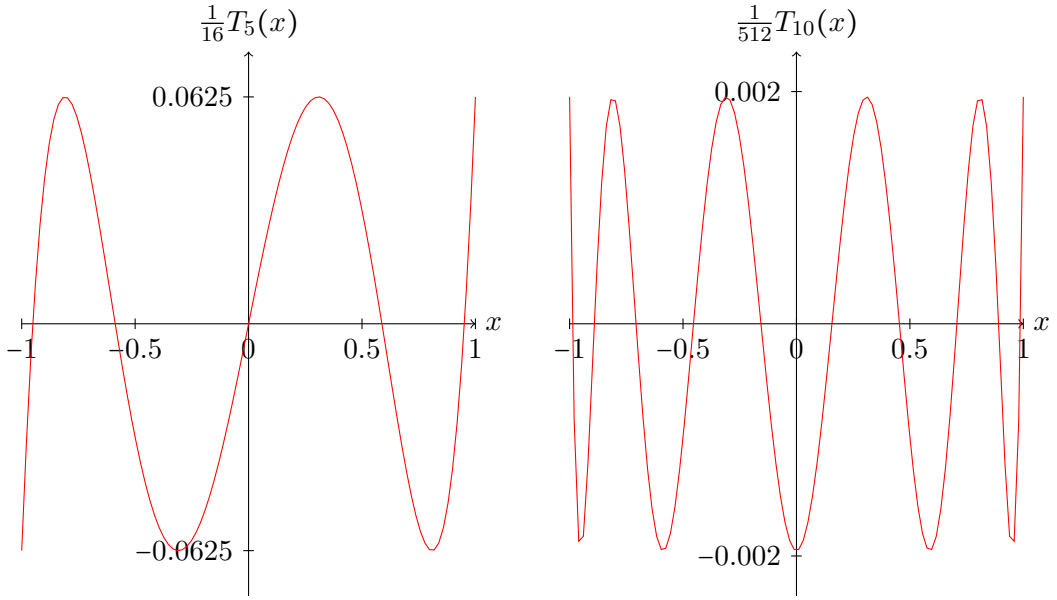


FIGURE 2.1 – Erreurs d'approximations de x^n par $p_{n-1} = x^n - 2^{1-n}T_n$ pour $n = 5$ et $n = 10$

Par la définition (2.1), il est facile de voir que le polynôme T_n atteint $n+1$ fois ses extremums qui sont de magnitude 1 avec des signes alternant en les points $x = \cos\frac{k\pi}{n}$ pour $k = 0, \dots n$).

Une conséquence immédiate de cette propriété est que le meilleur approximant de degré n du monôme x^{n+1} est le polynôme $p_n = x^{n+1} - 2^{-n}T_{n+1}$. En effet, selon l'équation (2.10), ce polynôme est de degré n. De plus selon la remarque ci-dessus, le polynôme $x^{n+1} - p_n = 2^{-n}T_{n+1}(x)$ atteint ses extremums en $n+2$ points comme le montre la figure 2.1.

Une application immédiate de ces approximations est donnée par les « économisations de séries » introduites dans § 1.

2 Séries de Tchebychev

En utilisant le notion d'orthogonalité des polynômes de Tchebychev, on introduit les séries de Tchebychev. Ces séries sont centrales dans ma thèse et sont le sujet de plusieurs chapitres. L'utilité de ces séries est essentiellement liée aux problèmes d'approximations sur un segment avec la norme uniforme.

S'il y a hésitation sur le choix d'une base de polynômes pour développer une fonction afin de l'approximer, Boyd [Boy01, p. 10] nous donne son principe moral :

1. *When in doubt, use Chebyshev polynomials unless the solution is spatially periodic, in which case an ordinary Fourier series is better*;
2. *Unless you're sure another set of basis functions is better, use Chebyshev polynomials*;

3. *Unless you're really, really sure that another set of basis functions is better, use Chebyshev polynomials.*

2.1 Définition

Comme la famille des polynômes de Tchebychev forme une base de $L_2(w)$, on peut développer toute fonction de cet espace dans cette base.

Définition 2.11. Les développements des fonctions $f \in L^2(w)$ dans la base T_n sont appelés séries de Tchebychev. Les séries de Tchebychev sont écrites selon la notation la plus usuelle :

$$f(x) = {\sum_{n\in\mathbb{N}}}' c_n T_n(x) = \frac{c_0}{2}T_0(x) + c_1T_1(x) + c_2T_2(x) + \cdots. \tag{2.16}$$

Il suit, en prenant le produit scalaire, que les coefficients c_n sont définis par

$$c_n = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_n(x)}{\sqrt{1-x^2}}dx. \tag{2.17}$$

Remarque 2.12. Par l'égalité (2.2), une manière équivalente d'écrire les séries de Tchebychev est :

$$f(x) = \sum_{n=-\infty}^{\infty} \frac{c_n}{2}T_n(x). \tag{2.18}$$

En effet, $T_n = T_{-n}$ et par l'égalité (2.17) $c_n = c_{-n}$. Cette écriture fait le lien entre les séries de Tchebychev et les séries de Laurent et permet de définir les indices des coefficients de Tchebychev sur $\mathbb{Z}$.

L'écriture (2.18) permet aussi d'expliquer la notation $'$ dans l'écriture (2.16). Ce symbole vient du fait que quand les coefficients sont indexés sur $\mathbb{Z}$ le terme c_0 n'est calculé qu'une fois.

L'égalité (2.1) montre que ces séries se comportent comme des séries de Fourier avec un changement de variable. Ainsi on voit immédiatement que cette série converge vers f sur $[-1,1]$ si f est une fonction continue. La convergence est uniforme si f satisfait la condition de Dini-Lipschitz ou est une fonction à variation bornée (et *a fortiori* si f est dérivable), voir [GST07, MH03].

2.2 Propriétés analytiques

On suppose maintenant que f est analytique dans un voisinage de $[-1,1]$ et admet un nombre fini de singularités (ce qui est le cas des fonctions D-finies). Soit

$$E_r = \{x \in \mathbb{C} : |x + \sqrt{x^2-1}| < r\}, \tag{2.19}$$

une ellipse de foyers en ± 1. On s'intéresse à la plus grande ellipse E_r telle que f soit analytique dans E_r (voir figure 2.2). Comme le nombre de points singuliers est fini, on a $0 < r \leqslant \infty$. Un théorème classique [MH03, Theorem 5.16] nous donne une relation entre r et les coefficients de la série de Tchebychev.

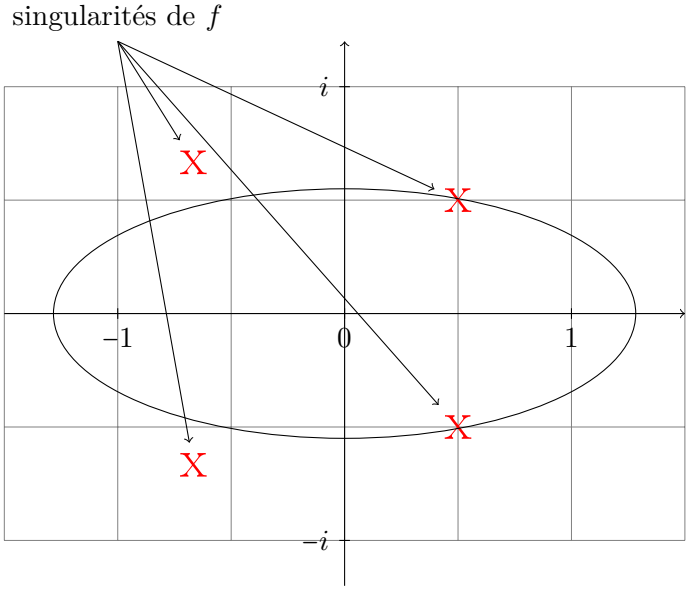


FIGURE 2.2 – Représentation de la plus grande ellipse E_r telle que f est analytique dans E_r lorsque ses singularités sont $\frac{1}{2} \pm \frac{1}{2}i$ et $\frac{2}{3} \pm \frac{2}{3}i$

Théorème 2.13. *Si la fonction $f(x)$ est analytique dans E_r, où $r > 1$, alors f est égale à son développement de Tchebychev dans E_r et en utilisant les notations (2.16), on a*

$$\left| f(x) - \sum_{n=0}^{k} {}' c_n T_n(x) \right| = O(r^{-k}), \quad \textit{pour} \quad x \in E_r.$$

Définition 2.14. Soit $\mathcal{C} \subset \mathbb{C}^{\mathbb{Z}}$ l'espace vectoriel des suites $(c_n)_{n\in\mathbb{Z}}$ telles que

$$(\forall n \in \mathbb{N})(c_n = c_{-n}) \qquad \text{et} \qquad (\exists \alpha < 1)(c_n = \mathcal{O}(\alpha^n)).$$

Les suites de coefficients de Tchebychev d'une fonction f qui est analytique dans un voisinage de $[-1, 1]$ sont dans $\mathcal{C}$. Inversement, pour toute suite $c \in \mathcal{C}$, la série $\sum_{n\in\mathbb{N}}{}' c_n T_n(x)$ converge uniformément dans un voisinage de $[-1, 1]$ vers une fonction analytique $f(x)$.

2.3 Propriétés « proche du minimax » des séries de Tchebychev

Les séries de Tchebychev tronquées sont des très bons approximants de fonction analytique dans des ellipses comme le montre le théorème 2.13. Elles ne sont pas les meilleurs approximants (*minimax*) pour un degré donné, mais ne sont pas loin. La notion de « proche du minimax » (*near-minimax* en anglais) permet de définir cette proximité.

Définition 2.15. Soit f une fonction continue dans $[-1,1]$, une approximation polynomiale f_n de degré n est dite « proche du minimax » avec une distance relative λ_n si

$$\|f - f_n\|_\infty \leqslant (1+\lambda_n)\,\|f - p_n\|_\infty\,,$$

où ρ est un réel positif et p_n est l'approximant « minimax » de f.

Le réel λ_N est appelé la *constante de Lebesgue* [MH03, §5.5].

Cheney et Price [CP70] ont donné une estimation asymptotique précise de λ_n lorsque p_n est le développement de Tchebychev tronqué de f.

$$\lambda_n = \frac{4}{\pi^2}\log n + 1.2703 + \cdots + O(1/n).$$

On en déduit que

$$\|f - \pi_n(f)\|_\infty \leqslant \Big(\frac{4}{\pi^2}\log n + \mathcal{O}(1)\Big)\|f - p_n^*\|_\infty \tag{2.20}$$

où p_n est le polynôme de degré n qui minimise $\|f-p\|_\infty$ et $\pi_n(f) = \sum_{k=0}^{n}{}' c_k T_k$, avec les c_k définis par (2.17).

Même si p_n peut-être calculé avec une précision arbitraire en utilisant l'algorithme de Remez [Che98, Chap. 3], on ne perd pas beaucoup en prenant à la place les polynômes $\pi_n(f)$.

Une des motivations pour utiliser les séries tronquées plutôt que d'autres approximations « proche du minimax » ou même p_n qui ont aussi de belles propriétés analytiques est l'existence d'une relation de récurrence entre les coefficients c_n lorsque la fonction f est D-finie. Une grande partie de cette thèse est consacrée aux calcul et à l'utilisation de cette structure (chapitres 5, 6 et 7).

Chapitre 3

Algorithmes efficaces pour des polynômes exprimés dans la base des Tchebychev

Résumé

Ce chapitre présente des algorithmes rapides pour des calculs entre polynômes exprimés dans la base des polynômes de Tchebychev. Des algorithmes de multiplication rapide sont rappelés et un nouvel algorithme de division rapide de polynômes est proposé. Ce chapitre décrit aussi deux algorithmes, de complexité quasi-linéaire, permettant de passer d'un polynôme exprimé dans la base des Tchebychev à un polynôme exprimé dans la base monomiale ainsi que l'opération inverse. Ces derniers algorithmes ne sont pas nouveaux mais la contribution de ce chapitre est de les expliciter et d'étudier plus précisément leurs complexités pour pouvoir les comparer.

Ce travail est commun avec Alin Bostan.

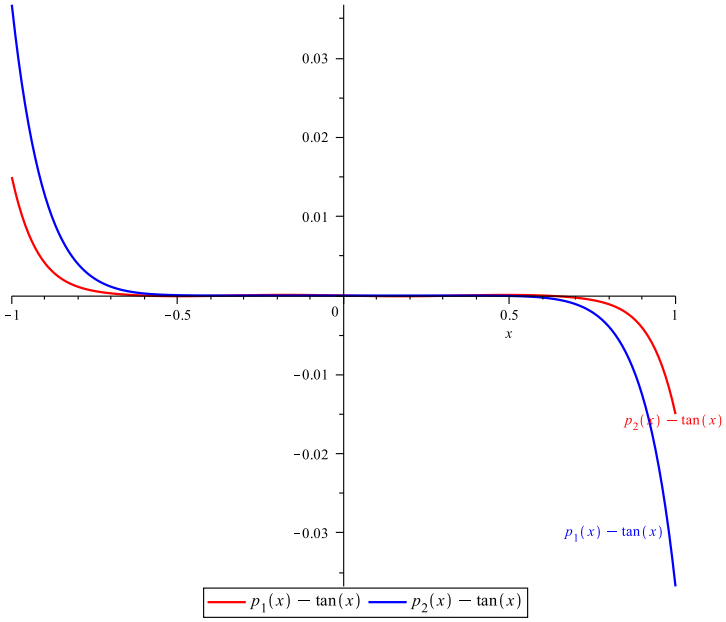
Sommaire

1 Introduction

Ce chapitre est consacré aux polynômes et séries exprimés dans la base des polynômes de Tchebychev lorsque leurs coefficients ne vérifient aucune relation particulière, contrairement au reste de cette thèse où les coefficients vérifieront une récurrence. Dans la base monomiale, beaucoup d'algorithmes rapides existent, notamment pour la multiplication et la division entre polynômes ; une bonne référence est le livre de von zur Gathen et Gerhard [vzGG03]. On peut de la même façon se questionner sur l'existence d'algorithmes de même complexité pour des polynômes exprimés dans la base de Tchebychev. Ce chapitre donne des algorithmes de multiplication et de division en étudiant leur complexité arithmétique. On remarque alors que ces opérations ne sont pas beaucoup plus coûteuses que dans la base monomiale. Giorgi [Gio12] a récemment proposé plusieurs de ces algorithmes. Dans la section 4, je décris un de ces algorithmes qui a la particularité de se ramener à deux produits de polynômes exprimés dans la base monomiale. L'algorithme de division rapide proposé dans la même section est, contrairement à la multiplication, une des contributions de ce chapitre ; le résultat est intéressant car je montre que l'on peut calculer le quotient de la division entre deux polynômes dans la base de Tchebychev pour le même coût que le quotient de deux polynômes dans la base monomiale. Cette section montre donc que les opérations de base que sont la multiplication et la division entre polynômes exprimés dans la base des Tchebychev sont semblables (en terme de complexité) aux mêmes opérations entre des polynômes exprimés dans la base monomiale.

Avant la présentation de ces algorithmes, les sections 2 et 3 montrent comment convertir rapidement des polynômes exprimés dans la base de Tchebychev en des polynômes exprimés dans la base monomiale, et réciproquement. La conversion d'un polynôme de degré n d'une base à l'autre s'effectue en un nombre constant de multiplications entre deux polynômes de degré n dans la base monomiale, c'est-à-dire en $\mathcal{O}(\mathsf{M}(n))$ opérations arithmétiques. De cette façon, on déduit de nombreux algorithmes rapides pour des opérations entre des polynômes exprimés dans la base de Tchebychev. En effet en utilisant la FFT, on ramène beaucoup d'algorithmes s'appliquant sur des polynômes de degré n ou des séries à précision n, à une complexité de $\mathcal{O}(n \log^2 n)$ opérations arithmétiques. Par exemple : le calcul de pgcd [GY79, BGY80], d'approximants de Padé [BGY80, GGWY82], d'évaluation multipoint, d'interpolation [BM74], ou encore de résultant [Sch80]. Une façon d'obtenir des algorithmes rapides pour les opérations précédentes, mais cette fois-ci entre polynômes exprimés dans la base de Tchebychev, est donc de convertir ces polynômes à l'aide d'un algorithme de changement de base rapide (en $\mathcal{O}(n \log n)$ avec la FFT), d'utiliser un algorithme rapide dans la base monomiale, puis de convertir le résultat dans la base de Tchebychev. On obtient de cette façon le même nombre d'opérations asymptotiques pour effectuer les opérations précédentes (pgcd, évaluation, interpolation, etc ...) dans la base de Tchebychev que dans la base monomiale. Par contre, utiliser la conversion rapide pour multiplier ou diviser deux polynômes dans la base de Tchebychev augmenterait fortement la constante de ces opérations dans le $\mathcal{O}$. Les algorithmes spécifiques de multiplication et de division présentés dans la seconde partie effectuent ces opérations sans changement de

FIGURE 3.1 – Économisation de la série $\tan(x)$



base en gardant une complexité proche de celle dans la base monomiale.

Une autre application des conversions est donnée par la méthode « d'économisation de série » [MH03, §3.5]. Le but de cette méthode est d'obtenir une bonne approximation sur $[-1,1]$ d'une fonction par un polynôme dans la base monomiale. Une bonne approximation d'une fonction par un polynôme de degré n est calculée à partir de son développement de Taylor tronqué à l'ordre n. On peut aussi l'obtenir en développant à l'ordre $m > n$ et en soustrayant des polynômes de Tchebychev (multipliés par des constantes) de degré $m, m-1, \ldots, n+1$ à l'approximation. Grâce à cette méthode, on obtient une meilleure approximation.

Exemple 3.1. La fonction $\tan(x)$ est approximée dans un voisinage de $x = 0$ par le polynôme de degré 9 obtenu en tronquant son développement de Taylor à l'ordre 10 :

$$\tan(x) \approx x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + \frac{62}{2835}x^9.$$

Pour l'approximer par un polynôme de degré 7, on peut soit soustraire à ce polynôme le terme $\frac{62}{2835}x^9$:

$$\tan(x) \approx x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 =: p_1(x),$$

soit soustraire le polynôme $\frac{31}{362880}T_9(x)$ et on obtient alors :

$$\tan(x) \approx \frac{40289}{40320}x + \frac{1039}{3024}x^3 + \frac{27}{280}x^5 + \frac{13}{126}x^7 =: p_2(x).$$

On peut voir sur le graphe 1 que la seconde approximation est meilleure que la première sur $[-1,1]$ (au sens de la norme uniforme).

La soustraction du polynôme de Tchebychev de degré n à un polynôme de degré n exprimé dans la base monomiale s'effectue en $\mathcal{O}(n)$ opérations arithmétiques (le $n^{\text{ème}}$ polynôme de Tchebychev possède $n/2$ termes dans la base monomiale). Si on souhaite par exemple soustraire $\mathcal{O}(\sqrt{n})$ polynômes de Tchebychev pour obtenir un polynôme de degré $n-\sqrt{n}$ en les soustrayant un à un, on effectue $\mathcal{O}(n^{3/2})$ opérations arithmétiques. En revanche, En utilisant un algorithme de conversion rapide, on effectue d'abord $\mathcal{O}(\mathsf{M}(n))$ opérations arithmétiques pour écrire le polynôme dans la base de Tchebychev ; puis on peut le tronquer en temps constant pour obtenir le polynôme désiré. L'expression de ce polynôme dans la base monomiale s'effectue alors en $\mathcal{O}(\mathsf{M}(n))$ opérations arithmétiques. En utilisant la FFT pour la multiplication de polynômes, on peut alors « économiser » la série plus rapidement. Mason et Handscomb proposent aussi cette conversion pour « économiser » une série mais sans changement de base rapide [MH03].

Dans tout ce chapitre $\mathbb{K}$ désigne un corps supposé de caractéristique nulle.

Changement de base Avant de donner des algorithmes rapides pour effectuer ce changement de base, définissons plus rigoureusement notre problème. On souhaite donner des algorithmes rapides qui prennent en entrée un entier n et un vecteur $(t_0,\dots,t_{n-1})$ dans $\mathbb{K}^n$ et qui rendent en sortie un vecteur $(c_0,\dots,c_{n-1})$ dans $\mathbb{K}^n$ tels que :

$$\sum_{i=0}^{n-1} c_i x^i = \sum_{i=0}^{n-1} t_i T_i(x).$$

L'application de changement de base dans l'anneau $\mathbb{K}_n[x]$ des polynômes de degré inférieur à n, « base de Tchebychev $\mapsto$ base monomiale » (passer des t_i aux c_i) est $\mathbb{K}$-linéaire ; sa matrice $\mathcal{T}_n$ de taille $n\times n$ dans la base canonique $\{1,x,\dots,x^{n-1}\}$ contient en colonne j les coefficients du polynôme $T_j(x)$ dans la base monomiale ((2.10) page 20) :

$$T_j(x) = \frac{j}{2}\sum_{k=0}^{\lfloor \frac{j}{2}\rfloor}(-1)^k\frac{(j-k-1)!}{k!(j-2k)!}(2x)^{j-2k} \quad \text{pour } j\geqslant 1.$$

Il s'avère plus commode et plus utile, de travailler avec la base modifiée des polynômes de Tchebychev $\frac{1}{2}T_0(x), T_1(x),\dots,T_n(x)$. Autrement dit, on souhaite déduire des t_i les coefficients c_i tels que $\sum{}' t_i T_i(x) = \sum c_i x^i$ (voir notation 2.16 page 23).

Exemple 3.2. Dans cette base, la matrice $\mathcal{T}_8$ s'écrit

$$\mathcal{T}_8 := \begin{bmatrix} 1/2 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -3 & 0 & 5 & 0 & -7 \\ 0 & 0 & 2 & 0 & -8 & 0 & 18 & 0 \\ 0 & 0 & 0 & 4 & 0 & -20 & 0 & 56 \\ 0 & 0 & 0 & 0 & 8 & 0 & -48 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & -112 \\ 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 64 \end{bmatrix}.$$

Le problème de conversion de base entre les bases de polynômes de Tchebychev et monomiale en degré fixé strictement plus petit que n (appelé $\mathsf{CtoM_n}$) se réduit à la multiplication de la matrice $\mathcal{T}_n$ avec le vecteur v_p représentant les coefficients du polynôme p dans la base de Tchebychev. Un algorithme naïf de conversion $\mathsf{CtoM_n}$ effectuant une multiplication matrice avec vecteur, a une complexité arithmétique quadratique en n, car malgré la présence de nombreux zéros et l'aspect triangulaire de $\mathcal{T}_n$, cette matrice contient un nombre quadratique d'éléments non-nuls (chaque polynôme T_j s'écrit avec environ $j/2$ coefficients non nuls dans la base monomiale).

Le problème de conversion de base dans le sens opposé « base monomiale ↦ base de Tchebychev » en degré fixé inférieur à n (que l'on note $\mathsf{MtoC_n}$) se ramène à la multiplication de la matrice inverse de $\mathcal{T}_n$ avec le vecteur représentant le polynôme écrit dans la base monomiale. Les coefficients de l'inverse de $\mathcal{T}_n$ sont donnés par la formule classique [MH03] :

$$x^n = 2^{1-n} {\sum_{k=0}^{\lfloor n/2 \rfloor}}' \binom{n}{k} T_{n-2k}(x),$$

qui est démontrée dans le chapitre 6. L'algorithme naïf de conversion $\mathsf{MtoC_n}$ a donc également une complexité arithmétique quadratique en n pour les mêmes raisons que l'algorithme naïf résolvant $\mathsf{CtoM_n}$.

Pour accélérer le calcul du produit de la matrice $\mathcal{T}_n$ avec le vecteur v_p, on va dans les sections 2 et 3 factoriser $\mathcal{T}_n$ comme produit de matrices *simples*, en ce sens que chacune d'entre elles peut être multipliée par un vecteur en seulement $\mathcal{O}(\mathsf{M}(n))$ opérations arithmétiques.

On présente deux algorithmes basés sur cette factorisation. Cette idée de factorisation est commune aux deux algorithmes mais l'approche et les factorisations sont différentes. Même s'ils sont fortement adaptés de travaux de Pan [Pan98] pour le premier et de Bostan, Salvy et Schost [BSS08] pour le second, c'est la première fois que des algorithmes rapides pour les problèmes $\mathsf{CtoM_n}$ et $\mathsf{MtoC_n}$ sont décrits explicitement. En effet l'algorithme de Pan était originalement conçu pour l'évaluation rapide de polynôme dans la base monomiale en

α tel que $\mathsf{M(n)} = \tilde{\mathcal{O}}(n^\alpha)$	2 (algorithme naïf)	$\log_2 3$ (Karatsuba)	1 (FFT)
algorithme 3.1 (Pan)	$10\mathsf{M}(n) + \mathcal{O}(n)$	$8\mathsf{M}(n) + \mathcal{O}(n)$	$6\mathsf{M}(n) + \mathcal{O}(n)$
algorithme 3.5 (Bostan *et al.*)	$4\mathsf{M}(n) + \mathcal{O}(n)$	$5\mathsf{M}(n) + \mathcal{O}(n)$	$6\mathsf{M}(n) + \mathcal{O}(n)$

FIGURE 3.2 – Comparaison des complexités des algorithmes pour la conversion $\mathsf{CtoM_n}$ (en nombre d'opérations arithmétiques)

α tel que $\mathsf{M(n)} = \tilde{\mathcal{O}}(\mathsf{n}^\alpha)$	2 (algorithme naïf)	$\log_2 3$ (Karatsuba)	1 (FFT)
algorithme 3.1 (Pan)	$10\mathsf{M}(n) + \mathcal{O}(n)$	$8\mathsf{M}(n) + \mathcal{O}(n)$	$6\mathsf{M}(n) + \mathcal{O}(n)$
algorithme 3.12 (Bostan *et al.*)	$9/4\mathsf{M}(n) + \mathcal{O}(n)$	$3,2\mathsf{M}(n) + \mathcal{O}(n)$	$5\mathsf{M}(n) + \mathcal{O}(n)$

FIGURE 3.3 – Comparaison des complexités des algorithmes pour la conversion $\mathsf{MtoC_n}$ (en nombre d'opérations arithmétiques)

des points de Tchebychev (l'ensemble des points de Tchebychev de degré n est l'ensemble des racines du $n^{\text{ème}}$ polynôme de Tchebychev). Notre apport est la remarque qu'effectuer ces évaluations revient à calculer le changement de base du même polynôme exprimé dans la base monomiale en un polynôme exprimé dans la base des polynômes de Tchebychev.

Le second algorithme est obtenu en spécialisant une méthode générale, due à Bostan, Salvy et Schost [BSS08], qui permet de déduire des algorithmes de conversion de polynômes entre différentes bases de polynômes orthogonaux classiques. J'explicite le cas particulier du changement de base entre la base de Tchebychev et la base monomiale. La déduction de cet algorithme est loin d'être triviale et inintéressante.

On remarquera que les idées du premier algorithme sont plus simples que celles du second. D'un autre côté après une étude approfondie des complexités de ces algorithmes, je montre que, même si tous les deux ont une complexité quasi-linéaire, le second algorithme est plus rapide que le premier (au sens de la complexité arithmétique) d'un facteur constant. Les tableaux 3.2 et 3.3 résument cette comparaison en fonction de l'algorithme de multiplication polynomiale utilisé.

2 Algorithme de Pan

2.1 Point de départ pour le problème $\mathsf{CtoM_n}$

L'idée de Pan [Pan98] est d'utiliser l'égalité (voir (2.3) page 18)

$$T_n\left(\frac{\omega + \omega^{-1}}{2}\right) = \frac{\omega^n + \omega^{-n}}{2}, \quad \forall \omega \in \mathbb{K}, \tag{3.1}$$

Entrée: $t_0, \dots, t_{n-1}$ des éléments de $\mathbb{K}$ tels que $p(x) := \sum_{i=0}^{n-1} {}'t_i T_i(x)$
Sortie: $c_0, \dots, c_{n-1}$ des éléments de $\mathbb{K}$ tels que $p(x) = \sum_{i=0}^{n-1} c_i x^i$
1: Calculer $p_0(x) := \frac{1}{2} \sum_{i=0}^{2n-2} t_{|n-1-i|} x^i$
2: Calculer $p_1(x) := (1-x)^{2n-2} p_0\left(\frac{1+x}{1-x}\right)$
3: Calculer $p_2(x) := p_1\left(\sqrt{x}\right)$
4: Calculer $p_3(x) := (x+1)^{n-1} p_2\left(\frac{x-1}{x+1}\right)$
5: **renvoyer** $2^{-n+1} p_3(x)$

Algorithme 3.1: Algorithme de conversion de Pan pour résoudre $\mathsf{CtoM_n}$

dont on déduit

$$p\left(\frac{\omega+\omega^{-1}}{2}\right) = \sum_{i=0}^{n-1} {}'t_i T_i\left(\frac{w+w^{-1}}{2}\right) = \frac{1}{2} \sum_{i=-n+1}^{n-1} {}'t_{|i|} \omega^i.$$

Les changements de variables

$$z = \frac{1-\omega}{1+\omega}, \quad y = z^2, \quad x = \frac{1+y}{1-y}, \tag{3.2}$$

mènent à la suite d'égalités :

$$\frac{\omega+\omega^{-1}}{2} = \frac{1}{2}\left(\frac{1-z}{1+z} + \frac{1+z}{1-z}\right) = \frac{1+z^2}{1-z^2} = \frac{1+y}{1-y} = x. \tag{3.3}$$

De ces formules, on déduit immédiatement l'algorithme 3.1 pour la conversion de base.

Si l'application $\mathbb{K}$-linéaire qui à un polynôme $p(x)$ de degré n associe le polynôme $(1-x)^n p\left(\frac{1+x}{1-x}\right)$ est représentée par une matrice *simple*, c'est-à-dire se calcule en un temps quasi-linéaire, chaque étape de l'algorithme 3.1 représente les éléments de la factorisation de $\mathcal{T}_n$ en matrices *simples*. En effet, dans cet algorithme, seules les étapes 2 et 4 ne s'exécutent pas en un nombre linéaire d'opérations.

2.2 Calcul rapide de $(cx+d)^{n-1} p\left(\frac{ax+b}{cx+d}\right)$

Dans le même article, Pan donne un algorithme pour effectuer les étapes 2 et 4 de l'algorithme 3.1 rapidement. Le point de départ est l'algorithme de décalage 3.2 dû à Aho, Steiglitz et Ullman [ASU75] qui ramène le calcul d'un polynôme décalé $p(x+1)$ à partir des coefficients de $p(x)$ à une multiplication entre deux polynômes plus un nombre d'opérations linéaires en fonction du degré. Je détaille d'abord cet algorithme.

Pour montrer la correction de l'algorithme 3.2, les auteurs utilisent l'égalité suivante :

$$\sum_{i=0}^{n-1} p_i (x+1)^i = \sum_{i=0}^{n-1} p_i \sum_{j=0}^{i} \binom{i}{j} x^j = \sum_{j=0}^{n-1} \frac{x^j}{j!} \sum_{i=j}^{n-1} p_i i! \frac{1}{(i-j)!}. \tag{3.4}$$

Entrée: $p(x) = \sum_{i=0}^{n-1} p_i x^i$, un polynôme dans $\mathbb{K}[x]$
Sortie: $p(x+1) = \sum_{i=0}^{n-1} q_i x^i$, où les q_i sont des coefficients de $\mathbb{K}[x]$.
1: Calculer $a = \sum_{i=0}^{n-1} p_i i! x^i$
2: Calculer $b = \sum_{i=0}^{n-1} \frac{1}{(n-i-1)!} x^i$
3: Calculer $ab = \sum_{i=0}^{2n-2} c_i x^i$
4: **renvoyer** $\sum_{i=0}^{n-1} q_i x^i = \sum_{i=0}^{n-1} \frac{c_{n-1+i}}{i!} x^i$

Algorithme 3.2: Algorithme de décalage

En définissant les coefficients a_i et b_i tels que :

$$a_i = \begin{cases} p_i i! & \text{si} \quad i \in [\![0, n-1]\!], \\ 0 & \text{sinon,} \end{cases}, \text{ et} \tag{3.5}$$

$$b_i = \begin{cases} \frac{1}{i!} & \text{si} \quad i \in [\![0, n-1]\!], \\ 0 & \text{sinon,} \end{cases} \tag{3.6}$$

on ramène, grâce à (3.4), le calcul du polynôme décalé $p(x+1)$ au produit des deux polynômes :

$$\sum_{i=0}^{n-1} a_i x^i \cdot \sum_{i=0}^{n-1} b_{n-1-i} x^i = \sum_{j=0}^{2n-2} \sum_{i=0}^{j} a_i b_{n-1+i-j} x^j.$$

En effet, en ne sélectionnant que les n coefficients associés aux monômes de plus hauts degrés, et en utilisant les propriétés $a_i = 0$ pour $i \geqslant n-1$ et $b_{i-j} = 0$ pour $i < j$, on a :

$$\sum_{j=n-1}^{2n-2} \sum_{i=0}^{j} a_i b_{n-1+i-j} x^{j-n+1} = \sum_{j=0}^{n-1} \sum_{i=0}^{j+n-1} a_i b_{i-j} x^j = \sum_{j=0}^{n-1} \sum_{i=j}^{n-1} a_i b_{i-j} x^j = \sum_{j=0}^{n-1} \sum_{i=j}^{n-1} \frac{p_i i!}{(i-j)!} x^j.$$

Il suffit de diviser par $j!$ le coefficient associé au monôme de degré j pour en déduire l'algorithme 3.2.

Pan déduit l'algorithme 3.3 de l'algorithme 3.2, ainsi que le lemme 3.3 qui en découle.

Lemme 3.3 ([Pan98]). *Soit p un polynôme de degré au plus $n-1$ à coefficients dans $\mathbb{K}$. L'algorithme 3.3 calcule les coefficients du polynôme $(cx+d)^{n-1} p\left(\frac{ax+b}{cx+d}\right)$ en $2\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques.*

Démonstration. La fraction rationnelle $\frac{ax+b}{cx+d}$ se réécrit comme $\frac{a}{c}\left(1 + \frac{bc-da}{da\left(\frac{c}{d}x+1\right)}\right)$, ce qui permet de déduire la correction de l'algorithme 3.3. On observe tout de suite avec cette écriture les deux décalages de polynômes nécessaires pour effectuer cet algorithme. Afin d'estimer la complexité, il suffit de remarquer que les seules opérations coûteuses sont les décalages aux lignes 2 et 4. En effet les opérations effectuées aux autres lignes sont des homothéties et le calcul d'un polynôme réciproque, opérations qui s'effectuent en $\mathcal{O}(n)$ opérations arithmétiques. Les deux opérations de décalage s'exécutent en $\mathsf{M}(n)$ opérations arithmétiques chacune, d'où le coût final de cet algorithme. □

Entrée: $p_0, \dots, p_{n-1}$ et a, b, c, d des éléments de $\mathbb{K}$ tels que $p(x) = \sum_{i=0}^{n-1} p_i x^i$ et $a,\ c,\ d,\ bc-da$ soient non nuls
Sortie: $q_0, \dots, q_{n-1}$ des éléments de $\mathbb{K}$ tels que $q(x) = (cx+d)^{n-1} p\left(\frac{ax+b}{cx+d}\right) = \sum_{i=0}^{n-1} q_i x^i$
1: Calculer $p_1(x) = p\left(\frac{a}{c}x\right)$
2: Calculer $p_2(x) = p_1(x+1)$
3: Calculer $p_3(x) = x^{n-1} p_2\left(\frac{1}{x}\right)$
4: Calculer $p_4(x) = p_3\left(\frac{da}{bc-da}(x+1)\right)$
5: **renvoyer** $\left(\frac{bc-da}{a}\right)^{n-1} p_4\left(\frac{c}{d}x\right)$

Algorithme 3.3: Calcul de $(cx+d)^{n-1} p\left(\frac{ax+b}{cx+d}\right)$

2.3 Complexité de l'algorithme 3.1

L'algorithme 3.3 permet d'effectuer les changements de variables entre ω et z ou y et x de la formule (3.3) page 34 de manière efficace. On peut alors en déduire le théorème 3.4.

Théorème 3.4. *L'algorithme 3.1 calcule la conversion d'un polynôme de degré strictement plus petit que n exprimé dans la base de Tchebychev en un polynôme dans la base monomiale en $2\mathsf{M}(n) + 2\mathsf{M}(2n) + \mathcal{O}(n)$ opérations arithmétiques.*

Remarque 3.5. Si $\mathsf{M}(n) = \mathcal{O}(n \log n)$, la complexité se ramène à $6\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques.

Démonstration. On montre dans un premier temps la correction de cet algorithme. Le polynôme $p_0(x)$ à la ligne 1 de l'algorithme est égal au polynôme $x^{n-1} p\left(\frac{x+x^{-1}}{2}\right)$, la suite de la validité de l'algorithme est vérifiée par l'équation (3.3) page 34. Lors des étapes 2 et 4, des multiplications par des polynômes de degré $n-1$ sont effectuées, ces multiplications servent à la fois à gérer le multiple x^{n-1} en trop de l'équation 1 et à continuer à travailler avec des polynômes et non des fractions rationnelles.

Il est important de noter que l'équation (3.3) permet aussi de montrer que p_1 est un polynôme en x^2, donc que p_2 est bien un polynôme en x.

Il reste à analyser la complexité de cet algorithme. Les calculs effectués aux lignes 1 et 3 s'exécutent en un nombre linéaire d'opérations arithmétiques. En effet, il ne s'agit dans les deux cas que de réécriture avec en plus une division par deux de chaque coefficient à la ligne 1. Les complexités des lignes 2 et 4 se déduisent du lemme 3.3 en faisant attention aux tailles des objets : la première substitution se fait avec un polynôme de degré $2n-2$ et la seconde avec un polynôme de degré $n-1$. □

2.4 Algorithme de Pan pour résoudre $\mathsf{MtoC_n}$

Les fractions rationnelles $\frac{x-1}{x+1}$ et $-\frac{x+1}{x-1}$ sont inverses l'une de l'autre pour la composition, on en déduit l'algorithme 3.4 de conversion inverse $\mathsf{MtoC_n}$. La complexité arithmétique de l'algorithme inverse est donc aussi $2\mathsf{M}(n) + 2\mathsf{M}(2n) + \mathcal{O}(n)$ opérations arithmétiques.

Entrée: $c_0, \ldots, c_{n-1}$ des éléments de $\mathbb{K}$ tels que $p(x) = \sum_{i=0}^{n-1} c_i x^i$
Sortie: $t_0, \ldots, t_{n-1}$ des éléments de $\mathbb{K}$ tels que $p(x) := \sum_{i=0}^{n-1} {}'t_i T_i(x)$
1: $p_0 := \sum_{i=0}^{n-1} c_i x^i$
2: $p_1 := (1-x)^{n-1} p\left(\frac{1+x}{1-x}\right)$
3: $p_2 := p_1(x^2)$
4: $p_3 := \left(\frac{1+x}{2}\right)^{2n-2} p\left(\frac{1-x}{1+x}\right)$
5: **renvoyer** $t_i := 2\mathrm{coeff}\,(p_3, x, i+n-1)$

Algorithme 3.4: Algorithme de conversion de Pan $\mathsf{MtoC_n}$

3 Algorithme de Bostan-Salvy-Schost

Bostan, Salvy et Schost [BSS08] ont donné une méthode permettant de calculer la conversion d'un polynôme de la base monomiale vers des bases de polynômes orthogonaux *classiques* avec une complexité quasi-linéaire. Les mêmes auteurs ont donné par la suite [BSS10] un algorithme plus général qui permet de calculer la conversion avec des polynômes orthogonaux *quelconques* en un nombre d'opérations quasi-linéaire, mais pour les polynômes orthogonaux classiques, cet algorithme perd un $\log n$ par rapport à l'algorithme déduit de [BSS08].

L'idée de Bostan *et alii* est aussi de trouver une factorisation de la matrice $\mathcal{T}_n$ en matrices *simples*. Une telle factorisation n'est pas simple à obtenir directement sur $\mathcal{T}_n$, mais une technique qui s'est avérée fructueuse à plusieurs reprises dans la littérature du calcul formel est de regarder la matrice transposée. Cela est suffisant, car si $\mathcal{T}_n^t$ se factorise en produit de matrices *simples*, alors $\mathcal{T}_n$ elle-même se factorise en produit de transposées de matrices *simples*. Or, le principe de transposition de Tellegen [Kal93, Sho94] affirme que tout algorithme qui permet d'effectuer le produit $S \cdot v$ d'une matrice par un vecteur en $\mathcal{O}(\mathsf{M}(n))$ peut être transformé en un algorithme de complexité $\mathcal{O}(\mathsf{M}(n))$ pour le produit $S^t \cdot w$. Dans la suite, on nommera *algorithme transposé*, l'algorithme obtenu par transposition.

3.1 Conversion transposée et composition

On observe que $\mathsf{CtoM_n}$ appliqué au vecteur $\ell = (1, \ldots, t^{n-1})$ donne les coefficients (en x) du polynôme $\sum_{k=0}^{n-1} {}'T_k(x) t^k$. En effet, si on nomme $T_{i,j}$ les coefficients tels que $T_i(x) = \sum_{i=0}^{i} T_{i,j} x^j$, l'application $\mathsf{CtoM_n}(\ell)$ donne le vecteur

$$\left(\sum_{i=0}^{n-1} T_{i,0} t^i, \ldots, \sum_{i=0}^{n-1} T_{i,n-1} t^i \right).$$

Il vient tout de suite que l'image du vecteur $\widetilde{\ell} = (1, \ldots, x^{n-1})$ par la conversion transposée $\mathsf{CtoM_n^t}$ est le vecteur des coefficients en t du même polynôme. L'application $\mathsf{CtoM_n^t}(\widetilde{\ell})$

donne

$$\left(\sum_{j=0}^{n-1} T_{0,j}x^j, \ldots, \sum_{j=0}^{n-1} T_{n-1,j}x^j\right).$$

Définissons la famille de fonctions C_i par :

$$\sum_{k\geqslant 0}{}' T_k(x)t^k = \sum_{i\geqslant 0}{}' C_i(t)x^i. \tag{3.7}$$

On obtient cette famille (unique) en développant par rapport à x la série génératrice des polynômes de Tchebychev. Ainsi, par unicité des C_i, la conversion transposée CtoM_n^t est l'application linéaire :

$$(f_0, \ldots, f_{n-1}) \in \mathbb{K}^n \mapsto \frac{f_0}{2}C_0(t) + f_1C_1(t)\cdots + f_{n-1}C_{n-1}(t) \mod t^n.$$

L'idée de base de la méthode présentée dans [BSS10] est que si la série génératrice des C_i est de la forme :

$$\sum_{i\geqslant 0}{}' C_i(t)x^i = \frac{v(t)}{1 - x\cdot h(t)}, \tag{3.8}$$

alors

$$\sum_i{}' C_i(t)f_i \mod t^n = v(t)\cdot F(h(t)) \mod t^n, \quad \text{avec } F = \sum_i f_i x^i.$$

Cette égalité s'obtient en développant par rapport à x les membres de l'équation (3.8).

Partant de la série génératrice des polynômes de Tchebychev (2.15) page 21 et de la définition des C_i (3.7), on observe que

$$v(t) = \frac{1}{2}\frac{1-t^2}{1+t^2} \quad \text{et} \quad h(t) = \frac{2t}{1+t^2},$$

sont bien définis pour l'équation (3.8). On obtient alors la factorisation

$$\mathsf{CtoM}_n^t = mul(\cdot, v) \circ \mathsf{eval}(\cdot, h), \tag{3.9}$$

où

$$\mathsf{eval}(\cdot, h) : (t_0, \ldots, t_n) \mapsto \sum_{i\geqslant 0}^{n-1} t_i h^i(x) \mod x^n \quad \text{et}$$

$$mul(\cdot, v) : (t_0, \ldots, t_{n-1}) \mapsto v(x)\sum_{i=0}^{n-1} t_i x^i \mod x^n.$$

L'algorithme calculant la conversion se déduit en transposant chaque opération. On obtient alors l'algorithme 3.5.

La suite de cette section montre comment effectuer chaque étape efficacement.

Entrée: $p := (p_0, p_1, \ldots, p_n)$ tels que $p_i \in \mathbb{K}$
Sortie: $b := (b_0, b_1, \ldots, b_n)$ tels que $\sum' p_i T_i(x) = \sum b_i x^i$
1: $a := \mathsf{mul}^t(p, v)$
2: **renvoyer** $b := \mathsf{eval}^t(a, h)$

Algorithme 3.5: $\mathsf{CtoM_n}$

3.2 Algorithmes rapides pour la composition avec $\frac{2t}{1+t^2}$

Avant de donner les algorithmes transposés, il est nécessaire de présenter les algorithmes directs.

Toujours dans l'article [BSS08], une méthode permettant de calculer rapidement certaines compositions de polynômes par des séries est expliquée. La composition d'un polynôme quelconque par la série $h = \frac{2t}{1+t^2}$ en est un cas particulier.

Le principe fondamental de cette méthode est d'isoler la variable x dans l'écriture de la série génératrice h. Pour h par exemple, l'isolation de la variable se ramène à l'écriture suivante :

$$h(x) = \frac{2x}{1+x^2} = \sqrt{1 - \left(\frac{2}{1+x^2} - 1\right)^2}. \tag{3.10}$$

Pour l'algorithme 3.3, Pan avait déjà utilisé cette idée d'isolation de la variable quand il a écrit la fraction $\frac{ax+b}{cx+d}$ comme $\frac{a}{c}\left(1 + \frac{bc-da}{da(\frac{c}{d}x+1)}\right)$. En fait Bostan et coauteurs proposent une généralisation de l'algorithme de Pan qui était lui même une généralisation de l'algorithme de décalage.

Par l'équation (3.10), le calcul de l'application $\mathsf{eval}(\cdot, h)$ se ramène aux calculs d'applications *simples* de compositions par $(ax+b)$, par $\frac{1}{x}$, par x^2 ou encore par $\sqrt{x}$. L'algorithme 3.6 effectue ce calcul en n'utilisant que ces compositions.

Proposition 3.6. *Soit p un polynôme de degré $n-1$, l'algorithme 3.6 permet de calculer $p\left(\frac{2x}{1+x^2}\right)$ en $4\mathsf{M}(n) + 4\mathsf{M}\left(\frac{n}{2}\right) + \mathcal{O}(n)$ opérations arithmétiques.*

Remarque 3.7. La fonction $\mathsf{M}(n)$ est au mieux linéaire : $2\mathsf{M}\left(\frac{n}{2}\right) \leqslant \mathsf{M}(n)$. La complexité est donc majorée par $6\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques.

Pour démontrer cette proposition, on a besoin du lemme suivant, qui affirme que la multiplication d'une série par une fraction rationnelle peut s'effectuer en temps linéaire. Même si ce résultat paraît simple, je ne l'ai pas trouvé dans la littérature.

Lemme 3.8. *Si $f \in \mathbb{K}(x)$ a un numérateur et un dénominateur de degré $\mathcal{O}(1)$ et n'a pas de pôle en 0, alors pour toute série formelle g à coefficients dans $\mathbb{K}$, la série tronquée $fg \bmod x^n$ peut se calculer en $\mathcal{O}(n)$ opérations arithmétiques.*

Démonstration. On définit d'abord les polynômes a et b de degré constant tels que $f = \frac{a}{b}$.

En utilisant un algorithme naïf, c'est-à-dire en multipliant terme à terme, le produit tronqué $g(x)a(x) \mod x^n$ se réalise en $\mathcal{O}(n)$ opérations arithmétiques, cette complexité s'explique par le degré constant de a.

Entrée: $p(x) \in \mathbb{K}[x]$ avec $\deg(p, x) < n$,
Sortie: $p\left(\frac{2x}{1+x^2}\right) \mod x^n$

1: Calculer $p_{1,0}$ et $p_{1,1}$ tel que $p(x) = p_{1,0}(x^2) + xp_{1,1}(x^2)$
2: **pour** $i \in \{0, 1\}$ **faire**
3: $\quad d_i := \deg(p_{1,i}(x), x)$
4: $\quad p_{2,i}(x) := p_{1,i}(1 - x)$
5: $\quad p_{3,i}(x) := p_{2,i}(x^2)$
6: $\quad p_{4,i}(x) := p_{3,i}(-2x + 1)$
7: $\quad p_{5,i}(x) := \mathrm{p}_{4,i}\left(\frac{1}{x}\right) x^{2d_i}$
8: $\quad p_{6,i}(x) := p_{5,i}(1 + x)$
9: $\quad p_{7,i}(x) := (1 + x)^{-2d_i} p_{6,i}(x) \mod x^{d_i}$
10: $\quad p_{8,i}(x) := p_{7,i}(x^2)$
11: **fin pour**
12: **renvoyer** $p_{8,0}(x) + \frac{2x}{1+x^2} p_{8,1}(x) \mod x^n$

Algorithme 3.6: $\mathsf{eval}\left(\cdot, \frac{2t}{1+t^2}\right)$

Comme $h = fg$, alors $bh = ag$. L'extraction du coefficient x^i donne donc une récurrence linéaire inhomogène d'ordre le degré de b. En effet, en notant h_i et b_i les coefficients de h et b. Le produit hb donne

$$hb = \sum_{i}^{n-1} \sum_{r=i-\deg(b)}^{i} h_r b_{i-r} x^i.$$

À partir de cette récurrence, le calcul des h_i s'effectue en $\mathcal{O}(n)$ opérations arithmétiques. □

Démonstration de la proposition 3.6. La correction de l'algorithme se déduit de l'écriture dans l'équation (3.10) de la fonction h. En effet, les compositions successives de cet algorithme assurent que le polynôme retourné est bien

$$p\left(\sqrt{1 - \left(\frac{2}{1 + x^2} - 1\right)^2}\right) \mod x^n = p\left(\frac{2x}{1 + x^2}\right) \mod x^n.$$

Il reste à donner le coût de chaque opération. Beaucoup de ces opérations se calculent en temps linéaire. C'est le cas des étapes 1, 5, 7 et 10 qui ne sont que des réécritures directes des coefficients. La multiplication effectuée à l'étape 12 s'effectue aussi en un temps linéaire. Cette propriété est donnée par le lemme 3.8. Les opérations non élémentaires sont les décalages effectués aux lignes 4, 6 et 8 et la multiplication à la ligne 4. L'algorithme 3.2 permet de calculer le polynôme décalé d'un polynôme de degré plus petit que n en $\mathsf{M}(n)$ opérations arithmétiques. À la ligne 4, ces décalages sont effectués sur des polynômes de degré au plus $\frac{n}{2}$ et les autres étapes décalent des polynômes de degré au plus n, on obtient donc une complexité de $4\mathsf{M}(n) + 2\mathsf{M}\left(\frac{n}{2}\right)$ opérations arithmétiques pour effectuer l'ensemble de ces opérations.

Entrée: $p(x) \in \mathbb{K}[x]$ de degré au plus $n-1$
Sortie: $\mathsf{eval}^t(p,h)$
1: Calculer $p_{7,0}$ et $p^*_{7,1}$ tels que $p(x) = p_{7,0}(x^2) + xp^*_{7,1}(x^2)$
2: $d_0 := \text{degrée}(p_{7,0}, x)$ et $d_1 := \text{degrée}(p^*_{7,1}, x)$
3: $p_{7,1}(x) := \mathsf{mul}^t(p^*_{7,1}, \frac{2}{(1+x)}, d_1)$
4: **pour** $i \in \{0,1\}$ **faire**
5: $\quad p_{6,i}(x) := \mathsf{mul}^t\left(p_{7,i}, (1+x)^{2d_i}\right)$
6: $\quad p_{5,i}(x) := \mathsf{shift}^t(p_{6,i}(x) + \sum_{i=d_i+1}^{n-1} 0x^i)$
7: $\quad p_{4,i}(x) := x^{n-1}p_{5,i}\left(\frac{1}{x}\right)$
8: $\quad p_{3,i}(x) := \mathsf{shift}^t(p_{4,i}(-2x))$
9: $\quad p_{2,i}(x) := \frac{1}{2}\left(p_{3,i}(x) + p_{3,i}(-x)\right)$
10: $\quad p_{1,i}(x^2) := \mathsf{shift}^t(p_{2,i}(-x))$
11: **fin pour**
12: **renvoyer** $p_{1,0}(x^2) + xp_{1,1}(x^2)$

Algorithme 3.7: $\mathsf{eval}^t\left(\cdot, \frac{2t}{1+t^2}\right)$

Il ne reste plus qu'à expliciter le coût des opérations réalisées à la ligne 9. Les coefficients du développement en série de la fonction $\left(1+x^2\right)^{-d}$ vérifient la récurrence :

$$(2d+n)u_n + (n+2)u_{n+2} = 0,$$

avec les conditions initiales $u_0 = 1$ et $u_1 = 0$. Calculer les $n/2$ premiers coefficients s'effectue en $\mathcal{O}(n)$ opérations arithmétiques en déroulant cette récurrence. Le coût de l'étape 9 n'est donc que le coût de la multiplication entre deux polynômes de degré aux plus $\frac{n}{2}$. On réalise deux fois cette étape, la complexité est donc de $2\mathsf{M}\left(\frac{n}{2}\right)$ opérations.

On retrouve alors la complexité de $4\mathsf{M}(n) + 4\mathsf{M}(\frac{n}{2})$ opérations arithmétiques annoncée. □

3.3 Transposition de l'algorithme d'évaluation

Par le principe de transposition de Tellegen, on déduit immédiatement de 3.6 l'algorithme 3.7 permettant d'effectuer l'application transposée. Cet algorithme s'obtient en transposant chaque opération.

Avant d'énoncer la complexité de cet algorithme, une présentation des algorithmes mul^t et shift^t doit être effectuée.

Application mul^t Une variante de l'application transposée de mul a déjà été étudiée par Hanrot, Quercia et Zimmermann dans [HQZ04]. Dans cet article les auteurs donnent un algorithme rapide pour le produit médian entre un polynôme de degré n et un polynôme de degré $2n$. Notre application est plus simple. En effet, l'application mul est un produit de deux

Entrée: $p \in \mathbb{K}[x]$ de degré au plus $n-1$
Sortie: $\mathsf{mul}^t(p, f)$
$a := x^{n-1} p\left(\frac{1}{x}\right) f \mod x^n$
renvoyer $x^{n-1} a\left(\frac{1}{x}\right)$

Algorithme 3.8: $\mathsf{mul}^t(\cdot, f)$

Entrée: $p = \sum_{i=0}^{n-1} p_i x^i$ avec $p_i \in \mathbb{K}$
Sortie: $\mathsf{shift}^t(p, x)$
1: Calculer $a(x) = \sum_{i=0}^{n-1} \frac{p_i}{i!} x^i$
2: Calculer $b(x) = a(x) \sum_{i=0}^{n-1} \frac{1}{i!} x^i \mod x^n$
3: **renvoyer** $\sum_{i=0}^{n-1} b_i i! x^i$

Algorithme 3.9: $\mathsf{shift}^t(\cdot)$

polynômes de degré n, tronqué à l'ordre n. Contrairement à la matrice de multiplication de polynômes qui est rectangulaire, la matrice de l'application mul est carrée.

Notre variante du produit médian est décrite dans l'algorithme 3.8, cet algorithme s'obtient en inversant l'application linéaire de multiplication tronquée d'un polynôme p avec une série formelle.

En remarquant que la seule opération coûteuse de cet algorithme est la multiplication entre deux polynômes, le calcul du réciproque d'un polynôme s'effectuant en temps constant (aucune addition ou multiplication n'est effectuée) ; on déduit que l'exécution de cet algorithme s'effectue en $\mathsf{M}(n)$ opérations arithmétiques.

Remarque 3.9. Lorsque la série à multiplier est particulière, on peut améliorer la complexité de l'algorithme 3.8. Par exemple si la série est une fraction rationnelle comme v, le lemme 3.8 entraine que l'application $\mathsf{mul}^t(\cdot, v)$ s'exécute en $\mathcal{O}(n)$ opérations arithmétiques.

Application shift^t Bostan et Schost [BS05] ont donné une interprétation de la transposée de l'application de décalage ; il s'agit de l'application qui calcule l'évaluation de Newton d'un polynôme (évaluer un polynôme dans la base de Newton) en une suite de points suivant une progression arithmétique. Ici nous nous contentons d'exposer cet algorithme par transposition directe de l'algorithme 3.2 sans montrer cette interprétation qui n'est pas essentielle pour la suite. On obtient alors l'algorithme 3.9.

Pour montrer la validité de l'algorithme 3.9, il suffit de vérifier que chaque opération est bien la transposée de l'opération correspondante dans l'algorithme 3.2 ; ce qui se vérifie rapidement.

La complexité de cet algorithme est encore ici $\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques, la seule opération coûteuse étant la multiplication à la ligne 2. Comme pour l'algorithme 3.2, les autres étapes consistent à multiplier ou diviser chaque coefficient par un élément de $\mathbb{K}$, ces étapes se réalisent en $\mathcal{O}(n)$ opérations arithmétiques.

Application eval^t On a décrit dans les paragraphes précédents les deux applications de l'algorithme 3.6 les plus difficiles à transposer. En effet la matrice qui associe au polynôme $p(x)$ le polynôme réciproque est anti-diagonale, donc autoadjointe. La matrice transposée de la matrice qui associe au polynôme $p(x)$ le polynôme $p(x^2)$ est la matrice qui associe à ce même polynôme, le polynôme $\frac{p(\sqrt{x})+p(-\sqrt{x})}{2}$. On déduit l'algorithme 3.7 ainsi que la proposition 3.10 qui en découle de ces règles de transpositions.

Proposition 3.10. *L'algorithme 3.7 page 41 calcule l'image d'un vecteur de taille n par la transposée de l'application $\mathsf{eval}(\cdot, h)$ en $8\mathsf{M}(\frac{n}{2}) + 2\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques.*

Démonstration. La correction de cet algorithme se déduit des propriétés de transposition énoncées plus haut.

L'application de la ligne 6 est la transposée de l'application de la ligne 8 de l'algorithme 3.6 qui calcule le décalage d'un polynôme de degré n. Pour obtenir un polynôme de degré n, on complète le polynôme $p_{6,i}$ de degré d_i par $n-d_i$ zéros. La matrice de complétion est la transposée de la matrice qui tronque une série d'ordre n à l'ordre d_i ; ce qui est l'action effectuée à la ligne 9 de l'algorithme 3.6. Comme on a complété ce polynôme avec des zéros, la multiplication effectuée lors de l'algorithme 3.9 peut se transformer en deux multiplications de polynômes de degré deux fois plus petit ; $a(x)\sum_{i=0}^{\frac{n}{2}} \frac{1}{i!}x^i \mod x^n$ et $x^{\frac{n}{2}}a(x)\sum_{i=0}^{\frac{n}{2}-1} \frac{1}{(i+\frac{n}{1}+1)!}x^{\frac{n}{2}+1}$. On ramène alors la complexité de ce calcul à $2\mathsf{M}(\frac{n}{2}) + \mathcal{O}(n)$ opérations arithmétiques au lieu des $\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques habituelles.

L'algorithme 3.9 est alors appelé six fois, dont deux fois avec des polynômes de degré n, deux fois avec des polynômes de degré $\frac{n}{2}$ et deux fois à la ligne 6 avec des polynômes de degré n mais avec un coût de $2\mathsf{M}(\frac{n}{2}) + \mathcal{O}(n)$ opérations arithmétiques. L'algorithme de multiplication 3.8 est appelé avec deux fois des polynômes de degré $\frac{n}{2}$. Cet algorithme est aussi appelé lors de l'étape 3 mais la fonction f est alors une fraction dont les numérateurs et dénominateurs ont des degrés constants. La multiplication effectuée dans l'algorithme 3.8 se fait en temps linéaire selon la remarque 3.9. On retrouve donc bien la complexité annoncée de $8\mathsf{M}(\frac{n}{2}) + 2\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques. □

Retour à l'algorithme 3.5 On déduit alors immédiatement la preuve du théorème 3.11.

Théorème 3.11. *Soit $p = (p_0, p_1, \ldots, p_n)$, l'algorithme 3.5 page 39 calcule la conversion du polynôme $p_0T_0(x) + \cdots + p_nT_n$ dans la base monomiale en $2\mathsf{M}(n) + 8\mathsf{M}\left(\frac{n}{2}\right) + \mathcal{O}(n)$ opérations arithmétiques.*

Remarque 3.12. Si $\mathsf{M}(n) = \tilde{\mathcal{O}}(n^\alpha)$, l'algorithme de Pan 3.1 effectue $(2 + 2^{\alpha+1})\mathsf{M}(n)$ opérations alors que l'algorithme 3.5 en effectue $(2 + \frac{8}{2^\alpha})\mathsf{M}(n)$ opérations. Le tableau 3.2 page 33 donne le nombre de multiplications de polynômes de degré n effectuées par les algorithmes en fonction de α. Sauf si on utilise une FFT, l'algorithme 3.5 est plus rapide que l'algorithme 3.1.

Entrée: $p \in \mathbb{K}[x]$ avec $\deg(p, x) \leqslant n$
Sortie: $p(g) \mod x^n$

Calculer $p_{0,0}$ et $p_{0,1}$ tels que $p(x) := p_{0,0}(x^2) + xp_{0,1}(x^2)$
pour $i \in \{0, 1\}$ **faire**
  $d_i := \deg(p_{0,i}, x)$
  $p_{1,i} := p_{0,i}(2x - 1)$
  $p_{2,i} := x^{d_i} p_{1,i}\left(\frac{1}{x}\right)$
  $p_{3,i} := p_{2,i}(x + 1)$
  Calculer $p_{4,i,0}$ and $p_{4,i,1}$ tels que $p_{3,i}(x) := p_{4,i,0}(x^2) + xp_{4,i,1}(x^2)$
  pour $j \in \{0, 1\}$ **faire**
    $p_{5,i,j} := p_{4,i,j}(1 - x)$
  fin pour
fin pour
$f_1 := \sqrt{1 - x}$
$f_2 := \frac{1}{1+\sqrt{1-x}}$
$p_{6,0} := f_2^{d_0} p_{5,0,0} + f_1 f_2^{d_0} p_{5,0,1} \mod x^{d_0}$
$p_{6,1} := f_2^{d_1+1} p_{5,1,0} + f_1 f_2^{d_1+1} p_{5,1,1} \mod x^{d_1}$
renvoyer $p_{6,0}(x^2) + xp_{6,1}(x^2)$

Algorithme 3.10: $\mathsf{eval}\left(., \sqrt{\frac{2}{\sqrt{1-x^2}+1} - 1}\right)$

3.4 Algorithme $\mathsf{MtoC_n}$

La première approche, pour effectuer l'algorithme inverse, est d'inverser chaque application de l'algorithme direct, donc d'inverser chaque application de la factorisation (3.9). L'inversion de l'algorithme $\mathsf{mul}^t(\cdot, v)$ se déduit rapidement de l'inverse de v qui est $v^{-1} = 2\frac{1+t^2}{1-t^2}$. En effet comme l'inverse d'une application transposée est la transposée de l'application inverse, on a donc

$$\left(\mathsf{mul}^t(\cdot, v)\right)^{-1} = \mathsf{mul}^t(\cdot, v^{-1}).$$

Comme pour l'application $\mathsf{mul}(\cdot, v)$, la remarque 3.9 entraîne que cette opération s'effectue en $\mathcal{O}(n)$ opérations arithmétiques.

Algorithme inverse de $\mathsf{eval}(\cdot, h)$ Le second algorithme à inverser est l'algorithme $\mathsf{eval}^t(\cdot, h)$. L'approche la plus simple est sûrement d'inverser d'abord l'algorithme $\mathsf{eval}(\cdot, h)$, ce qui revient à donner un algorithme rapide pour le calcul de $\mathsf{eval}(\cdot, g)$ où g est l'inverse compositionnelle de h ($h(g) = \mathrm{id}$). En utilisant encore le principe que la transposée de l'inverse d'une matrice est égale à l'inverse de la transposée de la même matrice, l'algorithme de l'inverse de $\mathsf{eval}^t(\cdot, h)$, se déduit en transposant l'algorithme $\mathsf{eval}(\cdot, g)$.

Une façon d'inverser la fonction h est d'inverser directement terme par terme sa représentation (3.10) page 39. De cette manière, on obtient directement l'écriture intéressante

de g (dans le sens où on isole la variable x) :

$$g(x) \coloneqq \sqrt{\frac{2}{\sqrt{1-x^2}+1} - 1}. \tag{3.11}$$

Comme la formule (3.10), cette formule permet de déduire l'algorithme 3.10 pour calculer $\mathsf{eval}(\cdot, g)$ en n'utilisant que des applications *simples*.

Cet algorithme est semblable à 3.6. Pour vérifier sa validité, il suffit d'analyser chaque changement de variable effectué. Quelques remarques sont quand même nécessaires. La fonction f_1 définie à la ligne 12 permet de retrouver la racine carrée de la ligne 7. La fonction f_2 définie à la ligne 13, a une double utilités, elle permet de corriger la multiplication par x^{d_i} de la ligne 13 mais aussi de retrouver la racine carrée de la ligne 1, grâce à l'égalité :

$$\frac{\sqrt{x}}{1+\sqrt{1-x}} = \sqrt{\frac{2}{\sqrt{1-x}+1} - 1},$$

ce qui explique le +1 dans l'exposant de f_2 de la ligne 15. Selon la formule précédente, lors de la multiplication par f_2, il manque un facteur $\sqrt{x}$ pour récupérer la racine carrée. La multiplication par x du polynôme $p_{6,1}$ à la dernière ligne permet de récupérer ce facteur manquant. Pour le calcul des coefficients des développements en séries des fonctions f_2^d et $f_1 f_2^d$, on peut utiliser les récurrences vérifiées par ces coefficients.

$$\begin{aligned}
\text{Les coefficients de } f_2^d \ &: \ (d+2n+1)(d+2n)u_n - 4(n+d+1)(n+1)u_{n+1} = 0 \\
\text{Les coefficients de } f_1 f_2^d \ &: \ (d+2\,n)\,(d+2\,n-1)\,u_n + 4\,(n+2)\,(n+2+d)\,u_{n+2} \\
&\quad - \left(8\,n^2 + (8\,d+14)\,n + 4 + d^2 + 7\,d\right) u_{n+1} = 0
\end{aligned}$$

Algorithme inverse de $\mathsf{eval}^t(\cdot, h)$ On obtient par transposition l'algorithme 3.11.

Proposition 3.13. *L'algorithme 3.11 permet de calculer l'opération* $\mathsf{eval}(\cdot, g)$ *en* $8\mathsf{M}(\frac{n}{2}) + 4\mathsf{M}(\frac{n}{4}) + \mathcal{O}(n)$

Démonstration. La correction de cet algorithme se déduit de la correction de l'algorithme $\mathsf{eval}(\cdot, g)$. Aux lignes 4, 5, quatre multiplications avec des polynômes de degré $n/2$ sont effectuées. Dans la première boucle, deux transposées de l'opération de décalage sont effectuées aux lignes 11 et 13 à chaque fois avec des polynômes de degré au plus $n/2$; il y a deux passages dans cette boucle, la complexité de ces opérations est donc $4\mathsf{M}(\frac{n}{2})$. Dans la seconde boucle un dernier shift^t est effectué avec un polynôme de degré au plus $n/4$; il y a quatre passages dans cette boucle, la complexité de ces opérations est donc $4\mathsf{M}(\frac{n}{4})$. L'addition de toutes ces complexités nous donne bien la complexité annoncée dans la proposition. □

Entrée: $p \in \mathbb{K}_n[x]$
Sortie: $\mathsf{eval}^t(p, g)$

1: Calculer $p_{6,0}$ et $p_{6,1}$ tels que $p(x) := p_{6,0}(x^2) + xp_{6,1}(x^2)$
2: $d_0 := \deg(p_{6,0}, x)$ et $d_1 := \deg(p_{6,1}, x)$
3: $f_1 := \sqrt{1-x}$, $f_2 := \frac{1}{1+\sqrt{1-x}}$
4: $p_{5,0,0} := \mathsf{mul}^t\left(p_{6,0}, f_2^{d_0}\right) \mod x^{d_0/2}$, $p_{5,0,1} := \mathsf{mul}^t\left(\mathsf{p}_{6,0}, \mathsf{f}_1\mathsf{f}_2^{\mathsf{d}_0}\right) \mod \mathsf{x}^{\mathsf{d}_0/2}$
5: $p_{5,1,0} := \mathsf{mul}^t\left(\mathsf{p}_{6,1}, \mathsf{f}_2^{\mathsf{d}_1+1}\right) \mod \mathsf{x}^{\mathsf{d}_1/2}$, $p_{5,1,1} := \mathsf{mul}^t\left(\mathsf{p}_{6,1}, \mathsf{f}_1\mathsf{f}_2^{\mathsf{d}_1+1}\right) \mod \mathsf{x}^{\mathsf{d}_1/2}$
6: **pour** $i \in \{0,1\}$ **faire**
7: **pour** $j \in \{0,1\}$ **faire**
8: $p_{4,i,j} := \mathsf{shift}^t\left(p_{5,i,j}(-x)\right)$
9: **fin pour**
10: $p_{3,i} := p_{4,i,0}(x^2) + xp_{4,i,1}(x^2)$
11: $p_{2,i} := \mathsf{shift}^t\left(p_{3,i}\right)$
12: $p_{1,i} := x^{d_i} p_{2,i}\left(\frac{1}{x}\right)$
13: $p_{0,i}(-x) := \mathsf{shift}^t\left(p_{1,i}\right)$
14: **fin pour**
15: **renvoyer** $p_{0,0}(x^2) + xp_{0,1}(x^2)$

Algorithme 3.11: $\mathsf{eval}^t\left(\cdot, \sqrt{\frac{2}{\sqrt{1-x^2}+1} - 1}\right)$

Entrée: $p := (p_0, p_1, \ldots, p_n)$
Sortie: $b := (b_0, b_1, \ldots, b_n)$ tels que $\sum' b_i T_i(x) = \sum p_i x^i$

1: $a := \mathsf{eval}^t(p, g)$
2: **renvoyer** $b := \mathsf{mul}^t(a, v^{-1})$

Algorithme 3.12: $\mathsf{MtoC_n}$

Algorithme $\mathsf{MtoC_n}$ L'algorithme $\mathsf{MtoC_n}$ se déduit alors de la factorisation :

$$\mathsf{MtoC_n} = \mathsf{mult}^t(\cdot, v^{-1}) \circ \mathsf{eval}^t(\cdot, g).$$

Le théorème suivant 3.14 est une conséquence de cet algorithme.

Théorème 3.14. *Soit $p = (p_0, p_1, \ldots, p_n)$, l'algorithme 3.12 calcule l'application* $\mathsf{MtoC_n}$ *en* $8\mathsf{M}(n/2) + 4\mathsf{M}(n/4)$ *opérations arithmétiques*

Remarque 3.15. Si $\mathsf{M}(n) = \tilde{O}(n^\alpha)$, l'algorithme de Pan 3.1 pour calculer l'application $\mathsf{MtoC_n}$ effectue $(2 + 2^{\alpha+1})\mathsf{M}(n)$ opérations alors que l'algorithme 3.5 en effectue $(\frac{8}{2^\alpha} + \frac{4}{4^\alpha})\mathsf{M}(n)$ opérations. Le tableau 3.3 page 33 donne alors le nombre de multiplications de polynômes de degré n effectuées par les algorithmes en fonction de α.

L'algorithme 3.5 est toujours asymptotiquement plus rapide que l'algorithme inverse de 3.1.

4 Multiplication et division rapides

Les algorithmes de multiplication et division [Str72] dans la base monomiale s'effectuent en $\mathcal{O}(\mathsf{M}(n))$ opérations arithmétiques. Par les algorithmes de conversion de bases décrits dans la section précédente, des algorithmes de multiplication et de division entre des polynômes exprimés dans la base de Tchebychev en $\mathcal{O}(\mathsf{M}(n))$ opérations se déduisent immédiatement. Le problème de l'utilisation des algorithmes pour le calcul de multiplication ou de division à base de conversions est que la constante multiplicative dans la complexité est importante ; cette constante est due à la conversion de la base de Tchebychev dans la base monomiale puis à la conversion inverse qui doivent être calculées. Par exemple, le coût pour multiplier deux polynômes de degré plus petit que n en utilisant cette technique est alors de $13\mathsf{M}(n) + 20\mathsf{M}\left(\frac{n}{2}\right) + \mathcal{O}(n)$ opérations arithmétiques.

Dans cette section, des algorithmes rapides directs pour ces deux opérations sont présentés. Les algorithmes de multiplication sont issus de [Gio12], alors que l'algorithme de division rapide est une nouvelle contribution.

4.1 Multiplication rapide

Étant donnés deux polynômes $a := \frac{1}{2}a_0T_0(x) + a_1T_1(x) + \cdots + a_nT_n(x)$ et $b := \frac{1}{2}b_0T_0(x) + b_1T_1(x) + \cdots + b_m(x)T_m(x)$, cette section donne un algorithme pour calculer le produit de a et b, c'est-à-dire le polynôme c tel que $c := \frac{1}{2}c_0T_0(x) + c_1T_1(x) + \cdots + c_{n+m}T_{n+m}(x)$. Un algorithme direct se déduit de la formule du produit de deux polynômes de Tchebychev :

$$T_n(x)T_m(x) = \frac{1}{2}\left(T_{n+m}(x) + T_{|n-m|}(x)\right).$$

Cet algorithme ressemble à l'algorithme de multiplication naïf dans la base monomiale. La complexité de cet algorithme est $\mathcal{O}(nm)$ opérations arithmétiques. Dans la base monomiale, d'autres algorithmes plus rapides existent pour multiplier des polynômes comme par exemple l'algorithme de Karatsuba. Pascal Giorgi [Gio12] a réduit la multiplication de polynômes dans la base de Tchebychev à deux multiplications dans la base monomiale (voir algorithme 3.13). La complexité de la multiplication en utilisant cette réduction devient très bonne, la multiplication dans la base monomiale étant optimisée. Un algorithme numérique asymptotiquement plus rapide et qui permet de calculer la multiplication sans se ramener à la multiplication entre polynômes dans la base monomiale a été donné par Baszenski et Tasche [BT97]. Cet algorithme utilise la transformée en cosinus discrète qui est bien adaptée pour la multiplication de polynômes de Tchebychev. Giorgi [Gio12] montre qu'expérimentalement cet algorithme à base de cosinus direct n'est pas stable numériquement. Dans cette section, on ne rentre pas dans les détails de l'algorithme de multiplication à base de cosinus discret et le lecteur est invité à consulter [BT97, Gio12] pour plus de détail.

Algorithmes de Giorgi

Les algorithmes de Giorgi pour calculer ce produit utilisent la formule (3.1) page 33. Grâce à cette formule, la multiplication entre deux polynômes exprimés dans la base des polynômes de Tchebychev se ramène à la multiplication des polynômes de Laurent $\frac{1}{2}\left(a_{n-1}x^{-n+1} + \cdots + a_{n-1}x^{n-1}\right)$ et $\frac{1}{2}\left(b_{m-1}x^{-m+1} + \cdots + b_{m-1}x^{m-1}\right)$. Le produit de ces deux polynômes de Laurent est $\frac{1}{2}\left(c_{n+m-2}x^{-n-m+2} + \cdots + c_{n+m-2}x^{n+m-2}\right)$. L'algorithme de multiplication 3.13 de deux polynômes dans la base de Tchebychev suivi de la proposition 3.16 se déduit alors de cette remarque.

Entrée: $a = \frac{1}{2}\sum_{k=0}^{n-1}{}'a_kT_k(x)$, $b = \frac{1}{2}\sum_{k=0}^{m-1}{}'b_kT_k(x)$ avec $a \in \mathbb{K}[x]$ et $b \in \mathbb{K}[x]$
Sortie: $\sum_{k=0}^{n+m-2}{}'c_kT_k(x) = ab$
1: Calculer $p_1 := \sum_{k=0}^{n-1}{}'a_kx^k \sum_{k=0}^{m-1}{}'b_kx^k$
2: Calculer $p_2 := \sum_{k=0}^{n-1}{}'a_kx^k \sum_{k=0}^{m-1} b_{m-k-1}x^k - b_0x^k/2$
3: **renvoyer** les coefficients c_i tels que

$$\frac{1}{2}\sum_{k=-n-m+2}^{n+m-2} c_ix^i = p_1(x) + p_1\left(\frac{1}{x}\right) + x^{-m+1}p_2(x) + x^{m-1}p_2\left(\frac{1}{x}\right)$$

Algorithme 3.13: Multiplication de polynômes dans la base de Tchebychev

Proposition 3.16 ([Gio12])**.** *L'algorithme 3.13 permet de calculer la multiplication entre deux polynômes de degré plus petit que n et m exprimés dans la base des polynômes de Tchebychev en au plus* $2\mathsf{M}(\max(n,m)) + \mathcal{O}(n+m)$ *opérations arithmétiques.*

Démonstration. La correction de cet algorithme se déduit de la formule (3.1) page 33. La complexité de cet algorithme vient des deux multiplications de polynômes de degré respectifs n et m. Les autres opérations étant des additions ou de la réécriture (inverser les coefficients), elles s'effectuent en $\mathcal{O}(n)$ opérations arithmétiques. □

Cet algorithme peut être accéléré lorsque la FFT est utilisée. En effet, lors de la multiplication des lignes 1 et 2, on multiplie à gauche par le même polynôme, on peut alors éviter une FFT en ne calculant la transformation qu'une seule fois au lieu de deux. De manière moins évidente, les transformations des facteurs droits des produits peuvent se calculer en utilisant une seule FFT. Le $k^{\text{ème}}$ coefficient du vecteur obtenu par la FFT du polynôme $x^m b\left(\frac{1}{x}\right)$ est donné par :

$$\sum_{i=0}^{m} b_i\omega^{-ki}, \text{ où } \omega \text{ est une racine primitive } n+m^{\text{ème}} \text{ de l'unité,}$$

ce qui correspond au $n+m-k^{\text{ème}}$ coefficient du vecteur obtenu par la FFT du polynôme b grâce à l'égalité $\omega^{-1} = \omega^{n+m-1}$. On peut donc bien déduire la FFT de $x^m b\left(\frac{1}{x}\right)$ par inversion du vecteur obtenu par la FFT de b. Cette remarque permet d'éviter de calculer 2 FFT de

taille $2n$ et donc les deux multiplications s'effectuent en 4 FFT de taille $2n$ au lieu des 6. On en déduit alors la proposition suivante :

Proposition 3.17 (Giorgi [Gio12]). *La multiplication par FFT de deux polynômes exprimés dans la base des polynômes de degré n et m de Tchebychev se calcule en $\frac{4}{3}\mathsf{M}(max(n,m)) + \mathcal{O}(n+m)$ opérations arithmétiques.*

4.2 Division rapide

L'opération entre deux polynômes qui vient juste après la multiplication est la division euclidienne. Le but de cette section est de donner des algorithmes pour le calcul de la division euclidienne notamment un nouvel algorithme de complexité proche de la complexité de la division entre deux polynômes dans la base monomiale.

Algorithme naïf

Un algorithme naïf pour cette division est déduit de l'algorithme de division entre deux polynômes dans la base monomiale. On trouve alors l'algorithme 4.2 qui a la même complexité que l'algorithme naïf de division dans la base monomiale. La correction de cet

Entrée: $a = \sum_{i=0}^{n-1}{}' a_i x^i$ et $b = \sum_{i=0}^{m-1}{}' b_i x^i$ dans $\mathbb{K}[x]$ avec $n > m$ et $a_{n-1} \neq 0$ et $b_{m-1} \neq 0$
Sortie: $r = \sum_{i=0}^{n-1}{}' r_i x^i$, $q = \sum_{i=0}^{n-1}{}' q_i x^i \in \mathbb{K}[x]$ tels que $a = bq + r$

```
i := n − m, q := 0, r := a
c_b := b_{m−1}
tantque i ≠ 0 faire
    c_r := coeff(r, x, i + m − 1)
    q := q + (2c_r / c_b) T_i(x)
    r := r − c_r T_i(x) (1/c_b) b
    i := i − 1
fin tantque
```

Algorithme 3.14: Algorithme de division entre polynômes de Tchebychev

algorithme se déduit de la formule de multiplication entre deux polynômes de Tchebychev :

$$T_n T_m = \frac{1}{2}\left(T_{n+m} + T_{|n-m|}\right).$$

Dans la boucle, $n-m$ passages sont effectués. La seule opération de cette boucle s'effectuant en temps non constant est le calcul du reste r. La multiplication d'un monôme par le polynôme b suivi de la soustraction se calcule en au plus $\mathcal{O}(m)$ opérations arithmétiques. En effet si b est de degré m, le polynôme $c_r T_i(x)$ est un polynôme qui contient au plus $2m$ coefficients. l'addition entre un polynôme contenant n coefficients et un polynôme contenant $2m$ coefficients s'effectue en $2m$ opérations si $n > 2m$, ce qui explique la complexité du calcul de chaque r. La complexité de cet algorithme est donc bien $\mathcal{O}(nm)$ opérations

arithmétiques. On peut remarquer que l'on obtient la même complexité que pour le calcul du polynôme dans la base monomiale [vzGG03].

Algorithme rapide par conversion

Dans la base monomiale, grâce à la méthode de Newton, Strassen [Str72] (voir aussi [vzGG03]) a ramené le calcul d'un quotient entre deux polynômes de degré respectif plus petit que n et m en $5\mathsf{M}(n-m)+\mathcal{O}(n-m)$ opérations arithmétiques et le calcul du reste en $\mathsf{M}(n)+\mathcal{O}(n)$ en multipliant le quotient obtenu par le dénominateur. On peut donc obtenir une complexité quasi-linéaire pour la division en utilisant l'algorithme de conversion de la section 3.

Nouvel algorithme rapide

L'algorithme que je propose ici permet de calculer le quotient entre deux polynômes de Tchebychev en la même complexité que pour le calcul dans la base monomiale (à un nombre linéaire d'opérations près).

La division entre les polynômes a et b permet de calculer le polynôme q de degré $n-m$ et r de degré $m-2$ tels que $a = bq + r$. Cette égalité reste vraie lorsque l'on évalue ces polynômes en $\frac{x+x^{-1}}{2}$. Si a_i, b_i, q_i et r_i sont respectivement les coefficients dans la base de Tchebychev des polynômes a, b, q et r, alors l'égalité (3.1) page 33 nous donne :

$$\sum_{i=-n+1}^{n-1} a_{|i|}x^i = \sum_{i=-m+1}^{m-1} b_{|i|}x^i \sum_{i=-n+m}^{n-m} q_{|i|}x^i + \sum_{i=-m+2}^{m-2} r_{|i|}x^i.$$

Dans cette équation, on souhaite calculer les coefficients $q_0, \dots q_{n-m}$. En coupant le polynôme de Laurent $q\left(\frac{x+x^{-1}}{2}\right)$ en deux, on garde seulement les coefficients associés aux degrés positifs. Si de plus on multiplie cette équation par x^n pour obtenir une égalité entre polynômes, on a :

$$\sum_{i=0}^{2n-2} a_{|i-n+1|}x^i - \sum_{i=-m+n}^{m+n} b_{|i-n+1|}x^i \sum_{i=n-1}^{2n-m-1} q_{|i-n+1|}x^i \tag{3.12}$$

$$= \sum_{i=-m+n}^{m+n-2} b_{|i-n+1|}x^i \sum_{i=n-1}^{2n-m-1} q_{|i-n+1|}x^i + \sum_{i=n-m+1}^{n+m-3} r_{|i-n+1|}x^i. \tag{3.13}$$

La partie droite de cette équation est un polynôme de degré $n+m-3$, cette équation représente donc l'égalité déduite de la division euclidienne entre

$$\sum_{i=0}^{2n-1} a_{|i-n+1|}x^i \quad \text{et} \quad \sum_{i=-m+n}^{m+n-2} b_{i-n+1}x^i,$$

le polynôme $\sum_{i=n}^{2n-m} q_{i-n}x^i$ et le membre droit de l'équation étant respectivement le quotient et le reste de cette division. De cette façon la division entre a et b dans la base de Tchebychev

Entrée: $a = \sum_{i=0}^{n-1}{}' a_i T_i(x)$ et $b = \sum_{i=0}^{m-1}{}' b_i T_i(x)$ des polynômes dans $\mathbb{K}[x]$
Sortie: $q := \sum_{i=0}^{n-m}{}' q_i T_i(x)$ et $r := \sum_{i=0}^{n-2}{}' r_i T_i(x)$ tels que $a = bq + r$.
1: Calculer le quotient $\sum_{i=0}^{n-m} q_i x^i$ de la division de $\sum_{i=0}^{2n-2} a_{|i-n+1|} x^i$ par $\sum_{i=n-m}^{n+m-2} b_{|i-n+1|} x^i$
2: Calculer $r = a - b \sum_{i=0}^{n-m} q_i T_i(x)$
3: **renvoyer** $\sum_{i=0}^{n-m}{}' q_i T_i(x),\ r$

Algorithme 3.15: Division euclidienne rapide dans la base de Tchebychev

est ramenée à la division entre les polynômes

$$\sum_{i=0}^{2n-2} a_{|i-n+1|} x^i \quad \text{et} \quad \sum_{i=-m+n}^{m+n-2} b_{|i-n+1|} x^i$$

dans la base monomiale. L'algorithme 3.15 s'en déduit.

Théorème 3.18. *L'algorithme 3.15 calcule le quotient et le reste de la division euclidienne dans la base des polynômes de Tchebychev d'un polynôme de degré strictement plus petit que n par un polynôme de degré strictement plus petit que m, les deux exprimés dans la base des polynômes de Tchebychev.*

Le quotient se calcule en $5\mathsf{M}(n-m) + \mathcal{O}(n-m)$ opérations arithmétiques et le reste se calcule en $2\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques.

Démonstration. La correction de cet algorithme se déduit de l'équation (3.12).

Pour calculer le quotient, cet algorithme se ramène à calculer un quotient dans la base monomiale entre un polynôme de degré $2n$ et $n+m$, ce qui s'effectue avec l'algorithme de Strassen [Str72] en $5\mathsf{M}(n-m)$ opérations arithmétiques. Le calcul du reste effectué à la ligne 2 se ramène à la multiplication entre deux polynômes dans la base de Tchebychev, ce qui selon la proposition 3.16 s'effectue en au plus $2\mathsf{M}(n) + \mathcal{O}(n)$ opérations arithmétiques. □

Chapitre 4

Polynômes et fractions de Ore

Résumé

Ce chapitre rappelle la théorie des polynômes de Ore qui permet de représenter les opérateurs différentiels ou de récurrence comme des polynômes. Il donne aussi des résultats de complexité sur les opérations de multiplication et de division entre ces polynômes. Finalement, et c'est l'objectif principal de ce chapitre, les fractions d'opérateurs de Ore sont présentées ainsi que leurs principales propriétés.

Sommaire

Une façon de représenter une équation différentielle linéaire à coefficients polynomiaux ou une récurrence linéaire à coefficients polynomiaux est de l'écrire comme opérateur agissant sur une fonction ou sur une suite. La fonction $f := e^{x^2}$ est solution de l'équation différentielle :

$$-2xy(x) + y'(x) = 0,$$

ce qui s'écrit en termes d'opérateurs :

$$(-2x + \partial_x) \cdot f = 0$$

où ∂_x est l'opérateur de dérivation. La suite $\left(\binom{n}{k}\right)_{n\in\mathbb{Z}}$ est solution de la récurrence

$$(n+1)u_n - (n+1-k)u_{n+1} = 0,$$

ou en termes d'opérateurs :

$$(n+1-(n+1-k)S_n) \cdot \left(\binom{n}{k}\right)_{n\in\mathbb{Z}} = 0$$

où S_n est l'opérateur de décalage agissant sur les suites par $S_n \cdot (u_n)_{n\in\mathbb{Z}} = (u_{n+1})_{n\in\mathbb{Z}}$. Deux opérations naturelles arrivent avec ces opérateurs, l'addition et la composition, qui n'est pas commutative. Par exemple dans le cas différentiel, la règle de commutativité

$$\partial_x x = x\partial_x + 1,$$

se déduit de la règle de Leibniz :

$$(fg)' = f'g + fg'.$$

Guillaume Libri [Lib33, Lib36] a été le premier à créer un lien entre la théorie des équations différentielles linéaires et la théorie des équations algébriques. Il montre notamment comment baisser l'ordre d'une équation différentielle lorsque l'on en connaît une solution particulière. Par la suite, plusieurs grands mathématiciens du XX$^{\text{ème}}$ siècle ont développé ce lien. Paul Émile Appell [App81] donne dans l'introduction de son mémoire un historique assez bref dans ce domaine. Même si Libri avait montré que ces résultats s'appliquaient aux équations aux différences, c'est-à-dire aux équations en $\Delta_n = S_{n+1} - S_n$, ceux-ci ont essentiellement été développés pour les équations différentielles linéaires.

Oystein Ore [Ore31, Ore33] a développé une théorie plus générale sur ces opérateurs en les identifiant comme polynômes non-commutatifs qui sont appelés dans cette thèse polynômes de Ore ou polynômes tordus. Des cas particuliers des polynômes de Ore sont les opérateurs différentiels mais aussi les opérateurs de récurrence ou encore les opérateurs aux différences. Des deux opérations naturelles Ore a déduit des lois pour construire un anneau de polynômes. Il a montré que ces anneaux sont euclidiens et donné une méthode pour calculer les p.p.c.m. entre opérateurs. Dans les mêmes articles, Ore définit le corps des fractions d'opérateurs, corps qui interviennent de façon centrale dans cette thèse. L'objectif

de ce chapitre est de définir ce corps et d'en donner les principales propriétés qui sont pour la plupart dues à Ore.

Pour y parvenir, les principales propriétés des polynômes de Ore, limitées essentiellement aux opérateurs différentiels linéaires et aux opérateurs de récurrence linéaires à chaque fois à coefficients des fractions rationnelles, sont définies et rappelées dans ce chapitre. Ore, en utilisant l'algorithme d'Euclide non commutatif entre opérateurs, a défini leurs p.p.c.m et le p.g.c.d. Manuel Bronstein et Marko Petkovšek [BP94b] ont donné une version plus moderne des algorithmes de calcul de ces p.g.c.d et p.p.c.m. Ces algorithmes et un algorithme de multiplication rapide dû à van der Hoeven [vdH02] sont rappelés dans la section 1.

Pour finir, les fractions de polynômes de Ore sont définies en section 2 Les opérations entre fractions sont rendues effectives d'une part par les algorithmes donnés dans ce chapitre et d'autre part par l'explication du package OreField. Ce package Maple a été développé lors de cette thèse et sert de brique de base pour le package gfsRecurrence présenté dans le chapitre 8.

1 Polynômes de Ore

Cette section présente un rappel sur les polynômes de Ore. Pour plus de détails on peut se référer à [BP94b, Chy98].

1.1 Introduction aux anneaux de polynômes de Ore

Les anneaux des opérateurs différentiels et des opérateurs de récurrence linéaire sont des cas particuliers d'anneau de polynômes de Ore. Ces anneaux sont munis des règles de commutation

$$\frac{d}{dx}p(x) = p(x)\frac{d}{dx} + p'(x) = 0; \qquad S_n p(n) = p(n+1)S_n.$$

Plus généralement, un anneau de polynômes en une indéterminée ∂ avec des coefficients dans un corps $\mathbb{K}$ est un anneau de polynômes de Ore, quand son produit est défini par la règle de commutation

$$\partial p = \sigma(p)\partial + \delta(p), \qquad p \in \mathbb{K} \tag{4.1}$$

où pour tous a et b dans $\mathbb{K}$,

$$\begin{aligned} \sigma(a+b) &= \sigma(a) + \sigma(b), \quad \sigma(ab) = \sigma(a)\sigma(b), \quad \sigma(a^{-1}) = \sigma(a)^{-1}, \\ \delta(a+b) &= \delta(a) + \delta(b), \quad \delta(ab) = \sigma(a)\delta(b) + \delta(a)b. \end{aligned}$$

Cet anneau est dénoté $\mathbb{K}\langle\partial;\sigma,\delta\rangle$. Les opérateurs différentiels linéaires sont obtenus avec $\sigma = \mathrm{Id}$ et $\delta = d/dx$; les opérateurs de récurrence linéaires avec $\sigma = S_n$ et $\delta = 0$. L'opération $S_n^{-1} \cdot (u_n)_{n\in\mathbb{Z}} = (u_{n-1})_{n\in\mathbb{Z}}$ est aussi naturelle, il est alors possible de définir des polynômes de Laurent-Ore avec $\delta\delta^{-1} = \delta^{-1}\delta = 1$ et $\delta^{-1}a = \sigma^{-1}(a)\delta^{-1}$. Les opérateurs de récurrence linéaires de Laurent-Ore seront souvent utilisés dans cette thèse.

Entrée: $A = \sum_{i=0}^{r_A} a_i(n)S_n^i$ et $B = \sum_{i=0}^{r_B} b_i(n)S_n^i$, où $a_i \in \mathbb{K}[n]$ et $b_i \in \mathbb{K}[n]$
Sortie: Le produit $C = BA = \sum_{i=0}^{d_C} c_i(n)S_n^i$ où $c_i \in \mathbb{K}[n]$
  $d_C := r_A + r_B$
  pour l de 0 à d_C **faire**
    $c_l(n) := \sum_{i=0}^{l} b_i(n)a_{l-i}(n+i)$ {avec $a_i(n) = 0$ pour $i > r_A$ et $b_i(n) = 0$ pour $i > r_B$}
  fin pour
  renvoyer $C = \sum_{i=0}^{d_C} c_i(n)S_n^i$

Algorithme 4.1: Multiplication tordue naïve

Notation 4.1. Afin de distinguer l'action d'un opérateur et le produit dans l'anneau d'opérateurs, qui correspond à la composition des actions, la notation $\cdot$ est utilisée pour celle-ci. Ainsi :

$$\partial_x \cdot f = f', \ S_n \cdot u_n = u_{n+1}.$$

Notation 4.2. Pour simplifier les notations, on notera les anneaux d'opérateurs différentiels et d'opérateurs de récurrence respectivement par $\mathbb{K}(x)\langle\partial_x\rangle$ et $\mathbb{K}(n)\langle S_n\rangle$. Pour certains algorithmes, on s'autorisera aussi à travailler dans leurs sous-anneaux, les opérateurs à coefficients polynomiaux notés $\mathbb{K}[x]\langle\partial_x\rangle$ et $\mathbb{K}[n]\langle S_n\rangle$.

Remarque 4.3. L'anneau des opérateurs différentiels linéaires à coefficients polynomiaux, $\mathbb{K}[x]\langle\partial_x\rangle$ est isomorphe à l'algèbre libre $\mathbb{K}\langle x, \partial_x\rangle$ quotientée par le polynôme $\partial_x x + 1 - x\partial_x$.

L'algorithme 4.1 ci-dessus décrit une méthode naïve pour multiplier deux opérateurs de récurrence, qui sera utilisée dans ce mémoire.

Comme pour l'algorithme de multiplication entre polynômes dans le cas commutatif, cet algorithme se généralise facilement à la multiplication de polynômes de Laurent-Ore.

La principale propriété des anneaux de polynômes de Ore, est que le degré (en la variable ∂_x) 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. L'argument s'étend aux polynômes de Ore. L'algorithme de division à droite existe toujours, la division à gauche existe lorsque σ est inversible, c'est-à-dire que σ est un automorphisme. C'est le cas dans l'anneau des opérateurs différentiels où σ est l'identité. C'est aussi le cas dans l'anneau des opérateurs de récurrence où $\sigma = S_n$ est un automorphisme de $\mathbb{K}(n)$ dans lui-même (à tout élément de $a(n) \in \mathbb{K}(n)$, on a $a(n-1) \in \mathbb{K}(n)$ tel que $\sigma(a(n-1)) = a(n)$). La principale différence avec la division commutative est que les divisions à droite et à gauche sont différentes.

Exemple 4.4. Dans le cas des opérateurs de récurrence si $A := nS_n^2 - n$ et $B := S_n + 1$, en effectuant une division euclidienne à droite on a :

$$A = n(S_n - 1)B,$$

en l'effectuant à gauche, on a :

$$A = B(S_n - 1)(n - 2) - 2.$$

Entrée: $A, B \in \mathbb{K}(n)\langle S_n\rangle$
Sortie: $R, Q \in \mathbb{K}(n)\langle S_n\rangle$ tels que $A = QB + R$ et $\deg R < \deg Q$
  $d_A := \text{degree}(A, S_n)$ et $d_B := \text{degree}(B, S_n)$
  $i := d_A - d_B$, $Q := 0$, $R := A$
  $C_B := \text{coeff}(B, d_B)$
  tantque $i \neq 0$ **faire**
    $C_R := \text{coeff}(R, i + d_B)$
    $Q := Q + C_R S_n^i \frac{1}{C_B}$
    $R := R - C_R S_n^i \frac{1}{C_B} B$
    $i := i - 1$
  fin tantque
  renvoyer Q et R

Algorithme 4.2: Algorithme de division à droite

Remarque 4.5. Une autre conséquence du stathme est que cet anneau est sans diviseur de zéro.

L'algorithme de division à droite entre deux opérateurs de récurrence 4.2 est décrit ci-dessus. On retrouve le même algorithme que dans le cas commutatif, en faisant attention à multiplier par ∂_x toujours du même côté.

La notation $\text{coeff}(B, r_B)$ représente le coefficient de $S_n^{r_B}$ dans B. C'est une fraction rationnelle en n. Comme pour la division euclidienne entre deux polynômes commutatifs, la terminaison de cet algorithme s'explique par la décroissance stricte des degrés du polynôme R à chaque itération dans la boucle.

Remarque 4.6. L'algorithme 4.2 se généralise aux polynômes de Laurent-Ore. En effet, si A et B sont des polynômes de Laurent-Ore de valuation (la valuation est le petit degré des monômes) r_A et r_B et $d = \min(r_A, r_B)$. Alors $S_n^d Q S_n^{-d}$ et $S_n^d R$, définis par la division entre $S_n^{-d} A$ et $S_n^{-d} B$:

$$S_n^{-d} A = Q S_n^{-d} B + R,$$

sont respectivement le quotient et le reste de la division de A par B.

On déduit de cet algorithme de division l'algorithme d'Euclide étendu à droite qui permet de calculer le plus grand commun diviseur à droite (noté gcrd comme *greatest common right divisor*), les coefficients de Bézout (les opérateurs U et V tel que $UA+VB = 1$) et un p.p.c.m cette fois-ci à gauche qui est noté lclm (*least common left multiple*). Dans l'algorithme 4.3, qui décrit l'algorithme d'Euclide étendu à droite, les opérations $\text{quotient}_{\text{droit}}(A, B)$ et $A \bmod_{\text{droit}} B$ sont respectivement le quotient et le reste de la division euclidienne à droite de A par B. On déduit de celui-ci la proposition suivante.

Proposition 4.7. *L'algorithme 4.3 prend en entrée deux opérateurs de récurrence à coefficients des fractions rationnelles en n et renvoie le* gcrd*, les coefficients de Bézout de ce* gcrd *et les cofacteurs du* lclm *entre les deux opérateurs.*

Entrée: $A, B \in \mathbb{K}(n)\langle S_n\rangle$
Sortie: $\operatorname{gcrd}(A,B)$, U_1 et V_1 tels que $U_1A + V_1B = \operatorname{gcrd}(A,B)$ et
U_2 et V_2 tels que $U_2A = -V_2B = \operatorname{lclm}(A,B)$
$R_0 := A$
$R_1 := B$
$U_0 := 1,\ V_0 := 0$
$U_1 := 0,\ V_1 := 1$
$i := 1$
tantque $R_i \neq 0$ **faire**
$Q_i, R_{i+1} := (\text{quotient et reste})_{\text{droit}}(R_{i-1}, R_i)$ par l'algorithme 4.2
$U_{i+1} := U_{i-1} - Q_iU_i,\ V_{i+1} := V_{i-1} - Q_iV_i$
$i := i + 1$
fin tantque
$n := i$
renvoyer $R_{n-1} = \operatorname{gcrd}(A,B),\ (U_{n-1}, V_{n-1}) = (U_1, V_1)$ et $(U_n, V_n) = (U_2, V_2)$

Algorithme 4.3: Algorithme d'Euclide étendu à droite

La preuve suivante s'inspire des preuves de [Coh63, BP96].

Démonstration. La terminaison de cet algorithme s'obtient en remarquant que les degrés des polynômes de Ore R_i sont strictement décroissants par rapport à i. Cette décroissance stricte s'explique par le fait que le degré d'un reste de la division entre deux polynômes est toujours strictement plus petit que le degré du diviseur. Il existe donc un i tel que $R_i = 0$.

La correction de cet algorithme se montre en plusieurs étapes. Pour montrer que $R_{n-1} = \operatorname{gcrd}(A,B)$, on montre d'abord que pour tout i compris entre 1 et n :

$$\operatorname{gcrd}(R_{i-1}, R_i) = \operatorname{gcrd}(R_i, R_{i+1}).$$

En décomposant R_{i+1} en $Q_iR_i - R_{i+1}$, on voit que $\operatorname{gcrd}(R_{i-1}, R_i)$ divise R_{i+1} à droite, donc divise $\operatorname{gcrd}(R_i, R_{i+1})$ à droite. En décomposant R_{i-1} en $R_{i-1} + Q_iR_i$, on a pour les mêmes raisons, $\operatorname{gcrd}(R_i, R_{i+1})$ qui divise à droite $\operatorname{gcrd}(R_{i-1}, R_i)$, d'où l'égalité. Si de plus on utilise la propriété $R_n = 0$, on obtient la suite d'égalités suivante qui permet de conclure :

$$\operatorname{gcrd}(A,B) = \cdots = \operatorname{gcrd}(R_{i-1}, R_i) = \operatorname{gcrd}(R_i, R_{i+1}) = \cdots = \operatorname{gcrd}(R_{n-1}, R_n) = R_{n-1}.$$

Pour montrer que les couples (U_{n-1}, V_{n-1}) et (U_n, V_n) sont respectivement les coefficients de Bézout et les cofacteurs du lclm entre A et B, on réexprime les calculs de l'algorithme par des multiplications de matrices. Pour cela, on définit la matrice M_{Q_i} par :

$$M_{Q_i} := \begin{bmatrix} 0 & 1 \\ 1 & -Q_i \end{bmatrix}.$$

On déduit de cette matrice les égalités suivantes :

$$\begin{bmatrix} U_i & V_i \\ U_{i+1} & V_{i+1} \end{bmatrix} = M_{Q_i} M_{Q_{i-1}} \cdots M_{Q_1} \tag{4.2}$$

et

$$\begin{bmatrix} R_i \\ R_{i+1} \end{bmatrix} = \begin{bmatrix} U_i & V_i \\ U_{i+1} & V_{i+1} \end{bmatrix} \begin{bmatrix} R_0 \\ R_1 \end{bmatrix}. \tag{4.3}$$

Grâce à ce calcul matriciel, on a immédiatement, pour tout i compris entre 0 et n, l'égalité :

$$U_i A + V_i B = R_i. \tag{4.4}$$

En utilisant cette propriété pour $i = n - 1$, on a :

$$U_{n-1} A + V_{n-1} B = \operatorname{gcrd}(A, B),$$

ce qui prouve la propriété sur les coefficients de Bézout. En l'utilisant pour $i = n$, on obtient :

$$U_n A + V_n B = 0,$$

$U_n A = -V_n B$ est donc bien un multiple commun à gauche de A et B. Il reste à montrer maintenant que c'est le plus petit. Soient deux opérateurs C_0 et C_1 tels que

$$C_0 A + C_1 B = 0. \tag{4.5}$$

En remarquant que la matrice $\begin{bmatrix} Q_i & 1 \\ 1 & 0 \end{bmatrix}$ est l'inverse de M_{Q_i}, on définit la suite d'opérateurs C_i pour i allant de 2 à n par :

$$\begin{bmatrix} C_i & C_{i+1} \end{bmatrix} := \begin{bmatrix} C_0 & C_1 \end{bmatrix} (M_{Q_i} \cdots M_{Q_1})^{-1}. \tag{4.6}$$

En combinant les équations (4.3), (4.5) et (4.6), on obtient la relation suivante :

$$\begin{bmatrix} C_i & C_{i+1} \end{bmatrix} \cdot \begin{bmatrix} R_i \\ R_{i+1} \end{bmatrix} = 0.$$

En particulier l'égalité est vraie lorsque $i = n - 1$, on a donc dans ce cas $C_{n-1} R_{n-1} = 0$ car $R_n = 0$, ce qui implique $C_{n-1} = 0$ puisque $R_{n-1} = \operatorname{gcrd}(A, B) \neq 0$ et qu'il n'y a pas de diviseur de 0. En utilisant l'égalité $\begin{bmatrix} C_0 & C_1 \end{bmatrix} \begin{bmatrix} U_0 & V_0 \\ U_1 & V_1 \end{bmatrix} = \begin{bmatrix} C_0 & C_1 \end{bmatrix}$ avec les équations (4.2) et (4.6), on obtient :

$$\begin{bmatrix} C_i & C_{i+1} \end{bmatrix} \begin{bmatrix} U_i & V_i \\ U_{i+1} & V_{i+1} \end{bmatrix} = \begin{bmatrix} C_0 & C_1 \end{bmatrix}.$$

Comme $C_{n-1} = 0$, on a $C_0 = C_n U_n$ et $C_1 = C_n V_n$. C_0 et C_1 sont des multiples à gauche de U_n et V_n. Ceci prouve la minimalité de ces opérateurs. □

Remarque 4.8. L'algorithme 4.3 se généralise aux polynômes de Laurent-Ore. En effet si A et B sont des polynômes de Laurent-Ore de valuation r_A et r_B, on note $d = \min(r_A, r_B)$ et l'entrée de l'algorithme est alors la paire d'opérateurs $(S_n^{-d} A, S_n^{-d} B)$.

Remarque 4.9. L'algorithme d'Euclide à gauche s'écrit de manière similaire, on en déduit le p.g.c.d à gauche (gcld) et le p.p.c.m à droite.

On utilisera dans la suite la notation suivante pour les cofacteurs.

Notation 4.10. Pour deux opérateurs A et B, on note par $(B)^A$ et $(A)^B$ les cofacteurs de leur lclm tels que :

$$\operatorname{lclm}(A,B) = (B)^A A = (A)^B B. \tag{4.7}$$

Les quelques lemmes suivants sur le lclm, ses cofacteurs et le gcld seront utiles pour la suite. Le premier lemme est une conséquence directe des définitions.

Lemme 4.11. *Trois opérateurs de récurrence A, B, C quelconques vérifient les égalités suivantes :*

$$\operatorname{gcld}(AB,AC) = A\operatorname{gcld}(B,C), \quad \operatorname{lclm}(BA,CA) = \operatorname{lclm}(B,C)A.$$

et

$$\operatorname{lclm}(A,\operatorname{lclm}(B,C)) = \operatorname{lclm}(\operatorname{lclm}(A,B),C) = \operatorname{lclm}(A,B,C), \tag{4.8}$$

Lemme 4.12 ([Ore33])**.** *Trois opérateurs de récurrence A, B, C quelconques vérifient les égalités suivantes :*

$$(A)^{BC} = \left((A)^C\right)^B, \quad (BC)^A = (B)^{(A)^C}(C)^A. \tag{4.9}$$

Démonstration. $\operatorname{lclm}(A,BC)$ est un multiple gauche de C, on a donc :

$$(A)^{BC}BC = \operatorname{lclm}(A,BC) = \operatorname{lclm}(A,C,BC) = \operatorname{lclm}(\operatorname{lclm}(A,C),BC). \tag{4.10}$$

On a par suite en utilisant plusieurs fois la définition d'un cofacteur et la propriété du lemme 4.11 sur le lclm l'égalité suivante :

$$\operatorname{lclm}(\operatorname{lclm}(A,C),BC) = \operatorname{lclm}((A)^C C,BC) = \operatorname{lclm}((A)^C,B)C = \left((A)^C\right)^B BC.$$

L'anneau des polynômes de Ore n'a pas de diviseur de zéro. L'égalité reste donc vraie en divisant à droite par BC, ce qui prouve l'équation (4.9).

Pour montrer la seconde équation, on utilise d'abord la définition d'un cofacteur pour obtenir :

$$(B)^{(A)^C}(C)^A A = (B)^{(A)^C}(A)^C C = \operatorname{lclm}(BC,(A)^C C).$$

Selon la définition d'un cofacteur de lclm, le membre droit de l'équation précédente est égal à $\operatorname{lclm}(BC,\operatorname{lclm}(A,C))$, qui selon l'équation (4.10) est lui même égal à $\operatorname{lclm}(BC,A)$. On en déduit donc la suite d'égalités suivante :

$$(B)^{(A)^C}(C)^A A = \operatorname{lclm}(BC,A) = (BC)^A A.$$

Encore une fois, la division à droite par A de cette équation permet de conclure. □

Lemme 4.13. *Trois opérateurs de récurrence A, B, C quelconques vérifient l'égalité suivante :*

$$(BA)^{CA} = (B)^{C}. \tag{4.11}$$

Démonstration. Ce lemme se déduit immédiatement de l'équation :

$$(BA)^{CA}CA = \mathrm{lclm}(BA, CA) = \mathrm{lclm}(B, C)A = (B)^{C}CA.$$

□

Lemme 4.14. *Trois opérateurs de récurrence A, B, C quelconques vérifient les égalités suivantes :*

$$(\mathrm{lclm}(A, B))^{C} = \mathrm{lclm}((A)^{C}, (B)^{C}) = (A)^{\mathrm{lclm}(B,C)}(B)^{C} = (\mathrm{lclm}(A, B, C))^{C}. \tag{4.12}$$

Démonstration. En revenant à la définition d'un cofacteur, on a d'une part

$$(\mathrm{lclm}(A, B))^{C}C = \mathrm{lclm}(\mathrm{lclm}(A, B), C) = \mathrm{lclm}(A, B, C)$$

et comme $\mathrm{lclm}(AC, BC) = \mathrm{lclm}(A, B)C$, on a d'autre part

$$\mathrm{lclm}((A)^{C}, (B)^{C})C = \mathrm{lclm}((A)^{C}C, (B)^{C}C) = \mathrm{lclm}(\mathrm{lclm}(A, C), \mathrm{lclm}(B, C)) = \mathrm{lclm}(A, B, C).$$

Il suffit de diviser l'ensemble à droite par C pour conclure sur la validité de la première équation.

Pour démontrer la deuxième égalité, on utilise à trois reprises la définition d'un cofacteur pour avoir la suite d'égalités suivante :

$$(A)^{\mathrm{lclm}(B,C)}(B)^{C}C = (A)^{\mathrm{lclm}(B,C)}\,\mathrm{lclm}(B, C) = \mathrm{lclm}(A, B, C) = (\mathrm{lclm}(A, B))^{C}C.$$

En divisant à droite par C, on obtient la formule souhaitée.

La dernière égalité est un conséquence immédiate des égalités :

$$(\mathrm{lclm}(A, B))^{C}C = \mathrm{lclm}(A, B, C) = \mathrm{lclm}((\mathrm{lclm}(A, B))^{C}C, (C)^{C}C) = (\mathrm{lclm}(A, B, C))^{C}C.$$

□

Lemme 4.15. *Pour tous opérateurs de récurrence A, B et C, l'opérateur*

$$\mathrm{lclm}((C)^{A}, (C)^{B})(A + B)$$

est un multiple à gauche de l'opérateur $\mathrm{lclm}(A + B, C)$.

Démonstration. Il est clair que l'opérateur $\mathrm{lclm}((C)^{A}, (C)^{B})(A + B)$ est un multiple à gauche de $A + B$. En distribuant par rapport à l'addition on a :

$$\mathrm{lclm}((C)^{A}, (C)^{B})(A + B) = \mathrm{lclm}((C)^{A}A, (C)^{B}A) + \mathrm{lclm}((C)^{A}B, (C)^{B}B).$$

En utilisant la définition du lclm, cet opérateur devient

$$\mathrm{lclm}(A, C, (C)^{B}A) + \mathrm{lclm}((C)^{A}B, B, C).$$

Celui-ci est un multiple à gauche de C, ce qui démontre le lemme. □

Remarque 4.16. La somme de deux suites P-recursives est aussi P-recursive. Le lclm permet de le montrer en calculant un opérateur de récurrence qui annule la somme. En effet si les suites u et v sont respectivement annulées par A et B, alors elles sont annulées par des multiples de A et B, on a donc :

$$\mathrm{lclm}(A, B) \cdot (u + v) = 0.$$

Exemple 4.17. En utilisant un lclm, il est facile de prouver l'égalité classique du triangle de Pascal :

$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}.$$

En utilisant la représentation des binomiaux en factoriels, on a $(n+1-k)\binom{n+1}{k} = (n+1)\binom{n}{k}$. La suite des binomiaux est donc annulée par l'opérateur P_k défini par :

$$P_k = (n + 1 - k)S_n - (n + 1).$$

Pour effectuer la somme souhaitée, on effectue donc le lclm entre P_k et P_{k+1}. On obtient de cette façon l'opérateur :

$$\mathrm{lclm}(P_k, P_{k+1}) = (n + 2 - k)(n + 1 - k)S_n^2 + 2(n + 2)(n + 1 - k)S_n + (n + 2)(n + 1).$$

Pour montrer que cette opérateur annule la suite $\binom{n+1}{k+1}$, il suffit d'effectuer la division entre $\mathrm{lclm}(P_k, P_{k+1})$ et l'opérateur

$$(n + 1 - k)S_n - (n + 2),$$

qui annule $\binom{n+1}{k+1}$. La division est exacte, donc $\mathrm{lclm}(P_k, P_{k+1})$ annule la suite $\binom{n+1}{k+1}$.

Pour finir la preuve, il suffit juste de vérifier que l'égalité est vérifiée pour les conditions initiales de la récurrence en faisant attention à ses singularités $k - 1$ et $k - 2$. Si on montre que la propriété est vraie pour $n < k$, $n = k$ et $n = k + 1$, on aura démontré la propriété. Pour $n < k$, on a :

$$\binom{n}{k} = \binom{n}{k+1} = \binom{n+1}{k+1} = 0,$$

donc l'égalité est vérifiée. Pour $n = k$ et $n = k + 1$, on a :

$$\binom{k}{k} + \binom{k}{k+1} = 1 = \binom{k+1}{k+1}, \quad \binom{k+1}{k} + \binom{k+1}{k+1} = k + 2 = \binom{k+2}{k+2}.$$

Les conditions initiales sont bien vérifiées, on a donc bien prouvé l'égalité du triangle de Pascal.

Remarque 4.18. Si la suite u est annulée par deux opérateurs de récurrence A et B, les deux opérateurs U et V qui sont les coefficients de Bézout du gcrd entre A et B nous donnent

$$\mathrm{gcrd}(A, B) \cdot u = UA \cdot u + VB \cdot u = 0.$$

Le gcrd fournit alors une récurrence d'ordre moindre.

Exemple 4.19. Le développement en série de Tchebychev de la fonction xe^x est :

$$xe^x := \sum_{n\in\mathbb{Z}} \frac{\mathrm{I}_{n+1}(1) + \mathrm{I}_{n-1}(1)}{2} T_n(x),$$

où $\mathrm{I}_n(1)$ est la $n^{\text{ème}}$ fonction de Bessel modifiée de première espèce évaluée en 1. Dans le chapitre 5, un algorithme pour calculer les récurrences satisfaites par les coefficients d'une série de Tchebychev solution d'une équation différentielle est donné. Cet algorithme prend en entrée une équation différentielle et donne en sortie une récurrence linéaire. Pour la fonction xe^x, on peut utiliser l'algorithme avec l'équation différentielle $xy'(x) - (x+1)y(x) = 0$. On obtient en sortie la récurrence :

$$-u_n + 2nu_{n+1} + (2n+8)\,u_{n+3} + u_{n+4} = 0.$$

On sait par ailleurs qu'il existe une récurrence d'ordre 2, déduite de la récurrence satisfaite par la suite des fonctions de Bessel modifiées, qui annule cette suite. Cette récurrence n'est donc pas d'ordre minimal. Une autre équation différentielle annule xe^x, il s'agit de l'équation à coefficients constants et d'ordre 2, $y''(x) - 2y'(x) + y(x) = 0$. En utilisant le même algorithme avec cette équation différentielle, on obtient encore une récurrence d'ordre 4

$$\begin{aligned}(n+3)\,u_n - \left(4n^2+16n+12\right)u_{n+1} + \left(4n^3+24n^2+42n+20\right)u_{n+2}\\ + \left(4n^2+16n+12\right)u_{n+3} + (n+1)\,u_{n+4} = 0.\end{aligned}$$

Le gcrd des deux récurrences est :

$$(n^2+3n+3)u_n - 2(n^3+3n^2+3n+1)u_{n+1} + (n^2+n+1)u_{n+2} = 0.$$

La remarque précédente nous dit que cette récurrence annule bien la suite $\mathrm{I}_{n+1}(1) + \mathrm{I}_{n-1}(1)$. On a donc obtenu de cette façon une récurrence d'ordre plus petit.

1.2 Multiplication rapide d'opérateurs de récurrence

Cette section rappelle les principaux résultats de complexité pour les algorithmes de multiplication d'opérateurs de récurrence, de calcul du lclm et de calcul du gcld. L'entrée de chaque algorithme est composée d'opérateurs de récurrence à coefficients polynomiaux en n. Comme expliqué dans la section 2.2, le modèle de complexité utilisé dans cette section est le nombre d'opérations arithmétiques effectuées. La taille d'un objet est donc le nombre d'éléments sur le corps $\mathbb{K}$ qu'il comporte. Par exemple si $A \in \mathbb{K}[n]\langle S_n\rangle$, alors sa taille sera bornée par le maximum des degrés en n de ses coefficients plus 1, multiplié par l'ordre de l'opérateur en S_n plus 1.

Cette section présente les algorithmes de multiplication entre polynômes de Ore. La complexité de l'algorithme naïf 4.1 est donnée avant d'étudier un algorithme rapide dû

à van der Hoeven [vdH02]. Un des résultats importants du chapitre 5 repose sur cette multiplication rapide.

Du point de vue de la complexité, l'opération la plus étudiée dans un anneau est souvent la multiplication. Contrairement à l'addition, cette opération n'a pas une complexité linéaire. L'existence d'algorithmes rapides pour calculer celle-ci est donc importante. On peut penser à la multiplication rapide d'entiers, la multiplication rapide de polynômes ou la multiplication rapide de matrices. Les algorithmes de multiplications rapides d'opérateurs différentiels ou d'opérateurs de récurrence sont assez récents. En 2002, van der Hoeven [vdH02] a donné un algorithme rapide de multiplication de polynômes non-commutatifs en ramenant cette multiplication au produit de deux matrices. Cet algorithme a été amélioré par Bostan, Chyzak et Le Roux [BCLR08]. Plus récemment Bostan, van der Hoeven et moi [BBvdH12] (voir annexe B) avons amélioré cet algorithme dans le cas où les degrés des polynômes sont grands (ou petits) devant les ordres des opérateurs.

Récemment van der Hoeven [vdH11] a donné des algorithmes rapides pour le calcul du lclm et le gcld entre opérateurs différentiels en basant la complexité de ces algorithmes sur la complexité du produit de deux opérateurs différentiels. Dans un article à paraître prochainement [BCLS12], les auteurs donnent d'autres algorithmes de calcul de lclm basés sur la multiplication polynomiale rapide.

Complexité de l'algorithme naïf

Avant d'étudier un algorithme rapide de multiplication, il est pertinent d'étudier la complexité de l'algorithme naïf 4.1 afin de pouvoir les comparer.

Dans cet algorithme l'opération de décalage d'un polynôme est effectuée plusieurs fois. Pour un polynôme de degré d, un algorithme naïf effectue cette opération de décalage en $\mathcal{O}(d^2)$ opérations arithmétiques. Il existe des algorithmes rapides qui effectuent cette opération en $\mathsf{M}(d) + \mathcal{O}(d)$ opérations arithmétiques. Un de ces algorithmes est expliqué dans le chapitre 3 (algorithme 3.2 page 35).

Si l'algorithme 4.1 prend en entrée deux polynômes A et B dans $\mathbb{K}[n]\langle S_n\rangle$ chacun de bidegré (r, d) en S_n et n, en utilisant les algorithmes naïfs de multiplication et de décalage d'un polynôme, celui-ci s'effectue en $\mathcal{O}(r^2d^2)$ opérations arithmétiques. En effet, à chaque passage dans la boucle, au plus $2d$ produits de polynômes de degré d et $2r$ opérations de décalage de polynôme de degré d sont effectuées. La multiplication naïve de deux polynômes de degré d et l'opération de décalage de polynôme s'effectuent toutes les deux en $\mathcal{O}(d^2)$ opérations arithmétiques, ce qui explique la complexité annoncée. La complexité de cet algorithme est ici la même qu'avec la multiplication naïve de polynômes bivariés dans des anneaux commutatifs.

En utilisant des algorithmes rapides pour la multiplication de polynômes et pour le décalage d'un polynôme, on peut descendre la complexité de cet algorithme de multiplication tordue à $\mathcal{O}(r^2\mathsf{M}(d))$ opérations arithmétiques.

La sortie de cet algorithme est le polynôme BA qui est de bidegré $(2r, 2d)$, on peut donc minorer la complexité d'un algorithme de multiplication à $\mathcal{O}(rd)$ opérations arithmétiques.

Entrée: $A = \sum_{i=0}^{r_A} a_i(n)S_n^i$ et $B = \sum_{i=0}^{r_B} b_i(n)S_n^i$, où $a_i \in \mathbb{K}_{d_A}[n]$ et $b_i \in \mathbb{K}_{d_B}[n]$
Sortie: $C = BA = \sum_{i=0}^{d_C} c_i(n)S_n^i$ où $c_i \in \mathbb{K}_{r_C}[n]$
1: Calculer les matrices $M^A_{r_C+r_B}(0)$ et $M^B_{r_C}(0)$
2: Calculer la matrice $M^C_{r_C} = M^B_{r_C}(0)M^A_{r_C+r_B}(0)$
3: **renvoyer** C en interpolant les valeurs de $M^C_{r_C}$

Algorithme 4.4: Multiplication rapide de polynômes tordus de van der Hoeven

Ce qui laisse espérer une meilleure complexité pour le produit.

Multiplication rapide d'opérateurs de récurrence

Cette section présente un algorithme dû à van der Hoeven [vdH02] pour multiplier rapidement deux opérateurs de récurrence A et B d'ordres respectifs r_A et r_B. C'est la première fois que l'algorithme de van der Hoeven est explicité pour la multiplication d'opérateurs de récurrence.

L'idée de l'algorithme est que l'on peut effectuer la multiplication de deux opérateurs de récurrence en évaluant et interpolant ces opérateurs sur des suites. Je vais présenter cet algorithme d'un autre point de vue, je vais montrer que la multiplication d'opérateurs se ramène à de la multiplication de matrices polynomiales. Cette multiplication sera effectuée par évaluation et interpolation des polynômes, ce qui se fera en bonne complexité, les matrices étant structurées.

On représente un opérateur $A = \sum_{i=0}^{r_A} a_i(n)S_n^i$ par la matrice $M_k^A(n)$ définie par :

$$M_k^A(n) = \begin{pmatrix} a_0(n) & \cdots & a_{r_A}(n) & & \\ & a_0(n+1) & \cdots & a_{r_A}(n+1) & \\ & & \ddots & & \ddots \\ & & & a_0(n+k) & \cdots & a_{r_A}(n+k) \end{pmatrix},$$

La produit AB s'exprime simplement comme $\sum_{i=0}^{r_A} a_i(n)S_n^i B$, pour tout k, on a donc

$$M_k^{AB}(n) = M_k^A(n)M_{k+r_A}^B(n). \tag{4.13}$$

On remarque qu'en prenant k assez grand, on peut retrouver $M_k^{AB}(n)$ à partir de $M_k^{AB}(0)$ (où d'un autre point) par interpolation. Par la formule de multiplication, on sait que les coefficients de C sont de degré inférieur à $d_A + d_B$, il suffit donc de prendre k plus grand que cette borne.

Théorème 4.20 ([vdH02])**.** *La multiplication de deux opérateurs de récurrence A et B de degré respectifs d_A et d_B en n et r_A et r_B en S_n s'effectue en $\mathcal{O}((d_A + d_B + r_A + r_B)^\omega)$ opérations arithmétiques avec l'algorithme 4.4.*

Démonstration. Le passage d'un opérateur A à la matrice $M^A_{d_A}(0)$ s'effectue par évaluation, action qui est effectuée à l'étape 1 de cet algorithme. Les coefficients polynomiaux de

l'opérateur C étant bornés par r_C, on retrouve $M^C_{r_C}(n)$ par interpolation des coefficients de la matrice $M^C_{r_C}(0)$, action qui est effectuée à l'étape 3. Les étapes 1 et 3 sont donc justifiées. L'équation 4.13 nous montre l'égalité qui prouve l'étape 2.

Il est effectué à l'étape 2 un produit entre une matrice de $r_C + 1$ lignes et $r_C + r_B + 1$ colonnes et une matrice de $r_C + r_B + 1$ lignes et $r_C + d_C + 1$ colonnes. En utilisant la notation ω de la section 2.2, cette étape s'effectue $\mathcal{O}((r_C + d_C)^\omega)$ opérations arithmétiques. Les coûts des étapes 1 et 3 sont donnés par le lemme 4.21, ce qui conclut la preuve. □

Lemme 4.21. *Soit A un opérateur de récurrence linéaire à coefficients polynomiaux de degrés d_A en n et r_A en S_n.*

- *Le calcul de $M^A_k(0)$ à partir de A s'effectue en $\mathcal{O}((k + r_A + d_A)^\omega)$ opérations arithmétiques.*
- *Le calcul de A à partir de $M^A_{d_A}(0)$ peut s'effectuer en $\mathcal{O}((d_A + r_A)^\omega)$.*

Démonstration. Le calcul de $M^A_{d_A}(0)$ se ramène au produit d'une matrice de Vandermonde par la matrice des coefficients de l'opérateur. Si on écrit les polynômes $a_i(n) = \sum_{j=0}^{d_A} a_{i,j} n^j$, on obtient les évaluations des polynômes a_i en les points souhaités grâce au produit

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & 2^{d_A} \\ \vdots & & & \vdots \\ 1 & k & \cdots & k^{d_A} \end{pmatrix} \cdot \begin{pmatrix} a_{0,0} & \cdots & a_{r_A,0} \\ a_{0,1} & & a_{r_A,1} \\ \vdots & & \vdots \\ a_{0,d_A} & \cdots & a_{r_A,d_A} \end{pmatrix}.$$

On peut remplir la matrice de Vandermonde recursivement en initialisant les coefficients de la première colonne à 1 et en utilisant la récurrence $u_{i,j} = (i-1)u_{i,j-1}$, où $u_{i,j}$ est le coefficient de la $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne. De cette façon on calcule cette matrice en $(k+1)d_A$ opérations arithmétiques. La première complexité de ce lemme s'en déduit immédiatement, le calcul le plus coûteux étant la multiplication de matrices.

L'interpolation peut s'effectuer en inversant d'abord la matrice carrée de Vandermonde de dimension $d_A + 1$ (on a $k = d_A$) puis en multipliant cette matrice à gauche par la matrice

$$\begin{pmatrix} a_0(0) & a_1(0) & \cdots & a_{r_A}(0) \\ a_0(1) & a_1(1) & \cdots & a_{r_A}(1) \\ \vdots & & & \vdots \\ a_0(d_A) & a_1(d_A) & \cdots & a_{r_A}(d_A) \end{pmatrix}.$$

Une matrice de dimension $d_A + 1$ s'inverse en $\mathcal{O}(d_A^\omega)$ opérations arithmétiques [BH74]. Le coût de cette opération se réduit donc au coût de la multiplication des matrices, opération qui s'effectue en $\mathcal{O}((d_A + r_A)^\omega)$ opérations arithmétiques. □

On peut fixer $d_A = d_B = r_A = r_B = k$ pour comparer cet algorithme avec l'algorithme naïf. La complexité de cet algorithme est alors $\mathcal{O}(k^\omega)$ opérations arithmétiques contre $\mathcal{O}(k^2\mathsf{M}(k))$ opérations arithmétiques pour l'algorithme dit naïf. Même en utilisant une

multiplication naïve de matrices (c'est-à-dire $\omega = 3$), l'algorithme de van der Hoeven est asymptotiquement plus rapide que l'algorithme naïf.

Remarque 4.22. Des algorithmes rapides d'évaluations multipoints et d'interpolation [vzGG03] permettent de calculer la matrice $M^A_{r_C+r_B}$ en $\mathcal{O}(r_A\mathsf{M}(r_C+r_B)\log(r_C+r_B))$ opérations arithmétiques, la matrice $M^B_{r_C}$ en $\mathcal{O}(r_B\mathsf{M}(r_C)\log(r_C))$ opérations arithmétiques et l'opérateur C à partir de la matrice $M^C_{r_C}$ en $\mathcal{O}(d_C\mathsf{M}(r_C)\log(r_C))$ opérations arithmétiques. En utilisant ces algorithmes avec une FFT pour la multiplication, les étapes 1 et 3 ne sont plus les opérations dominantes de l'algorithme 4.4. Seule l'étape 2 est dominante, la complexité de l'algorithme 4.4 repose donc sur cette étape.

Remarque 4.23. Bostan, Chyzak et Le Roux [BCLR08] montrent que l'on peut multiplier deux opérateurs de récurrence en $8k^\omega$ opérations arithmétiques lorsque les opérateurs ont des degrés k en n et k en S_n. Ils montrent aussi que l'on peut ramener la multiplication de deux matrices de taille $n\times n$ à la multiplication de deux opérateurs de récurrence de taille $n\times n$, et donc que ces problèmes sont équivalents. Trouver un algorithme de multiplication d'opérateurs plus rapide, revient donc à diminuer la valeur de ω, ce qui est un problème classique et difficile de complexité.

Un petit mot sur l'algorithme de division

La proposition suivante donne la complexité de l'algorithme de base pour la division entre opérateurs.

Proposition 4.24. *Si l'entrée de l'algorithme 4.2 page 57 est un couple d'opérateurs (A,B) dans $\mathbb{K}[n]\langle S\rangle$ de bidegré (d_A,r_A) pour A et (d_B,r_B) pour B, alors le nombre d'opérations arithmétiques pour calculer la division est de $\mathcal{O}(d_A^3\sup(r_A,r_B))$.*

Démonstration. 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 d_B+i et Q d'ordre d_A-d_B. Les opérateurs Q et R sont à coefficients des fractions rationnelles en n dont le numérateur et le dénominateur sont de même degré, ces degrés sont majorés par $(d_A-d_B-i)\sup(r_B,r_A)$. 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 $\mathrm{coeff}(R,i+dB)S^i\frac{1}{\mathrm{coeff}(B,dB)}$ par le numérateur de Q et de R. Pour R, on doit multiplier d_B+i polynômes de degré $(d_A-d_B-i)r_B$ entre eux. On doit l'effectuer d_A-d_B fois, le nombre d'opérations à effectuer est donc un $\mathcal{O}\left((d_A-d_B)^2\sup(r_A,r_B)d_A\right)$. La complexité de la proposition s'en déduit. □

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

Dans un article récent van der Hoeven [vdH11] donne un algorithme plus efficace dans le cas où les opérateurs sont en $\theta = x\partial_x$. En utilisant cet algorithme avec un algorithme rapide de multiplication d'opérateurs dans le cas déséquilibré [BBvdH12], on effectue la division

euclidienne entre deux opérateurs en $\tilde{\mathcal{O}}(k^{\omega+1})$ opérations. En adaptant son algorithme à des opérateurs de récurrence, on peut espérer la même complexité. On aurait alors un algorithme quasi-optimal.

2 Fractions de polynômes de Ore

De façon similaire à la construction d'un corps de fractions rationnelles à partir d'un anneau de polynômes commutatifs, Oystein Ore [Ore31] construit le corps des fractions de polynômes de Ore à partir de l'anneau des polynômes de Ore. Ces fractions sont utilisées par exemple lors d'études de fonction de transfert [Hal08, HK07], où on est amené à résoudre des systèmes linéaires dans des corps non-commutatifs. À ma connaissance, le travail de cette thèse utilise pour la première fois des fractions d'opérateurs pour calculer des récurrences satisfaites par certaines suites P-recursives.

Cette section donne une construction du corps des fractions des polynômes de Ore, en utilisant l'ensemble des paires de polynômes de Ore muni de deux lois internes puis en quotientant cet ensemble par une classe d'équivalence. L'article original de Ore ne traite pas de ces opérations dans l'ensemble des paires d'opérateurs mais traite directement des opérations dans le corps des fractions d'opérateurs de Ore. Cette construction a été choisie car dans la suite de cette thèse les paires d'opérateurs de Ore sont utilisées en tant que telles (c'est-à-dire en dehors des fractions). La construction de Ore et celle présentée dans cette section sont équivalentes. Le module Maple OreField qui a été développé lors de cette thèse permet de calculer avec les fractions d'opérateurs de récurrence. Ce module surcharge les opérateurs arithmétiques usuels pour les fractions d'opérateurs. Les lignes qui suivent montrent la façon d'initialiser ce module.

```
> with(OreField);
```

$$[\text{`*`}, \text{`+`}, \text{`-`}, \text{`.`}, \text{`^`}, adjointOfFrac, initOreField, normal]$$

```
> initOreField(n);
```

$$UnivariateOreRing\,(n, shift)$$

Ce package utilise les procédures du package OreTools [ALL03] pour le calcul du lclm et du gcld.

2.1 L'ensemble des paires de polynômes de Ore

Il est classique d'exprimer des identités entre deux suites en utilisant un opérateur de récurrence.

Exemple 4.25. Les suites de polynômes de Laguerre $L_n^{(\alpha)}$ et $L_n^{(\alpha+1)}$ vérifient l'égalité suivante [Nat10, http://dlmf.nist.gov/18.9.E13] :

$$L_n^{(\alpha)} = L_n^{(\alpha+1)} - L_{n-1}^{(\alpha+1)},$$

qui se traduit en terme d'opérateurs en :

$$L_{n+1}^{(\alpha)} \quad = \quad (S_n - 1) \cdot L_n^{(\alpha+1)}.$$

Exemple 4.26. Les suites des polynômes de Jacobi $P_n^{(\alpha,\beta)}$ et $P_n^{(\alpha,\beta+1)}$ vérifient l'égalité suivante [Nat10, http://dlmf.nist.gov/18.9.E5] :

$$(2n+\alpha+\beta+1)P_n^{(\alpha,\beta)} = (n+\alpha+\beta+1)P_n^{(\alpha,\beta+1)} + (n+\alpha)P_{n-1}^{(\alpha,\beta+1)}.$$

qui se traduit en terme d'opérateur en :

$$P_{n+1}^{(\alpha,\beta)} \quad = \quad \left(\frac{n+\alpha+\beta+2}{2n+\alpha+\beta+3}S_n + \frac{n+\alpha+1}{2n+\alpha+\beta+3}\right) \cdot P_n^{(\alpha,\beta+1)}.$$

Ce qui est moins classique, c'est de représenter une identité entre deux suites en s'aidant d'une paire d'opérateurs de récurrence.

Exemple 4.27. La suite de fonctions $\psi_n(x) = {}_2F_1\left(\begin{smallmatrix} n,2 \\ 1 \end{smallmatrix} \middle| x\right)$ vérifie :

$$n\psi_n = ((2n+1) - (n-1)x)\psi_{n+1}(x) - (n+1)(1-x)\psi_{n+2}(x).$$

On en déduit la relation entre les suites ψ_n et $x\psi_n$:

$$\left(-n + (2n+1)S_n - (n+1)S_n^2\right) \cdot \psi_n(x) = \left((n-1)S_n - (n+1)S_n^2\right) \cdot x\psi_n(x).$$

De même, les suites de fonctions $\psi_n(x)$ et $\psi_n'(x)$ vérifient l'égalité suivante :

$$(n-1)\psi_{n+1}(x) - (n+1)\psi_{n+2}(x) = \psi_n'(x) - \psi_{n+2}'(x),$$

qui se traduit en terme d'opérateurs en :

$$\left((n-1)S_n - (n+1)S_n^2\right) \cdot \psi_n(x) = (1 - S_n) \cdot \psi_n'(x).$$

On définit ces deux paires dans le module OreField de la façon suivante :

```
> X_1 := [-n,(2*n+1),-n-1]: X_2 := [0, n-1, -n-1]:
> X := OreFrac(X_1, X_2);
```

$$X := [-n + (2n+1)\,Sn + (-n-1)\,Sn^2, (n+1)\,Sn + (-n-1)\,Sn^2]$$

```
> Dx_1 := [0, n-1, -n-1]: Dx_2 := [1, -1]:
> Dx := OreFrac(Dx_1, Dx_2);
```

$$Dx := [(n-1)\,Sn + (-n-1)\,Sn^2, 1 - Sn]$$

Cet exemple utilise deux paires d'opérateurs. C'est l'ensemble de ces paires qui est étudié dans cette section et permettra par la suite de construire le corps des fractions de polynômes de Ore. On peut définir sur cet ensemble deux lois internes, une loi d'addition et une loi de multiplication.

Les lois d'addition et de multiplication entre paires reposent sur le lclm entre deux opérateurs et notamment sur les cofacteurs du lclm. On reprend la notation d'un cofacteur définit par l'équation (4.7) page 60.

Définition 4.28. L'addition entre deux paires d'opérateurs est définie par :

$$(A_1, A_2) + (B_1, B_2) = \Big((B_2)^{A_2} A_1 + (A_2)^{B_2} B_1, \operatorname{lclm}(A_2, B_2)\Big). \tag{4.14}$$

Elle est calculée par l'algorithme 4.5.

Entrée: deux paires (A_1, A_2) et (B_1, B_2)
Sortie: $(A_1, A_2) + (B_1, B_2)$
Calculer $(A_2)^{B_2}$ et $(B_2)^{A_2}$ tel que $\operatorname{lclm}(A_2, B_2) = (A_2)^{B_2} B_2 = (B_2)^{A_2} A_2$ en utilisant l'algorithme 4.3
renvoyer $\Big((B_2)^{A_2} A_1 + (A_2)^{B_2} B_1, (A_2)^{B_2} B_2\Big)$

Algorithme 4.5: Addition de deux paires d'opérateurs

La proposition suivante interprète cette addition sur des suites.

Proposition 4.29. *Si les suites u et v sont liées par $A_1 \cdot u = A_2 \cdot v$ et les suites u et w sont liées par $B_1 \cdot u = B_2 \cdot w$ où (A_1, A_2) et (B_1, B_2) sont deux paires d'opérateurs de récurrence, alors les suites u et $v + w$ sont liées par la relation :*

$$C_1 \cdot u = C_2 \cdot (v + w),$$

où $(C_1, C_2) = (A_1, A_2) + (B_1, B_2)$.

Cette proposition généralise la remarque 4.16 page 62 sur l'addition de deux suites lorsque celles-ci sont annulées par des opérateurs. En effet si $A_2 \cdot v = 0$ et $B_2 \cdot w = 0$, le second élément C_2 de la paire $(1, A_2) + (1, B_2)$ permet de retrouver l'égalité :

$$C_2 \cdot (v + w) = 0.$$

Démonstration. L'équation (4.14) entraîne :

$$C_1 \cdot u = (B_2)^{A_2} A_1 \cdot u + (A_2)^{B_2} B_1 \cdot u.$$

Comme $A_1 \cdot u = A_2 \cdot v$ et $B_1 \cdot u = B_2 \cdot v$, le membre droit de cette équation se réécrit :

$$\operatorname{lclm}(A_2, B_2) \cdot v + \operatorname{lclm}(A_2, B_2) \cdot w.$$

D'après (4.14) encore, il s'agit de $C_2 \cdot (v + w)$. □

Exemple 4.30. L'addition est surchargée dans le package OreField. En reprenant les notations de l'exemple 4.27, voici comment établir une relation entre les fonctions $\psi_n(x)$ et $x\psi_n(x) + \psi_n'(x)$ en utilisant la propriété d'addition de deux paires.

```
> X+Dx;
```

$$[-n + (3n+2)\,Sn + \left(-4n - 4 + n^2\right) Sn^2 - \left(n + 2\,(2n-1)\,Sn^3\right) + (n+3)\,(n+2)\,Sn^4,$$
$$(n-1)\,Sn + (-2n-1)\,Sn^2 + (n+2)\,Sn^3]$$

Définition 4.31. La multiplication de deux paires est définie par :

$$(B_1, B_2)\,(A_1, A_2) = \left((B_1)^{A_2} A_1, (A_2)^{B_1} B_2\right). \tag{4.15}$$

Elle est calculée par l'algorithme suivant 4.6.

Entrée: deux paires (A_1, A_2) et (B_1, B_2)
Sortie: $(B_1, B_2)(A_1, A_2)$
Calculer $(A_2)^{B_1}$ et $(B_1)^{A_2}$ tel que $\mathrm{lclm}(A_2, B_1) = (A_2)^{B_1} B_1 = (B_1)^{A_2} A_2$ en utilisant l'algorithme 4.3
renvoyer $\left((B_1)^{A_2} A_1, (A_2)^{B_2} B_2\right)$

Algorithme 4.6: Multiplication de deux paires d'opérateurs

La multiplication est surchargée dans le package OreField.
De cette définition, on déduit la proposition suivante :

Proposition 4.32. *Soient deux paires d'opérateurs de récurrence (A_1, A_2) et (B_1, B_2), et soient trois suites u, v et w telles que $A_1 \cdot u = A_2 \cdot v$ et $B_1 \cdot v = B_2 \cdot w$. Les suites u et w sont reliées par l'égalité suivante :*

$$C_1 \cdot u = C_2 \cdot w,$$

où $(C_1, C_2) = (B_1, B_2)(A_1, A_2)$.

Si les seconds éléments des paires sont égaux à 1, on retrouve le résultat connu de composition d'opérateurs. C'est-à-dire si $A \cdot u = v$ et $B \cdot v = w$, on a :

$$BA \cdot u = w.$$

Démonstration. Par la définition de la multiplication de paires, on a :

$$C_1 \cdot u = (B_1)^{A_2} A_1 \cdot u \quad \text{et} \quad C_2 \cdot w = (A_2)^{B_1} B_2 \cdot w.$$

Comme $A_1 \cdot u = A_2 \cdot v$ et $B_1 \cdot v = B_2 \cdot w$, les membres droits se réécrivent en :

$$C_1 \cdot u = (B_1)^{A_2} A_2 \cdot v \quad \text{et} \quad C_2 \cdot w = (A_2)^{B_1} B_1 \cdot v.$$

Revenant à la définition des cofacteurs, on a

$$(B_1)^{A_2} A_2 = (A_2)^{B_1} B_1,$$

ce qui permet de conclure. □

Remarque 4.33. Tout élément λ de $\mathbb{K}$ étant identifié à la paire $(\lambda, 1)$, la multiplication ou l'addition d'une paire par un scalaire s'en déduit immédiatement. Le code Maple suivant montre comment on peut effectuer l'addition ou la multiplication d'une paire avec la constante 3.

```
> OreFrac([1], [1,n])+3;
```

$$[4 + 3nSn, 1 + nSn]$$

```
> 3*OreFrac([1], [1,n]);
```

$$[3, 1 + nSn]$$

Exemple 4.34. Les polynômes de Tchebychev $T_n(x)$ satisfont les deux égalités classiques suivantes [AS64, Chapitre 22] :

$$T_{n+1}(x) - 2xT_n(x) + T_{n-1}(x) = 0, \tag{4.16}$$

et

$$2(1 - x^2)T'_n(x) = -nT_{n+1}(x) + nT_{n-1}(x). \tag{4.17}$$

Une égalité moins classique satisfaite par ces polynômes est la suivante :

$$T_n(x) = \frac{1}{2}\left(\frac{T'_{n+1}}{n+1} - \frac{T'_{n-1}}{n-1}\right). \tag{4.18}$$

En utilisant les propositions 4.29 et 4.32 et le package OreField, il est facile de déduire la troisième égalité des deux premières.

L'équation (4.16) donne la relation :

$$(1 + S_n^2) \cdot T_n = 2S_n \cdot xT_n,$$

qui fournit la paire X.

```
> X := OreFrac([1,0,1],[0,2]);
```

$$X := [1 + Sn^2, 2Sn]$$

De la même façon l'équation (4.17) donne la relation :

$$2S_n \cdot T_n = ((n+1) - (n+1)S_n^2) \cdot (1 - x^2)T'_n,$$

qui fournit la paire XDx qui représente l'opérateur $(1 - x^2)\partial_x$:

```
> XDx := OreFrac([n+1,0, -n-1],[0,2]);
```

$$XDx := [n + 1 + (-n - 1)\, Sn^2, 2Sn]$$

En utilisant les propriétés d'addition et de multiplication, l'opération $XDx(1-X^2)^{-1}$ permet de retrouver la paire d'opérateurs associée à l'opérateur de dérivation que l'on a grâce à l'équation (4.18).

```
> XDx.(1-X^2)^(-1);
```

$$[-2\,(n+1)\,(n+3)\,Sn^2, (n+3)\,Sn + (-n-1)\,Sn^3]$$

L'opération `(1-X)^2^(-1)` consiste juste à échanger le numérateur avec le dénominateur.

En multipliant chaque élément de la paire par S_n^{-2}, cette paire est bien la représentante de l'égalité ci-dessus que l'on peut définir en Maple par :

```
> Dx := OreFrac([0,-2*n*(n+2)],[n+2, 0, -n]);
```

$$Dx := [-2n\,(n+2)\,Sn, n+2-nSn^2]$$

À ce stade, on n'a pas encore défini une fonction pour simplifier directement une fraction. Celle-ci nous aurait évité de multiplier par S_n^{-2} *à la main*. On verra dans la suite qu'on aurait pu simplifier directement cette fraction par la procédure normal.

Cet exemple illustre une propriété intéressante et fondamentale pour la suite de cette thèse. Si on a une relation entre une famille de fonctions f_n et la famille de ses dérivées f_n' et une autre relation entre la famille f_n et la famille xf_n, pour tout opérateur différentiel L on sait calculer, grâce aux opérations d'addition et de multiplication, une relation entre la famille f_n et la famille $L \cdot f_n$. Cette propriété sera développée dans le chapitre 8

Exemple 4.35. Les polynômes de Tchebychev sont solutions de l'équation différentielle :

$$(1-x^2)T_n'' - xT_n' + n^2T_n(x) = 0.$$

On peut retrouver cette égalité à partir de la définition de X et Dx vue dans l'exemple précédent en utilisant Maple de la façon suivante :

```
> (Dx^2).(1-X^2)-Dx.X+OreFrac([n^2],[1]);
```

$$[0, n^2+7n+12-2n\,(n+4)\,Sn^2+n\,(n+1)\,Sn^4]$$

En utilisant les propositions 4.29 et 4.32, on sait que le premier élément de cette paire appliqué à T_n est égal au second appliqué à $L \cdot T_n$. On déduit de cette constatation la relation suivante :

$$\left(n^2+7n+12-2n\,(n+4)\,Sn^2+n\,(n+1)\,Sn^4\right)\cdot(L\cdot T_n(x)) = 0, \tag{4.19}$$

avec $L := (1-x^2)\partial_x^2 - x\partial_x + n^2$. Les polynômes de Tchebychev forment une suite de polynômes de degré en x strictement croissant par rapport à l'indice n. La famille $L \cdot T_n(x)$ est donc une famille de polynômes nuls ou de degré strictement croissant par rapport à l'indice n quand celui-ci est plus grand qu'un certain entier. On en déduit que si la suite $L \cdot T_n(x)$ est non nulle, celle-ci ne peut être annulée par un opérateur de récurrence à coefficients des fractions rationnelles en l'indéterminé n. L'équation (4.19) implique donc que $L \cdot T_n(x) = 0$.

La proposition suivante donne des propriétés importantes sur les lois d'addition et de multiplication entre paires de polynômes de Ore.

Proposition 4.36. *L'addition de deux paires de polynômes de Ore est une opération associative et commutative. La multiplication de deux paires de polynômes de Ore est une opération associative et distributive à gauche par rapport à l'addition.*

Preuve de la proposition 4.36. La propriété (4.8) donne l'associativité de l'addition, en effet si on effectue l'addition de la façon suivante, on obtient :

$$(A_1, A_2) + ((B_1, B_2) + (C_1, C_2)) = (A_1, A_2) + \Big((B_2)^{C_2} C_1 + (C_2)^{B_2} B_1, \operatorname{lclm}(B_2, C_2)\Big),$$

qui donne avec (4.8) :

$$\Big((\operatorname{lclm}(B_2, C_2))^{A_2} A_1 + (A_2)^{\operatorname{lclm}(B_2,C_2)} \Big((B_2)^{C_2} C_1 + (C_2)^{B_2} B_1\Big), \operatorname{lclm}(A_2, B_2, C_2)\Big).$$

En utilisant l'égalité (4.9) page 60, on a :

$$(A_2)^{\operatorname{lclm}(B_2,C_2)} = (A_2)^{(B_2)^{C_2} C_2} = \Big((A_2)^{C_2}\Big)^{(B_2)^{C_2}}.$$

On a donc :

$$(A_2)^{\operatorname{lclm}(B_2,C_2)} (B_2)^{C_2} = \Big((A_2)^{C_2}\Big)^{(B_2)^{C_2}} (B_2)^{C_2} = \operatorname{lclm}((A_2)^{C_2}, (B_2)^{C_2}) = (\operatorname{lclm}(A_2, B_2))^{C_2}.$$

De la même façon on a :

$$(A_2)^{\operatorname{lclm}(B_2,C_2)} (B_2)^{C_2} = (\operatorname{lclm}(A_2, C_2))^{B_2},$$

et donc

$$(A_1, A_2) + ((B_1, B_2) + (C_1, C_2)) = \Big((\operatorname{lclm}(B_2, C_2))^{A_2} A_1 + (\operatorname{lclm}(A_2, C_2))^{B_2} B_1 + (\operatorname{lclm}(A_2, B_2))^{C_2} C_1, \operatorname{lclm}(A_2, B_2, C_2)\Big).$$

Avec les mêmes arguments, on obtient la même paire quand on effectue les additions $((A_1, A_2) + (B_1, B_2)) + (C_1, C_2)$ et $((A_1, A_2) + (C_1, C_2)) + (B_1, B_2)$.

L'associativité de la multiplication est moins évidente. Elle repose sur le lemme 4.12 page 60 : Pour démontrer l'associativité, on effectue dans un premier temps le produit :

$$(C_1, C_2)((B_1, B_2)(A_1, A_2)).$$

On multiplie donc d'abord (B_1, B_2) par (A_1, A_2), on obtient en utilisant la règle de multiplication (4.15) :

$$(B_1, B_2)(A_1, A_2) = \Big((B_1)^{A_2} A_1, (A_2)^{B_1} B_2\Big).$$

Toujours en utilisant la même règle, on multiplie cette paire à droite avec (C_1, C_2) :

$$(C_1, C_2)\,((B_1, B_2)(A_1, A_2)) = \Big((C_1)^{((A_2)^{B_1} B_2)} (B_1)^{A_2} A_1,\ \Big((A_2)^{B_1} B_2\Big)^{C_1} C_2\Big).$$

On effectue dans un second temps le produit $((C_1, C_2)(B_1, B_2))(A_1, A_2)$. En utilisant encore les règles de multiplication, on obtient :

$$((C_1, C_2)(B_1, B_2))\,(A_1, A_2) = \Big(\big((C_1)^{B_2} B_1\big)^{A_2} A_1,\ (A_2)^{(C_1)^{B_2} B_1} (B_2)^{C_1} C_2\Big).$$

Il reste donc maintenant à montrer que les deux paires sont égales. Pour commencer, l'équation (4.9) page 60 nous donne l'égalité :

$$(C_1)^{(A_2)^{B_1} B_2} = \big((C_1)^{B_2}\big)^{(A_2)^{B_1}}.$$

En utilisant cette égalité et l'équation (4.9) page 60, on a :

$$(C_1)^{(A_2)^{B_1} B_2} (B_1)^{A_2} = \big((C_1)^{B_2}\big)^{(A_2)^{B_1}} (B_1)^{A_2} = \big((C_1)^{B_2} B_1\big)^{A_2}.$$

En multipliant cette équation par l'opérateur A_1 à droite, on retrouve les premiers éléments des deux paires. En échangeant les polynômes A_2 avec C_1, B_1 avec B_2 et C_2 avec A_1, on se retrouve avec les mêmes égalités à démontrer. On peut donc conclure sur l'associativité de la multiplication.

Il reste à montrer que la multiplication est distributive à gauche par rapport à l'addition.

Pour toutes paires (A_1, A_2), (B_1, B_2) et (C_1, C_2) on a d'une part

$$\begin{aligned}(A_1, A_2)\,((B_1, B_2) + (C_1, C_2)) &= (A_1, A_2)\Big((C_2)^{B_2} B_1 + (B_2)^{C_2} C_1,\ \operatorname{lclm}(B_2, C_2)\Big)\\ &= \Big((A_1)^{\operatorname{lclm}(B_2,C_2)}\Big((C_2)^{B_2} B_1 + (B_2)^{C_2} C_2\Big),\ (\operatorname{lclm}(B_2, C_2))^{A_1} A_2\Big),\end{aligned} \tag{4.20}$$

et d'autre part, en utilisant le lemme 4.13,

$$\begin{aligned}&(A_1, A_2)(B_1, B_2) + (A_1, A_2)(C_1, C_2) = \Big((A_1)^{B_2} B_1,\ (B_2)^{A_1} A_2\Big) + \Big((A_1)^{C_2} C_1, (C_2)^{A_1} A_2\Big)\\ &= \Big(\big((C_2)^{A_1} A_2\big)^{(B_2)^{A_1} A_2} (A_1)^{B_2} B_1 + \big((B_2)^{A_1} A_2\big)^{(C_2)^{A_1} A_2} (A_1)^{C_2} C_1,\ \operatorname{lclm}((B_2)^{A_1}, (C_2)^{A_1}) A_2\Big).\end{aligned}$$

Le lemme 4.13 page 61 permet de simplifier cette expression en supprimant les A_2 du premier élément de cette paire

On remarque qu'en utilisant l'équation (4.9) page 60, le premier élément de la paire précédente devient :

$$\big((C_2)^{A_1} A_1\big)^{B_2} B_1 + \big((B_2)^{A_1} A_1\big)^{C_2} C_1 = \big((A_1)^{C_2} C_2\big)^{B_2} B_1 + \big((A_1)^{B_2} B_2\big)^{C_2} C_1.$$

En utilisant à nouveau l'équation (4.9), cet opérateur est égal à :

$$\begin{aligned}&\big((A_1)^{C_2}\big)^{(B_2)^{C_2}} (C_2)^{B_2} B_1 + \big((A_1)^{B_2}\big)^{(C_2)^{B_2}} (B_2)^{C_2} C_1\\ &\qquad = (A_1)^{(B_2)^{C_2} C_2} (C_2)^{B_2} B_1 + (A_1)^{(C_2)^{B_2} B_2} (B_2)^{C_2} C_1\\ &\qquad = (A_1)^{\operatorname{lclm}(B_2,C_2)} (C_2)^{B_2} B_1 + (A_1)^{\operatorname{lclm} B_2,C_2} (B_2)^{C_2} C_1.\end{aligned}$$

On obtient bien de cette façon l'égalité voulue.

L'égalité des seconds éléments des deux paires est une conséquence immédiate du lemme 4.14 page 61.

□

Il existe un élément neutre pour l'addition qui est $(0,1)$. Il n'y a pas d'inverse pour l'addition. En effet, l'opposé naturel $(-A_1, A_2)$ se somme avec (A_1, A_2) en

$$(A_1, A_2) + (-A_1, A_2) = (0, A_2) \neq (0,1)\,.$$

En outre, la multiplication n'est pas distributive à droite sur l'addition : alors que

$$((A_1, A_2) + (-A_1, A_2))\,(B_1, B_2) = (0, A_2),$$

on a aussi

$$(A_1, A_2)(B_1, B_2) + (-A_1, A_2)(B_1, B_2) = (0, (B_2)^{A_1} A_2).$$

Malgré la pauvreté de la structure de cet ensemble de paires, celles-ci sont utiles dans le chapitre 8 afin de calculer des opérateurs de récurrence.

2.2 Corps des fractions d'opérateurs

Définition du corps

Comme dans le cas commutatif, le corps des fractions de polynômes de Ore se construit en quotientant l'ensemble des paires de polynômes de Ore par une relation d'équivalence donnée par le lclm.

Définition 4.37. Deux paires (A_1, A_2) et (B_1, B_2) sont dites équivalentes lorsque

$$(B_2)^{A_2} A_1 = (A_2)^{B_2} B_1. \tag{4.21}$$

Notation 4.38. On note $(A_1, A_2) \equiv (B_1, B_2)$ lorsque deux paires sont équivalentes.

Exemple 4.39. Les paires

$$(A_1, A_2) \;:=\; (nS_n + n, nS_n^2 - n) \text{ et } (B_1, B_2) \;:=\; (1, S_n - 1) \tag{4.22}$$

sont équivalentes. En effet, $A_2 = n(S_n + 1)(S_n - 1)$, on a donc $\mathrm{lclm}(A_2, B_2) = A_2$. On en déduit que

$$(A_2)^{B_2} B_1 = n(S_n + 1)B_1 = nS_n + n = A_1 = (B_2)^{A_2} A_1.$$

Remarque 4.40. Une définition équivalente est que deux paires sont équivalentes s'il existe deux opérateurs de récurrence K_A et K_B tels qu'on ait l'égalité suivante :

$$(K_A A_1, K_A A_2) = (K_B B_1, K_B B_2), \tag{4.23}$$

ce qui revient à dire que la différence de deux paires équivalentes est équivalente à 0 (définit comme la paire équivalente à $(0,1)$). En effet si deux paires sont équivalentes, on a l'égalité :

$$(B_2)^{A_2}A_1 = (A_2)^{B_2}B_1.$$

Alors en prenant les opérateurs $K_A = (B_2)^{A_2}$ et $K_B = (A_2)^{B_2}$ on retrouve l'égalité (4.23).

Réciproquement s'il existe K_A et K_B tels que l'égalité (4.23) est vérifiée, on a alors

$$K_A = \mathrm{gcld}(K_A, K_B)(B_2)^{A_2} \quad \text{et} \quad K_B = \mathrm{gcld}(K_A, K_B)(A_2)^{B_2}.$$

Comme l'anneau des polynômes de Ore est intègre, on peut diviser à gauche par $\mathrm{gcld}(K_A, K_B)$ l'équation déduite de (4.23) :

$$K_A A_1 = K_B B_1.$$

On retrouve de cette façon la relation (4.21).

Cette définition détermine une relation binaire qui est symétrique et réflexive. Il reste à montrer la transitivité pour montrer qu'il s'agit bien d'une relation d'équivalence. Soient trois paires (A_1, A_2), (B_1, B_2) et (C_1, C_2) telles que $(A_1, A_2) \equiv (B_1, B_2)$ et $(B_1, B_2) \equiv (C_1, C_2)$. On a

$$(B_2)^{A_2}A_1 = (A_2)^{B_2}B_1$$

et

$$(C_2)^{B_2}B_1 = (B_2)^{C_2}C_1.$$

En utilisant l'équation (4.12) page 61 on a l'égalité

$$(C_2)^{\mathrm{lclm}(A_2,B_2)}(A_2)^{B_2}B_1 = (\mathrm{lclm}(A_2, C_2))^{B_2}B_1$$

et

$$(A_2)^{\mathrm{lclm}(B_2,C_2)}(C_2)^{B_2}C_1 = (\mathrm{lclm}(A_2, C_2))^{B_2}C_1.$$

On a donc :

$$(C_2)^{\mathrm{lclm}(A_2,B_2)}(B_2)^{A_2}A_1 = (A_2)^{\mathrm{lclm}(B_2,C_2)}(B_2)^{C_2}C_1.$$

Il est facile de voir que l'on a aussi la suite d'égalité :

$$(C_2)^{\mathrm{lclm}(A_2,B_2)}(B_2)^{A_2}A_2 = \mathrm{lclm}(A_2, B_2, C_2) = (A_2)^{\mathrm{lclm}(B_2,C_2)}(B_2)^{C_2}C_2,$$

ce qui selon la remarque 4.40 implique que $(A_1, A_2) \equiv (C_1, C_2)$.

La classe des paires équivalentes à (A_1, A_2) est appelée une fraction et est dénotée $A_2^{-1}A_1$ (laquelle est égale à $B_2^{-1}B_1$).

Contrairement à l'ensemble des paires, l'ensemble des fractions vérifie les propriétés nécessaires pour être un anneau. On peut même montrer que c'est un corps comme l'affirme la proposition suivante.

Proposition 4.41 (Ore [Ore31])**.** *L'ensemble des fractions d'opérateurs muni des lois d'addition et de multiplication est un corps.*

Démonstration. Pour montrer que l'ensemble des fractions muni de l'addition est un groupe, il suffit de montrer l'existence d'un opposé pour chaque fraction. L'opposé de la fraction $A_2^{-1}A_1$ est $-A_2A_1^{-1}$, l'addition des deux paires associées donne $(0, A_2)$ qui est bien équivalent à l'élément neutre pour l'addition 0.

On a vu que la multiplication est associative et distributive à gauche pour les paires, elle le reste pour les fractions. Il reste donc à montrer qu'elle est aussi distributive à droite par rapport à l'addition. On a d'une part

$$\begin{aligned}(A_2^{-1}A_1 + B_2^{-1}B_1)C_2^{-1}C_1 &= \operatorname{lclm}(A_2, B_2)^{-1}((B_2)^{A_2}A_1 + (A_2)^{B_2}B_1)C_2^{-1}C_1 \\ &= \Big((C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\operatorname{lclm}(A_2, B_2)\Big)^{-1}\Big((B_2)^{A_2}A_1 + (A_2)^{B_2}B_1\Big)^{C_2}C_1,\end{aligned} \tag{4.24}$$

et d'autre part

$$\begin{aligned}A_2^{-1}A_1C_2^{-1}C_1 + B_2^{-1}B_1C_2^{-1}C_1 &= ((C_2)^{A_1}A_2)^{-1}(A_1)^{C_2}C_1 + ((C_2)^{B_1}B_2)^{-1}(B_1)^{C_2}C_1 \\ &= \Big(\operatorname{lclm}((C_2)^{A_1}A_2,(C_2)^{B_1}B_2)\Big)^{-1}\Big(\Big((C_2)^{B_1}B_2\Big)^{(C_2)^{A_1}A_2}(A_1)^{C_2}C_1+\Big((C_2)^{A_1}A_2\Big)^{(C_2)^{B_1}B_2}(B_1)^{C_2}C_1\Big).\end{aligned} \tag{4.25}$$

La première relation que l'on montre est que le dénominateur de la fraction présente dans l'équation (4.25) est un multiple gauche du dénominateur de la fraction de l'équation (4.24). Pour comparer ces deux fractions, il suffit alors de montrer que le numérateur de la fraction (4.24) multiplié à gauche par le polynôme

$$\Big(\operatorname{lclm}((C_2)^{A_1}A_2, (C_2)^{B_1}B_2)\Big)^{(C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}}\operatorname{lclm}(A_2, B_2), \tag{4.26}$$

est le numérateur de la fraction dans l'équation (4.24).

Le dénominateur de la fraction dans l'équation (4.25) est à la fois un multiple de A_2 et de B_2, on peut donc le factoriser par $\operatorname{lclm}(A_2, B_2)$:

$$\operatorname{lclm}((C_2)^{A_1}A_2, (C_2)^{B_1}B_2) = \Big(\operatorname{lclm}((C_2)^{A_1}A_2, (C_2)^{B_1}B_2)\Big)^{\operatorname{lclm}(A_2,B_2)}\operatorname{lclm}(A_2, B_2).$$

L'équation (4.12) page 61 permet de rentrer le polynôme $\operatorname{lclm}(A_2, B_2)$ dans le polynôme $\operatorname{lclm}((C_2)^{A_1}A_2, (C_2)^{B_1}B_2)$. En utilisant la définition du lclm, on a $\operatorname{lclm}(A_2, B_2) = (B_2)^{A_2}A_2$ et $\operatorname{lclm}(A_2, B_2) = (A_2)^{B_2}B_2$. Ces deux remarques nous donnent l'égalité :

$$\Big(\operatorname{lclm}((C_2)^{A_1}A_2, (C_2)^{B_1}B_2)\Big)^{\operatorname{lclm}(A_2,B_2)} = \operatorname{lclm}\Big(\big((C_2)^{A_1}A_2\big)^{(B_2,A_2)_2^A}, \big((C_2)^{B_1}B_2\big)^{(A_2)^{B_2}}B_2\Big).$$

Ce qui se simplifie en utilisant l'égalité (4.11) page 61 puis l'égalité (4.9) en :

$$\operatorname{lclm}\Big(\big((C_2)^{A_1}\big)^{(B_2)^{A_2}}, \big((C_2)^{B_1}\big)^{(A_2)^{B_2}}\Big) = \operatorname{lclm}\Big((C_2)^{(B_2)^{A_2}A_1}, (C_2)^{(A_2)^{B_2}B_1}\Big). \tag{4.27}$$

Si on montre que le polynôme

$$\Big((C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\operatorname{lclm}(A_2,B_2)\Big)^{\operatorname{lclm}((C_2)^{A_1}A_2,(C_2)^{B_1}B_2)}$$

est égal au polynôme constant 1, on montre que le dénominateur de la fraction de l'équation (4.25) est un multiple du dénominateur de la fraction de l'équation (4.24). En reprenant l'écriture de l'équation (4.26) pour représenter le polynôme $\operatorname{lclm}((C_2)^{A_1}A_2,\ (C_2)^{B_1}B_2)$ et en utilisant le lemme 4.11, on peut supprimer le polynôme $\operatorname{lclm}(A_2,B_2)$. La simplification donnée lors de l'équation (4.27) permet la réécriture de ce polynôme comme :

$$\begin{aligned}\Big(C_2^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\Big)^{\operatorname{lclm}\left((C_2)^{(B_2)^{A_2}A_1},\ (C_2)^{(A_2)^{B_2}B_1}\right)}\\ = (C_2)^{\operatorname{lclm}\left((C_2)^{(B_2)^{A_2}A_1},\ (C_2)^{(A_2)^{B_2}B_1}\right)\left((B_2)^{A_2}A_1+(A_2)^{B_2}B_1\right)}.\end{aligned} \tag{4.28}$$

En développant le polynôme en exposant par rapport à l'addition, on obtient :

$$\begin{aligned}&\operatorname{lclm}\Big((C_2)^{(B_2)^{A_2}A_1},\ (C_2)^{(A_2)^{B_2}B_1}\Big)\Big((B_2)^{A_2}A_1+(A_2)^{B_2}B_1\Big)\\ &\qquad=\operatorname{lclm}\Big((C_2)^{(B_2)^{A_2}A_1}(B_2)^{A_2}A_1,\ (C_2)^{(A_2)^{B_2}B_1}(B_2)^{A_2}A_1\Big)\\ &\qquad\quad+\operatorname{lclm}\Big((C_2)^{(B_2)^{A_2}A_1}(A_2)^{B_2}B_1,\ (C_2)^{(A_2)^{B_2}B_1}(A_2)^{B_2}B_1\Big).\end{aligned}$$

En utilisant la définition d'un cofacteur d'un lclm, ce polynôme se ramène à :

$$\operatorname{lclm}\Big((B_2)^{A_2}A_1,\ C_2,\ (C_2)^{(A_2)^{B_2}B_1}(B_2)^{A_2}A_1\Big)+\operatorname{lclm}\Big((A_2)^{B_2}B_1,\ C_2,\ (C_2)^{(B_2)^{A_2}A_1}(A_2)^{B_2}B_1\Big).$$

On en déduit que ce polynôme est un multiple gauche de C_2. On peut donc conclure, en appliquant le lemme 4.13, que le polynôme de l'équation (4.28) est égal à 1.

On a montré que le dénominateur de la fraction de l'équation (4.25) est le produit de l'opérateur

$$\Big(\operatorname{lclm}((C_2)^{A_1}A_2,(C_2)^{B_1}B_2)\Big)^{(C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\operatorname{lclm}(A_2,B_2)} \tag{4.29}$$

avec le dénominateur de la fraction (4.24). Pour montrer l'équivalence entre les paires, il suffit donc de montrer que le numérateur de la fraction de l'équation (4.25) est le produit de cet opérateur (4.29) par le numérateur de la fraction (4.24).

En utilisant la simplification du polynôme $\operatorname{lclm}((C_2)^{A_1}A_2,(C_2)^{B_1}B_2)$ faite à l'équation (4.27), le polynôme (4.29) devient

$$\Big(\operatorname{lclm}\Big(\big((C_2)^{A_1}\big)^{(B_2)^{A_2}},\ \big((C_2)^{B_1}\big)^{(A_2)^{B_2}}\operatorname{lclm}(A_2,B_2)\Big)\Big)^{(C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\operatorname{lclm}(A_2,B_2)},$$

qui se simplifie en utilisant le lemme 4.13 en

$$\left(\operatorname{lclm}\left(\left((C_2)^{A_1}\right)^{(B_2)^{A_2}},\ \left((C_2)^{B_1}\right)^{(A_2)^{B_2}}\right)\right)^{(C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}}.$$

La multiplication donne avec l'équation (4.9) page 60 le polynôme :

$$\begin{aligned}&\left(\operatorname{lclm}\left(\left((C_2)^{A_1}\right)^{(B_2)^{A_2}},\ \left((C_2)^{B_1}\right)^{(A_2)^{B_2}}\right)\right)^{(C_2)^{(B_2)^{A_2}A_1+(A_2)^{B_2}B_1}\left((B_2)^{A_2}A_1+(A_2)^{B_2}B_1\right)^{C_2}C_1}\\ &\quad=\left(\operatorname{lclm}\left(\left((C_2)^{A_1}\right)^{(B_2)^{A_2}},\ \left((C_2)^{B_1}\right)^{(A_2)^{B_2}}\right)\left((B_2)^{A_2}A_1+(A_2)^{B_2}B_1\right)\right)^{C_2}C_1.\end{aligned}$$

En distribuant ce produit de polynômes par rapport à l'addition, puis en simplifiant en utilisant la définition d'un cofacteur et l'équation (4.9) page 60, ce polynôme se réécrit en :

$$\left(\operatorname{lclm}\left((C_2)^{A_1}A_1,\ (B_2)^{A_2}A_1,\ \left((C_2)^{B_1}B_2\right)^{A_2}A_1\right)+\operatorname{lclm}\left(\left((C_2)^{A_1}A_2\right)^{B_2}B_1,\ (A_2)^{B_2}B_1,\ (C_2)^{B_1}B_1\right)\right)^{C_2}C_1. \tag{4.30}$$

On peut se contenter dans un premier temps de simplifier uniquement la partie gauche de cette somme. La symétrie dans l'écriture des polynômes des deux côtés de cette somme permettra de déduire une simplification pour la partie droite. En utilisant la définition d'un cofacteur on peut simplifier le polynôme $(C_2)^{A_1}A_1$ en $\operatorname{lclm}(A_1, C_2)$. La présence de ce polynôme A_1 dans ce lclm est redondante avec la présence du polynôme $\left((C_2)^{B_1}B_2\right)^{A_2}A_1$ qui est déjà un multiple de A_1. De la même manière en utilisant l'équation (4.9), on voit que ce polynôme est aussi un multiple de $(B_2)^{A_2}A_1$, on peut donc supprimer ce dernier du lclm pour obtenir :

$$\operatorname{lclm}\left((C_2)^{A_1}A_1,\ (B_2)^{A_2}A_1,\ \left((C_2)^{B_1}B_2\right)^{A_2}A_1\right)=\operatorname{lclm}\left(C_2,\ \left((C_2)^{B_1}B_2\right)^{A_2}A_1\right).$$

Il suffit à ce moment de réutiliser la définition d'un cofacteur pour réécrire ce lclm en

$$\operatorname{lclm}\left((C_2)^{A_1}A_1,\ (B_2)^{A_2}A_1,\ \left((C_2)^{B_1}\right)^{(A_2)^{B_2}}(B_2)^{A_2}A_1\right)=\left(\left((C_2)^{B_1}B_2\right)^{A_2}A_1\right)^{C_2}C_2.$$

Le membre de l'équation (4.30) est donc finalement égal à

$$\left(\left(\left((C_2)^{B_1}B_2\right)^{A_2}A_1\right)^{C_2}C_2+\left(\left((C_2)^{A_2}A_2\right)^{B_2}B_1\right)^{C_2}C_2\right)^{C_2}C_1,$$

ce qui en simplifiant par C_2 ((4.11) page 61) devient

$$\left(\left((C_2)^{B_1}B_2\right)^{A_2}A_1\right)^{C_2}+\left(\left((C_2)^{A_2}A_2\right)^{B_2}B_1\right)^{C_2}C_1.$$

On peut voir assez rapidement que le numérateur de la fraction dans l'équation (4.25) est le même polynôme que celui qui vient d'être calculé. En effet en utilisant l'équation (4.9) puis l'équation (4.9), on a l'égalité :

$$\Big((C_2)^{B_1}B_2\Big)^{(C_2)^{A_1}A_2}(A_1)^{C_2}C_1 = \Big(\Big((C_2)^{B_1}B_2\Big)^{A_2}A_1\Big)^{C_2}C_1.$$

On a la même relation pour la partie droite de la somme et on peut donc conclure sur la distributivité de la multiplication par rapport à l'addition.

Cet ensemble est donc bien un anneau. Pour montrer que c'est un corps il suffit de remarquer que pour n'importe quelle fraction $A_2^{-1}A_1$, la fraction $A_1^{-1}A_2$ en est son inverse. □

Fractions d'opérateurs irréductibles

Dans la suite de ce mémoire, l'utilisation du gcld pour simplifier une fraction est importante. Cette section donne une définition d'une fraction simplifiée au maximum que l'on appelle fraction irréductible. Les résultats ici sont probablement connus, mais je ne les ai pas trouvés dans la littérature.

Définition 4.42. Une fraction $A_2^{-1}A_1$ est dite *irréductible* lorsque $\mathrm{gcld}(A_1, A_2) = 1$.

Proposition 4.43. *Pour toute fraction d'opérateurs de récurrence $A_2^{-1}A_1$, il existe une fraction irréductible égale à cette fraction. De plus son numérateur et son dénominateur sont uniques à un facteur dans $\mathbb{K}$ près.*

Démonstration. On obtient l'existence en divisant le numérateur et le dénominateur par $\mathrm{gcld}(A_1, A_2)$. Supposons que $B_2^{-1}B_1 = C_2^{-1}C_1$ et que $\mathrm{gcld}(B_1, B_2) = \mathrm{gcld}(C_1, C_2) = 1$. On a donc les égalités :

$$(C_2)^{B_2} = (C_2)^{B_2}\,\mathrm{gcld}(B_1, B_2) \quad \text{et} \quad (B_2)^{C_2} = (B_2)^{C_2}\,\mathrm{gcld}(C_1, C_2).$$

Grâce au lemme 4.11, on a :

$$(C_2)^{B_2}\,\mathrm{gcld}(B_1, B_2) = \mathrm{gcld}((C_2)^{B_2}B_1, (C_2)^{B_2}B_2)$$

et

$$(B_2)^{C_2}\,\mathrm{gcld}(C_1, C_2) = \mathrm{gcld}((B_2)^{C_2}C_1, (B_2)^{C_2}C_2).$$

Par définition de la relation d'équivalence entre deux paires, $(C_2)^{B_2}B_1 = (B_2)^{C_2}C_1$ et par définition d'un cofacteur, $(C_2)^{B_2}B_2 = (B_2)^{C_2}C_2$, on a donc $(C_2)^{B_2} = (B_2)^{C_2}$.

Comme $\mathrm{gcld}((B_2)^{C_2}, (C_2)^{B_2}) = 1$ par définition du lclm, nécessairement $(B_2)^{C_2} = (C_2)^{B_2} = 1$ et donc $B_1 = C_1$ et $B_2 = C_2$. □

L'algorithme4.7 montre comment calculer une fraction irréductible :

Entrée: Une fraction $A_2^{-1}A_1$
Sortie: La fraction $B_2^{-1}B_1$ telle que $\gcld(B_1, B_2) = 1$ et $B_2^{-1}B_1 = A_2^{-1}A_1$
Calculer $\gcld(A_1, A_2)$
Calculer B_1 et B_2 tel que $A_1 = \gcld(A_1, A_2)B_1$ et $A_2 = \gcld(A_1, A_2)B_2$ en utilisant un algorithme de division à gauche.
renvoyer (B_1, B_2)

Algorithme 4.7: Normalisation d'une fraction

Exemple 4.44. Une autre façon de voir l'exemple 4.34 page 72 est de partir des égalités entre $T_n(x)$, $xT_n(x)$ et $T'_n(x)$ pour en déduire une égalité entre $T_n(x)$ et $(1-x^2)T'_n$. On définit d'abord l'opérateur de multiplication par x et l'opérateur de dérivation avec le code Maple suivant :

```
> X := OreFrac([1,0,1],[0,2]);
> Dx := OreFrac([0,-2*n*(n+2)],[n+2,0,-n]);
```

$$X := [1 + Sn^2, 2Sn]$$

$$Dx := [-2n\,(n+2Sn)\,, n+2-nSn^2]$$

En utilisant les propositions 4.29 page 70 et 4.32 page 71, on peut calculer l'opérateur XDx permettant d'obtenir une relation entre $T_n(x)$ et $(1-x^2)T'_n(x)$ de la façon suivante :

```
> XDx := Dx.(1-X^2);
```

$$XDx := [-(n+1\,(n+3)) + 2\,(n+1)\,(n+3)\,Sn^2 - \left(n+1\,(n+3)\,Sn^4\right),$$
$$(-2n-6)\,Sn + (2n+2)\,Sn^3]$$

On n'obtient pas de cette façon l'égalité classique entre les polynômes T_n et $(1-x^2)T'_n$. En normalisant cette fraction grâce à la procédure normal de OreField, on retrouve les opérateurs souhaités :

```
> normal(XDx);
```

$$[n+1+(-n-1)\,Sn^2, 2\cdot Sn]$$

C'est-à-dire la relation :

$$2(1-x^2)T'_{n+1} = (n+1)T_n - (n+1)T_{n+2}.$$

Le lemme suivant est utile pour minimiser l'ordre des récurrences.

Lemme 4.45. *Soit $A_2^{-1}A_1$ une fraction irréductible et B_1 un polynôme. Alors $((A_2)^{B_1})^{-1}(B_1)^{A_2}A_1$ est irréductible et égale à $B_1A_2^{-1}A_1$.*

Démonstration. On a par définition du lclm, $\gcld((A_2)^{B_1}, (B_1)^{A_2}) = 1$. Le polynôme $\gcld((A_2)^{B_1}, (B_1)^{A_2}A_1)$ est un diviseur à gauche de $\lclm(B_1, A_2) = (A_2)^{B_2}B_2$, et est donc un diviseur à gauche de :

$$\gcld(\lclm(B_1, A_2), (B_1)^{A_2}A_1),$$

que l'on peut simplifier en utilisant la définition d'un cofacteur et le lemme 4.11 page 60 :

$$(B_1)^{A_2} \operatorname{gcld}(A_1, A_2).$$

La fraction $A_2^{-1}A_1$ étant irréductible, $\operatorname{gcld}(A_1, A_2) = 1$, donc ce polynôme est égal à $(B_1)^{A_2}$. C'est-à-dire, le polynôme $\operatorname{gcld}((A_2)^{B_1}, (B_1)^{A_2}A_1)$ est à la fois un diviseur du polynôme $(B_1)^{A_2}$ et du polynôme $(A_2)^{B_1}$, il est donc égal à 1. □

Chapitre 5

Séries de Tchebyshev solutions d'équations différentielles

Résumé

Lorsqu'une série de Tchebychev est solution d'une équation différentielle à coefficients polynomiaux, ses coefficients satisfont une récurrence linéaire. Dans ce chapitre, on interprète cette récurrence comme le numérateur d'une fraction d'opérateurs de récurrence. Cette interprétation nous donne un point de vue simple des algorithmes existants pour calculer cette récurrence, nous permet d'analyser leurs complexités, et d'en concevoir un plus rapide pour des ordres grands.

Une grosse partie de ce chapitre est issue d'un article [BS09] écrit avec Bruno Salvy.

Sommaire

1 Introduction

Dans le cas fréquent où la fonction f est D-finie, la suite des coefficients de son développement en série de Taylor est P-récursive. Les coefficients de la série de Tchebychev vérifient eux aussi une récurrence linéaire à coefficients polynomiaux. Cette récurrence peut servir par exemple à calculer numériquement les coefficients plus rapidement, elle est d'ailleurs utilisée par [Wim84] puis dans les chapitres 6 et 7 pour cette raison. Une autre utilité de ces récurrences est le calcul de forme explicite des coefficients grâce notamment à l'algorithme de Petkovšek [Pet92], ou encore la vérification de ces formes explicites.

On rappelle que les coefficients de Tchebychev sont définis par le produit scalaire de l'espace de Hilbert associé aux polynômes de Tchebychev (voir (2.17) page 23). Clenshaw [Cle57] a donné une méthode numérique pour calculer les coefficients c_n d'une série de Tchebychev f à partir de l'équation différentielle vérifiée par f sans calculer toutes les intégrales. Dans ce cas, les coefficients c_n sont liés par une récurrence linéaire que nous appelleront récurrence de Tchebychev Des méthodes pour le calcul de ces récurrences ont été montrées par différents auteurs, premièrement pour les récurrences de petit ordre [Fox62, FP68, Luk69], puis plus généralement par Paszkowski [Pas75] et dans le contexte du calcul formel par Geddes [Ged77b]. L'algorithme de Paszkowski a été amélioré par Lewanowicz [Lew76] qui a donné un algorithme pour calculer la récurrence de plus petit ordre en un certain sens. Cependant, l'algorithme de Lewanowicz n'est pas beaucoup discuté dans la littérature car il paraît compliqué (voir l'article original et les commentaires dans [Wim84, p. 186]). Plus récemment, d'autres méthodes ont été exposées par Rebillard [Reb98] et Rebillard et Zakrajšek [RZ06].

Ce chapitre donne une représentation simple et unifiée de la plupart de ces algorithmes, et en présente un plus rapide pour les récurrences d'ordres grands. Laissant pour plus tard les preuves et définitions, l'idée de base peut être présentée par analogie avec le calcul des récurrences pour les coefficients d'une série de Taylor. La base monomiale $M_n(x) = x^n$ satisfait

$$xM_n(x) = M_{n+1}(x), \qquad M_n'(x) = nM_{n-1}(x+1). \tag{5.1}$$

Les relations analogues pour les polynômes de Tchebychev ont été rappelées dans (2.4) et (2.9) page 18 :

$$2xT_n(x) = T_{n+1}(x) + T_{n-1}(x), \tag{5.2}$$

$$2(1-x^2)T_n'(x) = -nT_{n+1}(x) + nT_{n-1}(x). \tag{5.3}$$

Étant donnée une série $f(x) = \sum c_n M_n(x)$, (5.1) conduit à l'expression des coefficients des séries de Taylor xf et f' en fonction de c_n : la multiplication par x amène à un décalage négatif des indices ; la dérivée amène à un décalage positif des indices suivie d'une multiplication par $n+1$. Algébriquement, on obtient ainsi un morphisme qui envoie x en X et ∂_x en D, avec $X := S_n^{-1}$, $D := (n+1)S_n$. Ici, S_n dénote l'opérateur de décalage ($S_n \cdot n = n+1$) déjà défini dans le chapitre 4. Maintenant, si f est solution d'une équation différentielle

linéaire à coefficients polynomiaux

$$p_k(x)f^{(k)}(x) + \cdots + p_0(x)f(x) = 0,$$

on déduit un opérateur de récurrence $p_k(X)D^k + \cdots + p_0(X)$ pour ses coefficients de Taylor.

Exemple 5.1. L'exemple le plus simple est la fonction exponentielle, pour laquelle $f' - f = 0$ devient $D - 1 = (n+1)S_n - 1$ (1 dénote l'identité), ce qui donne la récurrence $(n+1)c_{n+1} - c_n = 0$ satisfaite par $c_n = 1/n!$.

Le processus pour une série $f(x) = \sum' c_n T_n(x)$ commence de manière similaire : la multiplication par x amène à l'opérateur

$$X := (S_n + S_n^{-1})/2. \tag{5.4}$$

La différence avec les séries de Taylor provient du facteur $(1-x^2)$ dans (5.3). L'opération de dérivation suivie de la multiplication par $(1 - x^2)$ se ramène facilement à l'opérateur $(S_n - S_n^{-1})n/2$, mais un simple opérateur linéaire pour les coefficients de Tchebychev de f' n'existe pas. L'idée à ce stade est de diviser par $1 - x^2$, ce qui amène à introduire un *inverse formel* de $1 - X^2$ en utilisant les fractions d'opérateurs de récurrence. On écrit donc $D = (1 - X^2)^{-1}(S_n - S_n^{-1})n/2$. Ceci peut-être simplifié en : $1 - X^2 = -(S_n - S_n^{-1})^2/4$, après quoi

$$D = 2(S_n^{-1} - S_n)^{-1}n. \tag{5.5}$$

Exemple 5.2. Pour la fonction exponentielle, on a donc :

$$D - 1 = 2(S_n^{-1} - S_n)^{-1}n - 1 = (S_n^{-1} - S_n)^{-1}(2n - (S_n^{-1} - S_n)).$$

Le numérateur de cette fraction correspond à la récurrence satisfaite par les coefficients de Tchebychev de l'exponentielle.

$$2nc_n - c_{n-1} + c_{n+1} = 0. \tag{5.6}$$

Il s'avère que dans cet exemple, les coefficients de Tchebychev sont connus : $c_n = 2\,\mathrm{I}_n(1)$, où I_n est la fonction de Bessel première espèce modifiée, et ils satisfont bien la récurrence (5.6).

Cet exemple se généralise. On montre dans ce chapitre que tous les algorithmes mentionnés ci-dessus peuvent être interprétés comme : d'abord réécrire l'équation différentielle d'une façon ou d'une autre, après appliquer le morphisme susmentionné pour finalement retourner le numérateur du résultat. Dans le cas de l'algorithme de Lewanowicz, la fraction est normalisée (son numérateur et son dénominateur sont premiers entre eux). C'est pourquoi sa sortie peut avoir un ordre plus petit.

Dans la section 2, on applique les propriétés du chapitre 4 dans le cas spécifique des séries de Tchebychev. On donne ensuite une représentation unifiée des algorithmes de Paszkowski et Rebillard, on fournit une analyse de complexité et puis on conçoit un algorithme plus rapide dans la section 3. La dernière section, 5, donne des propriétés des solutions des récurrences de Tchebychev. Elle permet de faire correspondre les solutions de la récurrence avec les solutions de l'équation différentielle.

Entrée: $L := \sum_{i=0}^{k} p_i(x)\partial^i$
Sortie: (P, Q) tel que $\varphi(L) = Q^{-1}P$
  $P := p_k(X)$
  $Q := 1$
  pour tout i de $k-1$ à 0 **faire**
    Calculer $\mathrm{lclm}((S_n^{-1} - S_n), P) = \hat{P}P = \hat{U}(S_n^{-1} - S_n)$.
    $Q := \hat{P}Q$
    $P := \hat{U}2n + Qp_i(X)$
  fin pour
  renvoyer (P, Q)

Algorithme 5.1: Algorithme de Lewanowicz

2 Récurrences pour les coefficients de Tchebychev

Les outils théoriques développés dans le chapitre 4 nous permettent de prouver le morphisme qui envoie les opérateurs différentiels linéaires en des fractions d'opérateurs de récurrence linéaires. Les numérateurs de ces fractions nous donnent les récurrences satisfaites par les coefficients des séries de Tchebychev solutions d'équations différentielles.

Les algorithmes deviennent faciles par la suite, les difficultés algorithmiques étant concentrées dans l'algorithme d'Euclide étendu (algorithme 4.3 page 58) du chapitre précédent.

2.1 Morphisme

Nous définissons un morphisme de $\mathbb{Q}$-algèbre de $\mathbb{Q}[x]\langle\partial_x; \mathrm{Id}, d/dx\rangle$ dans le corps des fractions d'opérateurs de $Q(n)\langle S_n, S_n^{-1}; S_n\rangle$ par

$$\varphi(x) = X := \frac{1}{2}(S_n + S_n^{-1}), \quad \varphi(\partial_x) = D := (S_n^{-1} - S_n)^{-1}(2n).$$

La preuve que φ est un morphisme d'anneaux non commutatif bien défini se réduit à valider la commutation $\varphi(\partial_x x) = \varphi(x\partial_x + 1)$. Et en effet,

$$\begin{aligned} XD + 1 &= \frac{1}{2}(S_n + S_n^{-1})(S_n^{-1} - S_n)^{-1}(2n) + 1 \\ &= (S_n^{-1} - S_n)^{-1}(S_n + S_n^{-1})n + 1 \\ &= (S_n^{-1} - S_n)^{-1}\Big(\big((n+1)S_n + (n-1)S_n^{-1}\big) + (S_n^{-1} - S_n)\Big) \\ &= DX. \end{aligned}$$

2.2 Règle d'Horner et algorithme de Lewanowicz

Proposition 5.3. *Soit $L = p_k(x)\partial_x^k + \cdots + p_1(x)\partial_x + p_0(x)$ un opérateur différentiel linéaire dans $\mathbb{Q}[x]\langle\partial_x; \mathrm{Id}, d/dx\rangle$. L'évaluation de $\varphi(L)$ par la méthode d'Horner*

$$\varphi(L) = (\cdots(p_k(X)D + p_{k-1}(X))D + \cdots)D + p_0(X),$$

en utilisant les formules (4.14) page 70 et (4.15) page 71 pour le calcul des sommes et des produits, fournit une fraction $Q^{-1}P$ qui est irréductible.

L'algorithme déduit de cette assertion (algorithme 5.1) est dû à Lewanowicz. Il est rendu très clair par l'utilisation des opérateurs de récurrence. La preuve que le numérateur retourné par cet algorithme donne une récurrence pour les coefficients de Tchebychev est donnée dans la prochaine section.

Démonstration. On prouve que chaque itération de la boucle produit des opérateurs P et Q tels que $\mathrm{gcld}(P, Q) = 1$ et :

$$Q^{-1}P =: M_i = \varphi(p_k(x)\partial^{k-i} + \cdots + p_i).$$

À l'initialisation, $i = k$ et $\varphi(p_k(x))$ est un polynôme, on a donc $Q = 1$ et la propriété est vérifiée. Si c'est vrai pour M_i, la prochaine étape de la boucle calcule $Q^{-1}PD + p_{i-1}(X)$. Comme $D = (S_n^{-1} - S_n)^{-1}(2n)$, nous avons

$$\mathrm{lclm}((S_n^{-1} - S_n), \quad P) = \left(S_n^{-1} - S_n\right)^P P = (P)^{S_n^{-1} - S_n}(S_n^{-1} - S_n).$$

Le lemme 4.45 page 82 appliqué à l'inverse de $(S_n^{-1} - S_n)P^{-1}Q$ implique alors que

$$\mathrm{gcld}(\left(S_n^{-1} - S_n\right)^P Q, (P)^{S_n^{-1} - S_n}) = 1.$$

Il suit que

$$\mathrm{gcld}(\left(S_n^{-1} - S_n\right)^P Q, (P)^{S_n^{-1} - S_n} + \left(S_n^{-1} - S_n\right)^P Q p_{i-1}(X)\frac{2}{n}) = 1.$$

Encore par le lemme 4.45 appliqué à l'inverse, multiplier par $2n$ à droite préserve l'irréductibilité et la propriété est vérifiée pour M_{i-1}. □

Nous citons sans preuve le résultat suivant.

Proposition 5.4 ([Lew76])**.** *Quand le coefficient de tête $p_k(x)$ de l'équation différentielle ne s'annule ni en 1 ni en −1, alors tous les* gcrd *sont triviaux (donc les* lclm *sont maximaux), $Q = D^{-i}$ à l'étape i et le résultat Q est D^{-k}.*

Pour se donner une idée de l'importance du fait que le coefficient de tête de l'équation différentielle ne s'annule pas en ±1, on peut remarquer que $4(X^2 - 1) = (S_n^{-1} - S_n)^2$, donc

$$(X^2 - 1)D = \frac{1}{2}(S_n^{-1} - S_n)n.$$

Le gcrd entre le numérateur de $(X^2 - 1)$ et D est alors non trivial.

2.3 Développement en série de Tchebychev

Théorème principal pour le calcul des récurrences

On prouve maintenant notre principal résultat pour le calcul des récurrences de Tchebychev : le morphisme défini ci-dessus se comporte comme attendu avec les séries de Tchebychev.

Théorème 5.5. *Soit $L = p_0(x)+\cdots+p_k(x)\partial_x^k$ un opérateur différentiel linéaire à coefficients polynomiaux d'ordre k. Soit $f \in \mathcal{C}^k(]-1,1[)$ telle que l'une des hypothèses suivantes est vérifiée :*

$$\int_{-1}^{1} \frac{f^{(k)}(x)}{\sqrt{1-x^2}}\,dx \quad \text{est convergente ;} \tag{H}$$

$$\int_{-1}^{1} \frac{(1-x^2)^k f^{(k)}(x)}{\sqrt{1-x^2}}\,dx \quad \text{est convergente et } (1-x^2)^i | p_i, i = 0,\dots,k. \tag{H'}$$

Alors f admet un développement en série de Tchebychev $\sum' u_n T_n$, $L\cdot f$ admet un développement en série de Tchebychev $\sum' v_n T_n$ et les suites u et v sont liées par $P_1 \cdot u_n = P_2 \cdot v_n$, pour tout (P_1, P_2) tel que $P_2^{-1}P_1 = \varphi(L)$. En particulier, si $L \cdot f = 0$, alors les coefficients de Tchebychev de f satisfont $P_1 \cdot u_n = 0$ pour tout numérateur de $\varphi(L)$.

Remarque 5.6. La récurrence déduite d'une équation différentielle comme dans le théorème 5.5 est appelée la récurrence de Tchebychev.

Le cas facile est quand (H) est vraie. L'hypothèse (H') rend possible de travailler avec certaines fonctions qui sont singulières en ±1, mais avec une singularité « pas trop méchante » : c'est un point singulier régulier.

Démonstration. Premièrement, la convergence de l'intégrale en (H) ou (H') implique la convergence de l'intégrale similaire où $f^{(k)}$ est remplacé par $f^{(i)}$ pour $i = k-1,\dots,0$ aussi bien que des intégrales où ces fonctions sont multipliées par $T_n(x)$, $n \in \mathbb{N}$. Ceci montre qu'à la fois f et $L\cdot f$ admettent des développements en série de Tchebychev.

Si le résultat est vrai pour tout numérateur de $\varphi(L)$ alors il est vrai en particulier pour le numérateur de sa forme irréductible. Inversement, si $P_1\cdot u_n = P_2\cdot v_n$, alors $RP_1\cdot u_n = RP_2\cdot v_n$ pour tout R, il est donc suffisant de prouver le résultat pour la forme irréductible de $\varphi(L)$.

Lemme 5.7 (Cas basique)**.** *Sous les mêmes hypothèses, le résultat est vrai lorsque L est une constante multipliée par l'identité, $L = x$, $L = \partial_x$ si (H) est vraie, $L = (1-x^2)\partial_x$ si (H') est vraie.*

Démonstration. Si $L = \lambda$ est une constante multipliée par l'identité alors $P_1 = \lambda$, $P_2 = 1$ et $v_n = \lambda u_n$ vérifient la proposition clairement.

Si $L = x$, l'équation (5.2) implique

$$\begin{aligned} v_n &= \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)xT_n(x)}{\sqrt{1-x^2}}\,dx \\ &= \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)(T_{n+1}(x)+T_{n-1}(x))}{2\sqrt{1-x^2}}\,dx = X\cdot u_n. \end{aligned}$$

Si $L = \partial_x$ et (H) est vraie, on utilise la variante de l'équation (5.3) suivante quand $n \neq 0$

$$\frac{T_n}{\sqrt{1-x^2}} = \left(\frac{T_{n+1}-T_{n-1}}{2n\sqrt{1-x^2}}\right)' = -\frac{1}{n^2}(\sqrt{1-x^2}T_n')',$$

variante qui peut être vérifiée par (2.9) page 20. La continuité de f' et la convergence des intégrales de (H) impliquent que l'intégration par partie est possible et elle donne

$$\begin{aligned} u_n &= \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_n(x)}{\sqrt{1-x^2}}\,dx \\ &= \left[-\frac{2f(x)\sqrt{1-x^2}T_n'}{\pi n^2}\right]_{-1}^{1} + \frac{2}{\pi}\int_{-1}^{1} \frac{f'(x)(T_{n-1}(x)-T_{n+1}(x))}{2n\sqrt{1-x^2}}\,dx \\ &= D^{-1}\cdot v_n. \end{aligned}$$

Par la convergence de l'intégrale (H), les deux limites des termes entre les crochets sont nulles

Le cas où $n = 0$ se réduit à vérifier que $v_{-1} = v_1$, ce qui est vrai, les suites de coefficients de Tchebychev étant symétriques (voir proposition 2.12 page 23).

Si $L = (1-x^2)\partial_x$ et (H') est vraie, on part de

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

Un argument similaire au précédent donne

$$\begin{aligned} (n+1)u_{n+1}-(n-1)u_{n-1} &= \frac{2}{\pi}\int_{-1}^{1} f(x)\frac{(n+1)T_{n+1}(x)-(n-1)T_{n-1}(x)}{\sqrt{1-x^2}}dx \\ &= 2\frac{2}{\pi}\int_{-1}^{1}(1-x^2)f'(x)\frac{T_n}{\sqrt{1-x^2}}dx = 2v_n, \end{aligned}$$

lequel prouve le résultat puisque $\varphi((1-x^2)\partial_x) = (1-X^2)D = (S_n - S_n^{-1})n/2$. □

Lemme 5.8 (Produit)**.** *Supposons que le résultat est vrai pour L_2 avec f, ainsi que pour un autre opérateur L_1 avec $L_2\cdot f$. Soit $\varphi(L_1) = P_2^{-1}P_1$ et $\varphi(L_2) = Q_2^{-1}Q_1$, ces fractions étant irréductibles. Soit* $\operatorname{lclm}(Q_2,P_1) = (Q_2)^{P_1}P_1 = (P_1)^{Q_2}Q_2$*, supposons que $((Q_2)^{P_1}P_2)^{-1}(P_1)^{Q_2}Q_1$ est irréductible. Alors le résultat est vrai pour L_1L_2 avec f.*

Démonstration. Soient les suites v, w, u liées par $P_1 \cdot v = P_2 \cdot w$, $Q_1 \cdot u = Q_2 \cdot v$. Alors

$$(Q_2)^{P_1} P_2 \cdot w = (Q_2)^{P_1} P_1 \cdot v = (P_1)^{Q_2} Q_2 \cdot v = (P_1)^{Q_2} Q_1 \cdot u,$$

d'où le résultat. □

Par conséquence, le résultat est vrai quand $L = \lambda x^i$ est un monôme.

Lemme 5.9 (Somme)**.** *Supposons que le résultat est vrai pour un opérateur L avec f et pour un polynôme p avec le même f. Alors il est vrai pour $L + p$ avec f.*

Démonstration. Soit $\varphi(L) = P_2^{-1} P_1$ la représentation irréductible. Si $P_1 \cdot u = P_2 \cdot v$, $w = p(X) \cdot u$, alors

$$P_2 \cdot (v + w) = (P_1 + P_2 p(X)) \cdot u.$$

Ceci prouve la propriété pour $L + p$ puisque $\gcld(P_2, P_1 + P_2 p(X)) = \gcld(P_2, P_1) = 1$. □

Le résultat est maintenant établi pour L un polynôme arbitraire, comme somme de ses monômes.

Soient finalement $\theta = \partial_x$ si (H) est vraie et $\theta = (1 - x^2)\partial_x$ si (H') l'est. Dans les deux cas, L peut se réécrire $q_k(x)\theta^k + \cdots + q_0(x)$ avec les polynômes $q_k, \ldots, q_0$. L'hypothèse sur f implique que le résultat est vrai pour $L = 1$ avec $\theta^i \cdot f$ pour $i = 0, \ldots, k$ et aussi pour $L = q_i(x)$ avec $\theta^i \cdot f$ par le lemme 5.8.

Soient $L_k = q_k(X)$ et $L_i = L_{i+1}\theta + q_i$ pour $i = k-1, \ldots, 0$. Soit $\varphi(L_i) = P_{(2,i)}^{-1} P_{(1,i)}$ la représentation irréductible. On prouve par récurrence que le résultat est vrai pour L_i avec $\theta^i f$. Pour $i = k$, le résultat vient juste d'être prouvé. Si le résultat est vrai pour L_{i+1} avec $\theta^{i+1} f$, alors on obtient la fraction irréductible $P_{(2,i+1)}^{-1} P_{(1,i+1)} \varphi(\theta)$: quand $\theta = \partial_x$ cela découle du lemme 5.8, tandis que lorsque $\theta = (1 - x^2)\partial_x$, L_{i+1} est lui même un polynôme (par récurrence).

Ainsi le résultat est vrai pour $L_{i+1}\theta$ avec $\theta^i f$. Comme il est aussi vrai pour $q_i(x)$ avec $\theta^i f$ et que $q_i(x)$ est un polynôme, on obtient le résultat pour leurs sommes par le lemme 5.9. Ainsi par récurrence le résultat est prouvé pour L_0 avec f, ce qui conclut la preuve du théorème 5.5. □

Exemples

Exemple 5.10. La fonction $\exp(x)$ satisfait (H) pour tout k. Ceci prouve la récurrence (5.6) calculée dans l'exemple 5.2.

Exemple 5.11. La fonction $(1-x^2)^{-1/4}$ est annulée par $2(1-x^2)\partial_x - x$. l'hypothèse (H) n'est pas vérifiée, mais (H') l'est. L'application du morphisme donne $P_1 = (2n+3)S_n^2 - (2n+1)$, $P_2 = -2S_n$, de sorte que le théorème affirme que les coefficients de Tchebychev satisfont

$$(2n+3)c_{n+2} = (2n+1)c_n. \tag{5.7}$$

Les valeurs de ces coefficients peuvent être calculées par les propriétés classiques des intégrales Beta et en effet

$$c_n = \begin{cases} 0 & \text{si } n \text{ est impair,} \\ \frac{2\Gamma\left(\frac{n}{2}+\frac{1}{4}\right)}{\sqrt{\pi}\Gamma\left(\frac{n}{2}+\frac{3}{4}\right)} & \text{sinon.} \end{cases}$$

Exemple 5.12. La fonction $\arccos x$ donne un exemple qui montre que les hypothèses analytiques (H) ou (H') sont nécessaires. Cette fonction est annulée par $L = (1-x^2)\partial_x^2 - x\partial_x$. Une application directe du morphisme donne les opérateurs $P_1 = n^2$, $P_2 = 1$, lesquels pourraient suggérer que la récurrence est $n^2 c_n = 0$. Cependant, ni (H) ni (H') ne sont vérifiées dans ce cas. La multiplication de L par $(1-x^2)$ donne un nouvel opérateur tel que (H') est vérifiée. Alors le théorème prouve que les coefficients sont annulés par

$$(n+4)^2 S_n^4 - 2(n+2)S_n^2 + n^2. \tag{5.8}$$

Cette récurrence est vérifiée par les coefficients de Tchebychev qui valent :

$$c_n = \begin{cases} \pi & \text{si } n = 0, \\ 0 & \text{si } n > 0 \text{ est pair,} \\ -\frac{4}{n^2\pi} & \text{sinon.} \end{cases}$$

3 Nouvelle vision des algorithmes existants

On interprète maintenant les algorithmes de Paszkowski [Pas75] et Rebillard [Reb98] comme le calcul d'un numérateur d'une fraction d'opérateurs de récurrence. On propose aussi un nouvel algorithme plus rapide. Les trois algorithmes calculent la même récurrence. Partant de

$$L = \sum_{i=0}^{k} p_i(x)\partial_x^i, \tag{5.9}$$

ces algorithmes évitent d'utiliser des fractions en remplaçant les dérivations par des intégrations, exploitant le fait que

$$I := D^{-1} = \left(\frac{1}{2n}\right)\left(-S_n + S_n^{-1}\right)$$

est un polynôme (et non une fraction). Ces algorithmes calculent le polynôme $I^k\varphi(L)$, qui est le numérateur de $\varphi(L) = I^{-k}I^k\varphi(L)$. Ainsi, par le théorème 5.5, leur résultat est une récurrence annulant les coefficients des séries de Tchebychev solutions de L.

Si $p_k(1)p_k(-1) \neq 0$, la proposition 5.4 montre que I^k est le dénominateur de la fraction irréductible et de plus dans ce cas tous les algorithmes calculent la fraction irréductible. Sinon, le résultat de ces algorithmes peut avoir un plus grand ordre que celui retourné par l'algorithme de Lewanowicz.

Entrée: $L = \sum_{i=0}^{k} p_i(x)\partial_x^i$
Sortie: $I^k\varphi(L)$
Calculer $q_0, \ldots, q_k$ tels que $L = \sum_{i=0}^{k} \partial_x^i q_i(x)$
$R := q_k(X)$
pour tout i de 1 à k **faire**
$R := R + I^i q_{k-i}(X)$
fin pour
renvoyer R

Algorithme 5.2: Algorithme de Paszkowski

Exemple 5.13. La fonction $(1-x^2)^{-1/4}$ a été traitée avec l'exemple 5.11. L'algorithme de Lewanowicz retourne la récurrence d'ordre deux (5.7). Le numérateur retourné par les autres algorithmes est d'ordre 4.

$$(2n+1)c_n - 4(n+2)c_{n+2} + (2n+7)c_{n+4} = 0.$$

Il est cependant possible de récupérer une récurrence d'ordre plus petite : le gcld de $A = (2n+7)S_n^4 - 4(n+2)S_n^2 + (2n+1)$ avec I est I, de sorte que A se factorise comme $A = (S_n^2 - 1)P$ avec P comme dans l'exemple 5.11.

Plus généralement, en divisant le résultat du calcul de $I^k\varphi(L)$ à gauche par le gcld avec I^k, on retrouve le résultat de l'algorithme de Lewanowicz.

3.1 Algorithme de Paszkowski

Le point de départ de l'algorithme de Paszkowski est de réécrire L de (5.9) sous la forme (eqInv) comme ci-dessous

$$L = \sum_{i=0}^{k} \partial_x^i q_i(x). \tag{eqInv}$$

Les polynômes q_i peuvent être calculés par récurrence en commençant par $q_k = p_k$ et soustrayant $\partial_x^k q_k$ pour produire un opérateur d'ordre plus petit.

Alors

$$I^k\varphi(L) = \sum_{i=0}^{k} I^{k-i} q_i(X). \tag{5.10}$$

L'algorithme 5.2 s'en déduit.

3.2 Algorithme de Rebillard

Le point de départ de l'algorithme de Rebillard est l'identité

$$X_k = I^k X D^k = (2n)^{-1}((n+k)S_n + (n-k)S_n^{-1}),$$

Entrée: $L = \sum_{i=0}^{k} p_i(x)\partial_x^i$
Sortie: $I^k\varphi(L)$
  Calculer $p_i(X_k)$, $i = 0, \dots, k$
  $R := p_k(X_k)$
  pour tout i de 1 à k **faire**
    $R := R + p_{k-i}(X_k)I^i$
  fin pour
  renvoyer R

Algorithme 5.3: Algorithme de Rebillard

qui découle d'une simple récurrence. De là, il déduit

$$I^k\varphi(L) = \sum_{i=0}^{k} I^k p_i(X) D^k I^{k-i} = \sum_{i=0}^{k} p_i(X_k) I^{k-i}.$$

L'algorithme 5.3 s'en déduit.

3.3 Analyse de complexité

On donne maintenant une analyse de complexité des algorithmes de Paszkowski, Rebillard et Lewanowicz. Ceci révèle une source d'inefficacité pour des ordres grands, ce que l'on corrige dans notre nouvel algorithme dans la section suivante.

On a besoin de considérer les tailles des polynômes en deux variables n et S_n. On dit qu'un polynôme a un *bidegré* (d, k) en (n, S_n) quand il a un degré m en n et p en S_n.

D'abord, on analyse avec plus de précision la forme de I^i.

Proposition 5.14 (Rebillard [Reb98])**.** *Pour tout $i \in \mathbb{N}^*$,*

$$I^i = \frac{1}{r(i)}\left((n+1)_{i-1} S_n^{-i} + \sum_{k=1}^{i-1} s(k) S_n^{-i+2k} + (n-i+1)_{i-1} S_n^i\right),$$

où $r(i) = 2^i n \prod_{k=1}^{i-1}(n^2 - k^2)$,

$$s(k) = (-1)^k \binom{i}{k} (n-i+2k)(n+k+1)_{i-1-k}(n-i+1)_{k-1},$$

et on utilise le symbole de Pocchammer $(a)_i = a(a+1)\cdots(a+i-1)$.

En particulier, le bidegré de $r(i)I^i$ en (n, S_n) est $(i-1, 2i)$. La preuve est une fastidieuse mais simple récurrence que l'on omet ici. De cette formule se déduit une estimation précise des tailles des polynômes qui sont calculés.

Corollaire 5.15. *Si L dans (5.9) a un bidegré (d, k) en (x, ∂_x), alors $r(k)I^k\varphi(L)$ est un polynôme de bidegré en (n, S_n) au plus $(2k-1, 2(k+d))$.*

Démonstration. D'abord, L peut être réécrit comme dans l'algorithme de Paszkowski $\sum_{i=0}^{k} \partial_x^i q_i(x)$ avec $\deg q_i \leqslant d$. L'identité

$$r(k)I^k\varphi(L) = \sum_{i=0}^{k} \frac{r(k)}{r(i)}(r(i)I^i)q_{k-i}(X) \tag{5.11}$$

montre que c'est un polynôme en n. Chaque terme de cette somme est le produit entre un polynôme de bidegré $(2(k-i),0)$, un polynôme de bidegré $(i-1,2i)$ et un polynôme de bidegré au plus $(0,2d)$. Ainsi chaque sommant a un bidegré au plus $(2k-i-1,2i+2d)$, d'où le résultat. □

Proposition 5.16. *Étant donné L comme ci-dessus en entrée, l'algorithme de Paszkowski requiert $\mathcal{O}(dk^3)$ opérations arithmétiques.*

Démonstration. La première étape est le calcul des q_i à partir des p_i. La méthode récursive requiert seulement $\mathcal{O}(dk^2)$ opérations arithmétiques comme cela est montré dans la section suivante.

L'étape suivante est la boucle. Le principal coût de cette étape est la multiplication de I^i par $q_{k-i}(X)$. On multiplie le polynôme de bidegré $(i-1,2i)$, avec un polynôme en S_n seulement, de degré $2d$. Le coût de ces multiplications est $\mathcal{O}(i^2d)$ opérations arithmétiques. La somme de i jusqu'à k donne le résultat. □

Proposition 5.17. *Dans les mêmes conditions, l'algorithme de Rebillard requiert $\mathcal{O}(d^3k + d^2k^3)$ opérations arithmétiques.*

Démonstration. La première étape est le calcul des $p_i(X_k)$. Le polynôme X_k^i est de bidegré $(3i,2i)$ en (n,S_n). Ainsi chaque $p_i(X_k)$ peut-être calculé en $\mathcal{O}(d^3)$ opérations et l'ensemble en $\mathcal{O}(d^3k)$ opérations.

Le coût de la $i^{\text{ème}}$ étape de la boucle est dominé par le coût de la multiplication de $p_{k-i}(X_k)$ par I^i. Le polynôme $p_{k-i}(X^k)$ est de bidegré $(3d,2d)$ en (n,S_n), puisque I^i est de bidegré $(2i-1,2i)$. La multiplication naïve requiert alors $\mathcal{O}(d^2i^2)$ opérations. La somme jusqu'à k donne le résultat. □

Le résultat de l'algorithme de Lewanowicz est différent en général. On donne une comparaison dans les cas où il coïncide.

Proposition 5.18. *Dans les mêmes conditions, et si les gcrd durant son exécution sont triviaux, l'algorithme de Lewanowicz requiert $\mathcal{O}(dk^3)$ opérations arithmétiques.*

Démonstration. On donne seulement un aperçu. Comme tous les gcrd sont triviaux, il s'avère que le calcul des lclm et des cofactors sont de mêmes ordres de complexité que le calcul du produit Qp_i, où on a de plus $Q = I^{k-i}$. Les mêmes arguments que lors de l'analyse de l'algorithme de Paszkowski permettent de conclure. □

En fait les récurrences retournées par l'algorithme de Lewanowicz et de Paszkowski coïncide souvent. En effet la proposition 5.4 page 90 nous dit que pour que les récurrences ne coïncident pas, il faut que l'équation différentielle admette une singularité en ±1. Ce phénomène est assez rare.

4 Nouvel algorithme rapide

On donne maintenant un autre algorithme pour le calcul du même opérateur $I^k\varphi(L)$. L'élaboration de notre algorithme est motivée par sa complexité algorithmique. Cet algorithme calcule la récurrence de Tchebychev en $\mathcal{O}((d+k)k^{\omega-1})$ opérations arithmétiques. Il part du même point que celui de Paszkowski, c'est-à-dire de l'équation (eqInv) page 95. Le premier obstacle à la conception de l'algorithme rapide est donné justement par l'écriture de l'équation différentielle sous la forme (eqInv). Pour cette raison la première partie de cette section est la description d'un nouvel algorithme pour calculer une équation sous cette forme, cet algorithme effectue cette conversion en temps quasi-linéaire (en dk).

La seconde partie de cette section est la description du nouvel algorithme rapide pour le calcul de la récurrence.

4.1 Inversion polynômes-opérateurs dérivés

Pour que l'algorithme rapide ne soit pas dominé par l'inversion des polynômes et des opérateurs différentiels permettant d'obtenir l'équation (eqInv), on donne dans cette section deux algorithmes permettant de calculer cette transformation d'opérateurs. Dans le premier algorithme, on ramène la transformation à une multiplication de matrices qui a le même coût asymptotique que l'algorithme rapide pour le calcul d'une récurrence de Tchebychev présenté ci-dessous. En remarquant que le produit de matrices est en fait un produit avec une matrice structurée, on donne un algorithme de complexité quasi-optimale en la taille de la sortie. Cette sous-section est en partie issue d'un travail avec Alin Bostan et Joris van der Hoeven [BBvdH12].

Partant des coefficients $p_{i,j}$ de l'opérateur L, on souhaite calculer les coefficients $q_{i,j}$ tels que

$$L = \sum_{i<k, j<d} p_{i,j} x^j \partial_x^i, \quad \text{pour certains scalaires} \quad q_{i,j} \in \mathbb{K}.$$

Pour simplifier les notations, on dit que les coefficients $p_{i,j}$ et $q_{i,j}$ sont définis pour i et j dans $\mathbb{Z}$. On a donc $p_{i,j} = q_{i,j} = 0$ pour $i \notin \{0, \dots, k\}$ et $j \notin \{0, \dots, d\}$.

Une approche naïve du problème consiste à exprimer les coefficients $q_{i,j}$ en fonction des coefficients $p_{i,j}$. La règle de Leibniz sur les monômes donne

$$x^j \partial_x^i = \sum_{\ell=max(i-j,0)}^{i} (-1)^{i-\ell} \frac{j!}{(j-i+\ell)!} \binom{i}{\ell} \partial_x^\ell x^{j-i+\ell}. \tag{5.12}$$

En insérant cette équation dans l'opérateur L, on obtient :

$$L = \sum_{i=0}^{k}\sum_{j=0}^{d}\sum_{\ell=max(i-j,0)}^{i}(-1)^{i-\ell}\frac{j!}{(j-i+\ell)!}\binom{i}{\ell}p_{i,j}\partial_x^\ell x^{j-i+\ell}.$$

L'inversion des sommes et le décalage de j, $\ell - i$ fois nous donne :

$$L = \sum_{\ell=0}^{k}\sum_{i=\ell}^{k}\sum_{j=0}^{d-i+\ell}(-1)^{i-\ell}\frac{(j-\ell+i)!}{j!}\binom{i}{\ell}p_{i,j-\ell+i}\partial_x^\ell x^j.$$

Comme les coefficients $p_{i,j-\ell+i}$ sont nuls pour $j > d - i + \ell$, on ne change pas la somme en bornant à nouveau j par d. En inversant les indices i et ℓ et en procédant par extraction de coefficients, on a finalement

$$q_{i,j} = \sum_{\ell=i}^{k}(-1)^{\ell-i}\binom{\ell}{i}\frac{(j+\ell-i)!}{j!}p_{\ell,\ell-i+j}. \tag{5.13}$$

De cette formule se déduit immédiatement un algorithme pour calculer les coefficients $q_{i,j}$. Le calcul d'un coefficient $q_{i,j}$ en utilisant cette méthode s'effectue en $\mathcal{O}(k-i)$ opérations arithmétiques en utilisant la récurrence vérifiée par les factorielles (ce qui permet de calculer efficacement $(-1)^{\ell-i}\binom{\ell}{i}\frac{(j+\ell-i)!}{j!}$). On calcule de cette façon kd coefficients avec i allant de 0 à k. La complexité de cet algorithme est donc de $\mathcal{O}(k^2 d)$ opérations arithmétiques.

Cette complexité n'est pas satisfaisante pour notre problème puisque l'on a annoncé un calcul de la récurrence de Tchebychev en $\mathcal{O}((d+k)k^{\omega+1})$ opérations. On propose dans la suite un algorithme plus rapide afin d'effectuer cette inversion.

En multipliant par $j!$ le coefficient $q_{i,j}$ dans (5.13), on a :

$$j!q_{i,j} = \sum_{\ell=i}^{k}(-1)^{\ell-i}\frac{\ell!}{(\ell-i)!}\frac{(j+\ell-i)!}{i!}p_{\ell,\ell-i+j} = \sum_{\ell=i}^{k}(-1)^{\ell-i}\binom{\ell}{i}(j+\ell-i)!p_{\ell,\ell-i+j}.$$

En effectuant le changement $j = i + j$, on a l'égalité

$$(i+j)!q_{i,i+j} = \sum_{\ell=i}^{k}\left((-1)^{\ell-i}\binom{\ell}{i}\right)\quad\left((\ell+j)!p_{\ell,\ell+j}\right), \tag{5.14}$$

l'indice ℓ pouvant aller de 0 à k, $\binom{\ell}{i}$ étant nul pour $i > \ell$ et $p_{\ell,\ell+j}$ étant nul pour $\ell > k - j$. On reconnaît alors la formule de multiplication de matrices :

$$\begin{pmatrix} & & 0!q_{0,0} & \cdots & d!q_{0,d} \\ & \iddots & & \iddots & \\ 0!q_{0,k} & \cdots & d!q_{k,d} & & \end{pmatrix} = \begin{pmatrix} & & \\ & (-1)^{\ell+j}\binom{\ell-1}{j-1} & \\ & & \end{pmatrix}\begin{pmatrix} & & 0!p_{0,0} & \cdots & d!p_{0,d} \\ & \iddots & & \iddots & \\ 0!p_{k,0} & \cdots & d!p_{k,d} & & \end{pmatrix}. \tag{5.15}$$

On appelle P la matrice qui représente les coefficients de p, Q celle qui représente les coefficients de q et M_{Pa} (comme matrice de Pascal) celle qui représente les binomiaux. On

Entrée: $L \coloneqq \sum_{i=0}^{k} \sum_{j=0}^{d} p_{i,j} x^j \partial_x^i$
Sortie: $L \coloneqq \sum_{i=0}^{k} \sum_{j=0}^{d} q_{i,j} \partial_x^i x^j$
Remplir la matrice P, tel que $P_{i,j+d-i} \coloneqq (j-1)! p_{i-1,j-1}$
Remplir la matrice M_{Pa}, tel que $M_{Pa\,i,j} \coloneqq (-1)^{i+j} \binom{j-1}{i-1}$
Calculer la matrice Q, tel que $Q \coloneqq M_{Pa} \cdot P$
renvoyer Pour i et j allant de 0 à k et de 0 à d

$$q_{i,j} \coloneqq \frac{1}{j!} Q_{j+d-i+1,j+1}$$

Algorithme 5.4: Inversion polynômes opérateur de dérivé

peut remarquer que la partie haute gauche de la matrice Q n'est pas nulle. Ces coefficients ne nous intéressant pas, leurs valeurs peuvent être quelconques.

Il reste alors à énoncer la proposition suivante :

Proposition 5.19. *Soit $L = \sum_{i=0}^{k} p_i(x) \partial_x^i$ un opérateur différentiel linéaire à coefficients polynomiaux de degré d.*

L'algorithme 5.4 calcule les polynômes $q_i(x)$, tels que $L = \sum_{i=0}^{k} \partial_x^i q_i(x)$, en $\mathcal{O}(dk^{\omega-1})$ opérations arithmétiques.

Démonstration. La correction de l'algorithme est une conséquence immédiate de l'égalité (5.14). La complexité de l'algorithme vient de la multiplication des matrices M_{Pa} de dimension $(k+1, k+1)$ et de la matrice Q de dimension $(k+1, k+d+1)$. On peut couper la matrice Q (par colonnes) en d/k matrices de dimensions $(k+1, k+1)$. On se ramène alors à d/k produits de matrices carrés de dimensions $(k+1, k+1)$, ce qui s'effectue en $\mathcal{O}(dk)^{\omega-1}$ opérations arithmétiques. Remplir les matrices P et M_{Pa} s'effectue en un temps linéaire en la taille des matrices. Afin de remplir M_{Pa}, on peut utiliser la récurrence vérifiée par les binomiaux. Pour extraire les coefficients de la matrice Q, on va encore utiliser un nombre linéaire, en la taille de la matrice, d'opérations arithmétiques. □

Par ailleurs, on remarque que la matrice M_{Pa} est la matrice inverse de décalage d'un polynôme. C'est-à-dire la matrice permettant de passer d'un polynôme $p(x)$ à un polynôme $p(x-1)$. On peut donc réduire l'algorithme 5.4 à des appels à l'algorithme de décalage d'un polynôme comme le réalise l'algorithme 5.5.

On peut alors déduire la proposition suivante.

Proposition 5.20. *Soit $L = \sum_{i=0}^{k} p_i(x) \partial_x^i$ un opérateur différentiel linéaire à coefficients polynomiaux de degré plus petit d.*

L'algorithme 5.5 calcule les polynômes $q_i(x)$, tels que $L = \sum_{i=0}^{k} \partial_x^i q_i(x)$, en $\mathcal{O}((d+k)\mathsf{M}(k))$ opérations arithmétiques.

Démonstration. La preuve de la correction de cet algorithme se déduit de la représentation matricielle (5.14) des coefficients q_i, la matrice M_{Pa} étant la matrice représentant l'application permettant de passer d'un polynôme $p(x)$ au polynôme $p(x-1)$.

Entrée: $L := \sum_{i=0}^{k} \sum_{j=0}^{d} p_{i,j} x^j \partial_x^i$
Sortie: $L := \sum_{i=0}^{k} \sum_{j=0}^{d} q_{i,j} \partial_x^i x^j$
  pour tout j de 0 à $d+k+1$ **faire**
    $F_\ell = \sum_i \ell! p_{i+\ell,i} x^{i+\ell}$
    Calculer $G_\ell(x) = F_\ell(x-1) = \sum_i i! q_{i+\ell,i} x^{i+\ell}$ {Utiliser l'algorithme 3.2 du chapitre 3}
  fin pour
  renvoyer $\frac{1}{j!} q_{i,j}$

Algorithme 5.5: Inversion polynômes opérateur de dérivée à partir de l'algorithme de décalage

La seule étape non triviale de cet algorithme est le calcul des polynômes décalés $G_\ell(x)$. Les polynômes F_ℓ étant de degré plus petits que k, cette opération s'effectue en $\mathsf{M}(k)$ opérations arithmétiques [ASU75] (voir aussi algorithme 3.2 page 35). On en déduit le résultat final. □

Remarque 5.21. Lorsque k est grand devant d, on observe que l'algorithme 5.4 est de complexité k^ω et l'algorithme 5.5 est de complexité $k\mathsf{M}(k)$. Par exemple si d est constant, ces algorithmes sont moins bons que l'algorithme naïf.

Dans [BBvdH12], on montre qu'en transformant légèrement les algorithmes, dans le cas où $d > k$, on obtient des algorithmes de complexité respective $\mathcal{O}(kd\min(k,d)^{\omega-2})$ et $\mathcal{O}(\max(k,d)\mathsf{M}(\min(k,d)))$, donc de complexité toujours linéaire en la plus grande quantité.

4.2 Algorithme pour le calcul de récurrence

Dans les analyses des algorithmes de calcul de récurrence de Tchebychev précédentes, une grande partie des complexités vient du fait que durant les calculs, les bidegrés des polynômes intermédiaires grossissent linéairement et ces polynômes sont multipliés par des polynômes de degré constants. À la place de ces calculs, on propose de travailler avec des polynômes aux degrés équilibrés et donc d'utiliser les algorithmes rapides pour le produit d'opérateurs de récurrence [vdH02, BCLR08] décrits dans la section 4. Nous déduisons la complexité suivante.

Théorème 5.22. *L'algorithme 5.6 calcule l'opérateur de récurrence $I^k\varphi(L)$ en $\mathcal{O}((d+k)k^{\omega-1})$ opérations arithmétiques.*

Ici, ω est l'exposant réalisable pour la multiplication de matrices dans $\mathbb{Q}$ (section 2.2). On prouve maintenant ce résultat.

Entrée: Polynômes $a_0(x), \dots, a_k(x)$
Sortie: $P_{(0,\dots,k)} = \sum_{i=0}^{k} I^i a_i(X)$
  si $k = 0$ **alors**
    renvoyer $a_0(X)$
  sinon
    $\ell := \lceil k/2 \rceil$
    Calculer récursivement $P_{(0,\dots,\ell-1)}$ et $P_{(\ell,\dots,k)}$.
    renvoyer $P_{(0,\dots,\ell-1)} + I^\ell P_{(\ell,\dots,k)}$.
  finsi

Algorithme 5.6: Diviser pour régner pour la récurrence de Tchebychev

En partant de (eqInv), on écrit

$$\begin{aligned}\sum_{i=0}^{k} I^i q_{k-i}(X) &= \sum_{i=0}^{\ell-1} I^i q_{k-i}(X) + I^\ell \sum_{i=\ell}^{k} I^{i-\ell} q_{k-i}(X) \\ &=: P_{(0,\cdots,l-1)} + I^l P_{(l,\cdots,k)}.\end{aligned}$$

On choisit $\ell = \lceil k/2 \rceil$ et on applique la même idée récursivement. La preuve de cet algorithme se réalise en deux étapes

La partie utilisant le plus de temps est le produit $I^\ell P_{(\ell,\dots,k)}$, pour lequel on donne un algorithme spécial.

Pour simplifier la présentation, supposons que $k = 2\ell$. Le corollaire 5.15 implique que I^ℓ a un degré 2ℓ en S_n, $P_{(\ell,\dots,k)}$ a un degré au plus $2\ell + 2d$ en S_n. Leurs coefficients sont des fractions rationnelles dont les degrés des numérateurs et dénominateurs sont aussi bornés respectivement par 2ℓ et $2\ell + 2d$. Si d est grand, les degrés en S_n sont déséquilibrés, donc on décompose d'abord

$$P_{(\ell,\dots,k)} = A_0(n, S_n) S_n^{-d-\ell} + A_1(n, S_n) S_n^{-d+\ell+1} + \cdots, \tag{5.16}$$

où les A_i ont des degrés au plus 2ℓ en S_n. Notons que cette décomposition est seulement une extraction des coefficients et n'utilise aucune opération arithmétique. On se ramène donc au produit des I^ℓ avec les A_i. Les deux opérateurs ont des coefficients qui sont des fonctions rationnelles et donc ne peuvent pas être directement multipliés en utilisant l'algorithme de multiplication rapide 4.4 page 65. On sait par ailleurs que $r(2\ell) I^\ell P_{(\ell,\dots,k)}$ a des coefficients polynomiaux en n de degré au plus $k-1$, ce qui est donc vrai aussi pour $r(2\ell) I^\ell A_i$.

Pour effectuer le produit rapidement, on utilise le fait que l'algorithme 4.4 page 65 procède par évaluation et interpolation : durant la phase d'évaluation, on évalue les coefficients *rationnels* de A_i comme s'ils étaient des polynômes (et avec la même complexité grâce à notre borne des degrés), en évitant les zéros $-\ell, \dots, \ell$ de leur dénominateur ; de façon similaire, on évalue les coefficients polynomiaux de $r(2\ell) I^\ell$.

L'algorithme 4.4 permet la multiplication de deux opérateurs dans $\mathbb{K}[n]\langle S_n \rangle$. Ici on souhaite multiplier deux polynômes de Laurent, pour y arriver, on multiplie ces polynômes

par des monômes afin de les ramener dans $\mathbb{K}[n]\langle S_n\rangle$. La décomposition (5.16) donne des polynômes A_i de valuation $-\ell$. On sait par ailleurs que l'opérateur I^l a aussi comme valuation $-\ell$. On multiplie donc les opérateurs $S_n^{2\ell} r(2\ell) I^\ell S_n^{-\ell}$ et $S_n^\ell A_i$, ce qui nous permet d'avoir des polynômes avec une valuation non négative.

Le degré en n de l'opérateur $r(2l)I^\ell A_i$ est, selon le corollaire 5.15, égal à $4\ell - 1$, le degré en n de l'opérateur $r(\ell)I^\ell$ est $\ell - 1$, comme $r(2\ell)/r(\ell)$ est un polynôme de degré ℓ, l'opérateur $r(2\ell)I^\ell$ a un degré $2\ell - 1$. En reprenant les notations de l'algorithme 4.4, on représente l'opérateur

$$S_n^\ell A_i = S_n^\ell \frac{1}{r(\ell)(n)} \sum_{i=-\ell}^{\ell} a_i(n) S_n^i = \frac{1}{r(\ell)(n+\ell)} \sum_{i=0}^{2\ell} a_i(n+\ell) S_n^i$$

par la matrice :

$$M_{6\ell}^{A_i} = \begin{pmatrix} \frac{1}{r(\ell)}(\ell) \\ \frac{1}{r(\ell)}(\ell+1) \\ \vdots \\ \frac{1}{r(\ell)}(7\ell) \end{pmatrix}^t \cdot \begin{pmatrix} a_0(\ell) & \cdots & a_{2\ell}(\ell) & & \\ & a_0(\ell+1) & \cdots & a_{d_A}(\ell) & \\ & & \ddots & & \ddots \\ & & & a_0(7\ell) & \cdots & a_{2\ell}(7\ell) \end{pmatrix}.$$

L'opérateur

$$S_n^{2\ell} r(2\ell) I^\ell S_n^{-\ell} = S_n^{2\ell} r(2\ell)(n) \sum_{i=0}^{2\ell} \frac{I_{\ell,i}}{r(\ell)}(n) S_n^{i-\ell} = \frac{r(2\ell)}{r(\ell)}(n+2\ell) \sum_{i=0}^{2\ell} I_{\ell,i}(n+2\ell) S_n^i$$

est représenté par la matrice :

$$M_{4\ell}^{I^\ell} = \begin{pmatrix} \frac{r(2\ell)}{r(\ell)}(2\ell) \\ \frac{r(2\ell)}{r(\ell)}(2\ell+1) \\ \vdots \\ \frac{r(2\ell)}{r(\ell)}(6\ell) \end{pmatrix}^t \cdot \begin{pmatrix} I_{\ell,0}(2\ell) & \cdots & I_{\ell,2\ell}(2\ell) & & \\ & I_{\ell,0}(2\ell+1) & \cdots & I_{\ell,2\ell}(2\ell+1) & \\ & & \ddots & & \ddots \\ & & & I_{\ell,0}(6\ell) & \cdots & I_{\ell,2\ell}(6\ell) \end{pmatrix}.$$

La multiplication des deux matrices $M_{4\ell}^{I^\ell}$ et $M_{6\ell}^{A_i}$ permet d'obtenir la matrice représentant l'opérateur de récurrence $S_n^{2\ell} r(2\ell) I^\ell A_i$. Pour obtenir l'opérateur à partir de cette matrice, il suffit d'interpoler les coefficients de la matrice $M_{4\ell}^{I^\ell} M_{6\ell}^{A_i}$ comme lors de l'algorithme 4.4. Pour obtenir l'opérateur $r(2\ell)I^\ell A_i$, au lieu d'interpoler aux points 0 jusqu'à 4ℓ, on interpole les polynômes aux points -4ℓ jusqu'à 0. L'algorithme 5.7 résume ce procédé.

La complexité de chacune de ces multiplications est donc $\mathcal{O}(\ell^\omega)$ opérations arithmétiques. L'algorithme 5.8 calcule la récurrence de Tchebychev en renvoyant l'opérateur de récurrence $I^\ell P$. Notons qu'un facteur constant peut être préservé en ne recalculant pas l'évaluation de $r(2\ell)I^\ell$ à chaque fois.

Proposition 5.23. *Le coût pour multiplier I^ℓ par $\sum_{i=0}^{\ell} I^i a_i(X)$ avec $\deg a_i \leqslant d$ en utilisant l'algorithme 5.8 est $\mathcal{O}((\ell+d)\ell^{\omega-1})$ opérations arithmétiques.*

Entrée: I^ℓ et A_i
Sortie: $r(2\ell)I^\ell A_i$
Calculer les matrices $M_{4\ell}^{I^\ell}$et $M_{6\ell}^{A_i}$
Calculer la matrice $M_{4\ell}^{I^\ell}M_{6\ell}^{A_i}$
Retrouver $r(2\ell)I^\ell A_i$ en interpolant cette matrice aux points -4ℓ jusqu'à 0
renvoyer $r(2\ell)I^\ell A_i$.

Algorithme 5.7: Multiplication rapide de $I^\ell A_i$

Entrée: I^ℓ et $P := \sum_{i=0}^{\ell} I^i a_i(X)$
Sortie: $I^\ell P$
décomposer P comme dans l'équation (5.16)
$R := 0$
pour tout i de 0 à $\lfloor(\ell+d)/\ell\rfloor$ **faire**
$R := R + \text{multiplicationrapide}(r(2\ell)I^\ell, A_i)S_n^{-d-\ell+i(k+1)}$ {Utiliser l'algorithme 5.7}
fin pour
renvoyer $1/r(2l)R$.

Algorithme 5.8: Multiplication rapide pour la récurrence de Tchebychev

Démonstration. Nous avons vu qu'avec l'algorithme 5.7, chaque multiplication $r(\ell)I^\ell A_i$ a comme complexité $\mathcal{O}(\ell^\omega)$. Ceci est effectué $(\ell+d)/\ell$ fois. La multiplication à droite par des puissances de S_n n'utilise pas d'opération arithmétique. Les additions requièrent un plus petit nombre d'opérations, d'où le résultat. □

Maintenant, soit $T(k,d)$ la complexité de l'algorithme 5.6. En utilisant cette proposition, nous avons

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

la complexité estimée dans le théorème 5.22 se déduit de la convergence des séries géométriques. À nouveau, un facteur constant peut être économisé en ne calculant les puissances de I qu'une seule fois.

5 Solutions de la récurrence de Tchebychev

Si f est une série de Tchebychev D-finie et analytique sur le segment $[-1,1]$, on a vu que ses coefficients sont solutions d'une récurrence de Tchebychev associée à l'équation différentielle. Le corollaire 5.15 nous dit que si l'équation différentielle est d'ordre k, la récurrence est au moins d'ordre $2k$. Il existe donc des solutions de la récurrence qui ne sont pas les coefficients d'une série de Tchebychev solution de l'équation différentielle. Le but de cette section est de déterminer l'ensemble des solutions de cette récurrence et de caractériser celles qui sont des suites de coefficients de série de Tchebychev solution de l'équation différentielle.

Cette section est une partie du travail commun avec Mioara Joldeş et Marc Mezzarobba[BJM].

5.1 Intersection entre l'espace des coefficients des séries et l'espace des solutions de la récurrence

Le théorème 5.5 page 91 permet de caractériser la récurrence satisfaite par les coefficients d'une série de Tchebychev solution d'une équation différentielle comme numérateur d'une fraction d'opérateurs. On a vu que pour des raisons de dimension de l'espace des solutions de la récurrence, les suites solutions de la récurrence ne sont pas toujours la suite des coefficients d'une série de Tchebychev solution de l'équation différentielle associée. Le théorème suivant propose quand même une réciproque à ce théorème si la suite des coefficients solution de la récurrence vérifie certaines propriétés.

Théorème 5.24. *Soient u et v des fonctions analytiques sur un voisinage complexe du segment $[-1,1]$, admettant les développements de Tchebychev*

$$u(x) = \sum_{n=0}^{\infty}{}' u_n T_n(x), \quad v(x) = \sum_{n=0}^{\infty}{}' v_n T_n(x).$$

Si L est un opérateur différentiel et $Q^{-1}P := \varphi(L)$, alors l'équation différentielle $L \cdot u(x) = v(x)$ est satisfaite si et seulement si

$$P \cdot (u_n) = Q \cdot (v_n). \tag{5.17}$$

Démonstration. Le seulement si de cette proposition est déjà énoncée dans le théorème 5.5 page 91. Pour le si, considérons deux suites $u, v \in C$ (C est l'espace vectoriel des suites bi-infinies symétriques à convergence exponentielle comme défini dans 2.14 page 24) telles que $P \cdot u = Q \cdot v$, et soit la suite $y \in C$ définie par

$$L \cdot u(x) = \sum_{n=0}^{\infty}{}' y_n T_n(x).$$

D'après le théorème 5.5, on a donc $P \cdot u = Q \cdot y$, mais alors $Q \cdot v = Q \cdot y$, et l'on conclut que $v = y$ par le lemme suivant. □

Lemme 5.25. *Restreint à l'espace vectoriel C des suites bi-infinies symétriques à convergence exponentielle, l'opérateur P_2 défini comme dans le théorème 5.5 page 91 est injectif.*

Démonstration. La fonction f est analytique sur le segment $[-1,1]$, la proposition 5.4 page 90 implique donc que si l'équation différentielle est d'ordre k, on a :

$$P_2 = r(k)I^k,$$

en utilisant les notations de la proposition 5.14 page 96. Montrons par récurrence sur l'ordre de l'équation différentielle $k \geqslant 1$ que

$$(v \in C) \wedge \big(|n| \geqslant k \implies (Q_k \cdot v)_n = 0\big) \implies v = 0. \tag{5.18}$$

Premièrement, on a $(\ker r(1)I) \cap C = \{0\}$ $(r(1)I = (S_n^{-1} - S_n))$ puisque toute suite de C converge vers 0 quand $n \to \pm\infty$. Maintenant supposons (5.18) vérifiée, et soit $v \in C$ telle que $(r(k+1)I^{k+1} \cdot v)_n = 0$ pour $|n| \geqslant k+1$. On remarque que la suite w, définie par

$$w = r(k+1)I^{k+1} \cdot v,$$

est un élément de C. En effet, comme par hypothèse $w_n = 0$ pour $|n| > k+1$, w est bien à convergence exponentielle. De plus comme v est symétrique, $I \cdot v = \frac{1}{2n}\left(S_n^{-1} - S_n\right) \cdot v$ est aussi symétrique, en itérant on montre alors que $I^k \cdot v$ est symétrique. Par sa définition dans la proposition 5.14, on sait que le polynôme $r(k+1)$ est symétrique, donc $w = r(k+1)I^k \cdot v$ est bien symétrique.

Comme $k \geqslant 1$, on a

$$\begin{aligned} nr(k+1)I^{k+1} &= r(k+1)(S_n^{-1} - S_n)I^k \\ &= ((n+k)(n+k-1)S_n^{-1}r(k) - (n-k)(n-k+1)S_n r(k))I^k \\ &= ((n+k)(n+k-1)S_n^{-1} - (n-k)(n-k+1)S_n)r(k)I^k \end{aligned}$$

d'où pour $|n| \geqslant k+1$,

$$(n+k)(n+k-1)w_{n-1} = (n-k)(n-k+1)w_{n+1}. \tag{5.19}$$

À moins que w_n ne soit ultimement nulle, il s'ensuit que $w_{n+1}/w_{n-1} \to 1$ lorsque $n \to \infty$, ce qui est incompatible avec le fait que $w \in C$. Il suit que $w_n = 0$ pour $|n|$ grand, et en utilisant encore (5.19), que $w_n = 0$ dès que $|n| \geqslant k$. On conclut en appliquant l'hypothèse (5.18). □

5.2 Symétrie de l'espace des solutions symétriques

Une autre propriété des récurrences de Tchebychev qui sera utilisée dans la suite est donnée par la proposition suivante :

Proposition 5.26 ([Reb98])**.** *Si L est un opérateur différentiel n'admettant pas de singularité en 1 et −1, l'opérateur P défini comme dans le théorème 5.5 page 91 s'écrit comme :*

$$P = \sum_{i=-s}^{s} b_i(n) S_n^i,$$

où $b_i(n) = -b_{-i}(-n)$.

Remarque 5.27. D'après la proposition 5.26 et avec ces notations, on a pour toute suite u annulée par P :

$$P \cdot u_{-n} = \sum_{i=-s}^{s} b_i(n) u_{-n+1} = -\sum_{i=-s}^{s} b_{-i}(-n) u_{-n+i} = (P \cdot u)_{-i} = 0.$$

Donc si (u_n) est annulée par l'opérateur P, alors (u_{-n}) l'est aussi. Toute solution (u_n) donne donc naissance à une solution symétrique $(u_n + u_{-n})$. N'importe quelle solution n'est pas pour autant symétrique.

Une autre remarque intéressante suit cette proposition. À la différence de ce qui se passe avec les récurrences des séries de Taylor, les coefficients de queue et de tête $b_{\pm s}$ de la récurrence (5.17) peuvent s'annuler pour des valeurs arbitrairement grandes de n, même si l'équation différentielle est non singulière. Les zéros de b_s sont appelés les *singularités de tête* de (5.17), et ceux de b_{-s}, les *singularités de queue*. Dans le cas des récurrences de Tchebychev, les singularités de tête et de queue sont des valeurs opposées.

Exemple 5.28. Pour tout $i \in \mathbb{Z}$, la récurrence de Tchebychev associée à l'équation différentielle $y''(x) + xy'(x) + iy(x) = 0$, à savoir

$$(n+1)(n+i-2)\,u_{n-2} - 2n(-2n^2 - i + 1)\,u_n - (n-1)(n-i+2)\,u_{n+2} = 0,$$

présente une singularité de tête en $n = i - 2$ et une singularité en queue en $n = -i + 2$.

On dispose néanmoins d'un contrôle partiel sur les singularités.

Proposition 5.29. *Avec les notations du théorème 5.26, les coefficients de la récurrence de Tchebychev, associée à une équation différentielle d'ordre k donnée par l'algorithme de Paszkowski satisfont les relations*

$$b_{j-i}(-j) = -b_{j+i}(-j), \qquad |j| \leqslant k-1, \quad i \in \mathbb{N}, \tag{5.20}$$

avec $b_j = 0$ pour $|j|$ strictement plus grand que l'ordre de la récurrence.

Démonstration. On procède par récurrence sur k. Lorsque $j = 0$, l'assertion (5.20) se ramène à $b_{-i}(0) = -b_i(0)$, laquelle se déduit de la proposition 5.26. En particulier, ceci prouve le résultat pour $k = 1$. Maintenant soit $k \geqslant 2$ et supposons que la proposition est vraie lorsque L est d'ordre $k-1$. On écrit $L = \hat{L} + \partial^k p_k(x)$ où $p_k \in \mathbb{Q}[x]$ et $\hat{L}$ est un opérateur différentiel d'ordre $k-1$. On définit $\hat{P} = \sum_{k\in\mathbb{Z}} \hat{b}_k(n) S_n^k$ comme l'opérateur de la récurrence de Tchebychev associé à $\hat{L}$. On a alors selon la formule (5.11) page 97 :

$$r(k)^{-1} P = Ir(k-1)^{-1}\hat{P} + p_k(\tfrac{1}{2}(S_n + S_n^{-1})). \tag{5.21}$$

Comme

$$Ir(k-1)^{-1} = (nr(k))^{-1}((n-k+2)(n-k+1)S_n^{-1} - (n+k-2)(n+k-1)S_n)$$

par la règle de commutation de l'opérateur de décalage, la relation (5.21) se réécrit comme

$$\begin{aligned} P = \frac{1}{n}\sum_l \Big(&(n-l+2)(n-l+1)\hat{b}_{l+1}(n-1) \\ &- (n+l-2)(n+l-1)\hat{b}_{l-1}(n+1)\Big) S_n^l \\ &+ r(l) p_k\left(\frac{1}{2}(S_n + S_n^{-1})\right). \end{aligned}$$

Le cas $j = 0$ a déjà été vérifié, on suppose donc $0 < |j| < k$. Comme $r(k)(-j) = 0$ et p_k est un polynôme, il suit par extraction du coefficient S_n^l dans la dernière égalité et en évaluant à $n = -j$ que

$$-jb_l(-j) = (j+k-2)(j+k-1)\hat{b}_{l+1}(-j-1) - (j-k+2)(j-k+1)\hat{b}_{l-1}(-j+1). \quad (5.22)$$

Maintenant $\hat{b}_{j-i}(-j) = -\hat{b}_{j+i}(-j)$ pour $|j| < k-1$ par hypothèse de récurrence, et le terme invoquant $\hat{b}_{l\pm1}$ s'annule pour $j = \pm(k-1)$ et $j = \mp(k-2)$. Dans chaque cas, on obtient $b_{j-i}(-j) = b_{j-i}(-j)$. □

Cette observation aura de lourdes conséquences sur la structure d'une récurrence de Tchebychev, notamment avec les corollaires suivants.

Corollaire 5.30. *Si $(u_n)_{n\in\mathbb{Z}}$ est une suite symétrique, on a $(P\cdot u)_n = 0$ pour tout $|n| < k$.*

Démonstration. On a

$$(P\cdot u)_n = \sum_{\ell\in\mathbb{Z}} b_\ell(n)u_{n+\ell} = \sum_{i\in\mathbb{Z}} b_{i-n}(n)u_i$$

or la proposition 5.29 avec $j = -n$ et $|n| < k$ et l'égalité $u_i = u_{-i}$ entraînent

$$\sum_{i\in\mathbb{Z}} b_{i-n}(n)u_i = -\sum_{i\in\mathbb{Z}} b_{-i-n}(n)u_i = -\sum_{i\in\mathbb{Z}} b_{i-n}(n)u_i$$

autrement dit $(P\cdot u)_n = -(P\cdot u)_n$. □

Corollaire 5.31. *La dimension de l'espace des solutions symétriques d'une récurrence de Tchebychev d'ordre $2s$ associée à un opérateur différentiel d'ordre k est au moins $s+k$.*

Démonstration. Soit $P = \sum_{i=-s}^{s} b_i(n)S_n^i$ un opérateur de récurrence de Tchebychev d'ordre $2s$ obtenue par l'algorithme de Paszkowski à partir d'une équation différentielle d'ordre k.

On se fixe $s+k$ coefficients $(u_0,\dots,u_{s+k-1})$. On veut montrer qu'il existe une suite symétrique annulée par P dont les termes d'indices compris entre 0 et $s+k-1$ sont égaux à u_i.

Soit $(y_n)_{n\in\mathbb{Z}}$ une suite qui satisfait $y_i = \frac{u_i}{2}$ pour tout $0 \leqslant |i| < s+k$ et $P\cdot y_n = 0$ pour tout $|n| \geqslant k$. Par la forme de P, cette suite existe toujours.

On a alors pour tout $|n| \geqslant k$

$$\sum_{i=-s}^{s} b_i(n)y_{-n+i} = \sum_{i=-s}^{s} b_{-i}(-n)y_{-n+i} = -\sum_{i=-s}^{s} b_i(-n)y_{-n-i} = (P\cdot y)_{-n} = 0,$$

On a donc $P\cdot y_{-n} = 0$ pour $|n| \geqslant k$, il s'ensuit que $P\cdot(y_n + y_{-n}) = 0$.

D'autre part le corollaire 5.30 nous indique la symétrie de la suite $y_n + y_{-n}$ entraine que $P\cdot(y_n + y_{-n})$ pour $|n| < k$.

On a donc montrer que $y_n + y_{-n}$ est une suite symétrique, solution de P pour tout n et vérifie $y_n + y_{-n} = u_n$ pour $|n| < s+k$.

□

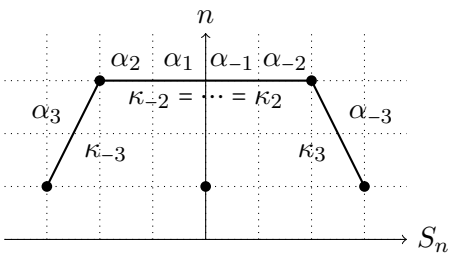


FIGURE 5.1 – Forme typique du polygone de Newton d'une récurrence de Tchebychev.

On n'a pas encore ici déterminé exactement la dimension de l'espace des solutions symétriques, elle sera donnée dans la section suivante.

5.3 Solutions convergentes et divergentes

Une autre caractéristique importante des récurrences de Tchebychev est qu'elles admettent des solutions divergentes, qui ne correspondent pas à des développements de solutions de l'équation différentielle dont elles sont issues (cf Rebillard [Reb98, chap. 5]).

Exemple 5.32. Dans l'exemple 5.2, on a vu la récurrence de Tchebychev associée à la fonction exponentielle. Cette récurrence est d'ordre 2 et l'une des solutions est la suite $(\mathrm{I}_n(1))_{n\in\mathbb{Z}}$ qui est la suite des coefficients de Tchebychev de la fonction exponentielle. Une autre solution de cette récurrence est la suite des fonctions de Bessel modifiées de deuxième espèce $(\mathrm{K}_n(1))_{n\in\mathbb{Z}}$. La suite $(\mathrm{I}_n(1))_{n\in\mathbb{Z}}$ converge vers zéro comme $\Theta(\frac{2^{-n}}{n!})$. L'autre suite diverge comme $\Theta(2^{n-1}(n-1)!)$.

Cet exemple illustre le comportement typique des solutions divergentes de la récurrence. L'analyse des germes des solutions à l'infini donnée par l'étude du polygone de Newton de la récurrence et l'interprétation de ce polygone par le théorème de Perron-Kreuser nous donne résultats plus précis sur ces solutions divergentes.

Le polygone de Newton d'un opérateur de récurrence de Tchebychev P (voir figure 5.1) est l'enveloppe supérieure convexe des points $A_l = (l, \deg p_l) \in \mathbb{R}^2$. À chaque arête $E = [A_i, A_j]$ $(i < j)$ du polygone est associée l'équation caractéristique $\sum_{A_k\in E} \mathrm{lc}(p_l)\alpha^{l-i}$, où $\mathrm{lc}(p)$ est le coefficient de tête de p. On dénote par $\alpha_s, \ldots, \alpha_1, \alpha_{-1}, \ldots, \alpha_s$ les racines de ces équations, les arêtes étant parcourues de gauche à droite et les racines des équations caractéristiques d'une même arête par module croissant. Soient $-\kappa_s, \ldots, -\kappa_1, -\kappa_{-1}, \ldots, -\kappa_{-s}$ les pentes des arêtes correspondantes. Le théorème de Perron-Kreuser [Gue63] assure que la récurrence $P \cdot u = 0$ annule $2s$ suites définies à partir d'un certain rang, linéairement indépendantes, de comportement asymptotique à l'infini décrit par les couples (κ_i, α_i). Il est plus commode ici de donner un énoncé précis en termes de germes de solutions. Rappelons la définition.

Définition 5.33. On appelle *germe de suite* au voisinage de $+\infty$ une classe d'équivalence de suites $(u_n)_{n\in\mathbb{N}}$ définies à partir d'un certain rang modulo identification des suites qui coïncident sur l'intersection de leurs domaines de définitions. On appelle *germe de solution* d'une récurrence linéaire un germe de suite dont les représentants satisfont la récurrence sur leur domaine de définition.

Dans le cas d'une récurrence à coefficients polynomiaux, les singularités étant en nombre fini, les germes de solutions en $+\infty$ s'identifient aux solutions définies à partir de n'importe quel rang N dépassant la plus grande singularité de tête de la récurrence. L'espace des germes de solutions en $+\infty$ est donc de dimension égale à l'ordre de la récurrence. Le comportement asymptotique d'un germe en $+\infty$ est défini sans ambiguïté.

Le résultat du théorème de Perron-Kreuser est que la récurrence $P \cdot u = 0$ admet une base $(e_1, \dots, e_s, e_{-1}, \dots, e_{-s})$ de germes de solution à l'infini tels que

$$\limsup \left| \frac{e_{i,n}}{n!^{\kappa_i}} \right| = |\alpha_i|$$

pour tout i.

Par les propriétés de symétrie des coefficients de P,

$$\kappa_{-i} = -\kappa_i \quad \text{et} \quad |a_{-i}| = |\alpha_i|^{-1} .$$

De l'existence de cette base, on déduit la proposition suivante.

Proposition 5.34. *La dimension de l'espace des solutions convergentes d'une récurrence de Tchebychev d'ordre* $2s$ *est* s*.*

De cette proposition, on peut déduire le résultat fondamental suivant sur les solutions des récurrences de Tchebychev.

Théorème 5.35. *La dimension de l'espace des solutions symétriques d'une récurrence de Tchebychev d'ordre* $2s$ *associée à une équation différentielle d'ordre* k *est* $s + k$*.*

Ce théorème sera central pour le calcul des coefficients dans le chapitre 7.

Démonstration. Dans le corollaire 5.31, on a vu que la dimension est au moins $s + k$. Il reste à montrer que cette dimension est au plus $s + k$. Soit L un opérateur différentiel de degré k et soit P un opérateur de récurrence de degré $2s$ obtenu par l'algorithme de Paszkowski à partir L. Selon le théorème 5.24, si une suite symétrique et convergente y_n est annulée par P, la série de Tchebychev $\sum y_n T_n(x)$ est annulée par L. La dimension de l'espace des solutions de L est k, alors la dimension de l'espace des solutions convergentes et symétriques est k. De plus, la dimension de l'espace des solutions convergentes de P est s, on déduit que la co-dimension de l'espace des solutions convergentes symétriques est au moins $s - k$, alors la co-dimension de l'espace des solutions symétriques est au moins $s - k$. Vu la dimension de l'espace des solutions de P, on déduit que la dimension de l'espace des solutions symétriques est au plus $s + k$. □

L'ensemble de ces propriétés nous donne une idée précise du comportement solutions de la récurrence de Tchebychev. Ces comportements seront exploités pour calculer numériquement les coefficients à partir de la récurrence dans le chapitre 7.

Chapitre 6

Formes closes et calcul numérique des coefficients de Tchebychev

Résumé

Ce chapitre expose une méthode permettant de représenter chaque coefficient de Tchebychev d'une fonction f comme l'évaluation d'une série entière en 1.

De cette représentation sont déduites des nouvelles preuves de formules donnant des formes closes pour les coefficients de Tchebychev de fonction hypergéométrique. Un nouvel algorithme de calcul d'approximations à précision arbitraire des coefficients de Tchebychev de fonction D-finie en temps linéaire obtenu par cette méthode est aussi présenté.

Sommaire

1 Introduction

Les fonctions élémentaires et spéciales qui sont solutions d'équations différentielles admettent souvent des développements en séries de Taylor « jolis » et simples. Les développements suivants en sont des exemples :

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \quad \arcsin(x) = \sum_{n=0}^{\infty} \frac{(2n)!x^{2n+1}}{4^n (2n+1) n!^2}, \quad \operatorname{erf}(x) = \sum_{n=0}^{\infty} 2\frac{2(-1)^n x^{2n+1}}{\sqrt{\pi}(2n+1) n!}.$$

Ces formules se déduisent naturellement des équations différentielles vérifiées par les fonctions. En effet, pour toutes ces séries, les récurrences vérifiées par les coefficients de Taylor sont à deux termes, c'est-à-dire de la forme :

$$a_0(n)u_n + a_d(n)u_{n+d} = 0.$$

Par exemple les récurrences vérifiées par les coefficients de Taylor de exp et erf sont respectivement :

$$u_n - (n+1)u_{n+1} = 0 \quad \text{et} \quad 2nu_n + \left(n^2 + 3n + 2\right)u_{n+2} = 0.$$

Les formes closes des coefficients s'obtiennent directement à partir de ces récurrences et de leurs conditions initiales.

Les mêmes fonctions développées en séries de Tchebychev admettent aussi de jolies formules. Ainsi, pour la fonction exponentielle [Sny66, MH03] on a [1]

$$\exp(x) = {\sum_{n=0}^{\infty}}' 2\,\mathrm{I}_n(1)T_n(x). \tag{6.1}$$

Cette formule se déduit de la représentation intégrale de la fonction de Bessel modifiée de première espèce $\mathrm{I}_n(x)$ [Nat10, http://dlmf.nist.gov/10.32.E3] :

$$\mathrm{I}_n(x) = \frac{1}{\pi}\int_0^{\pi} e^{x\cos(\theta)}\cos(n\theta)\,d\theta,$$

suivie du changement de variable $t = \cos(\theta)$ (en utilisant l'égalité $T_n(\cos(\theta)) = \cos(n\theta)$).

D'autres se calculent en résolvant la récurrence vérifiée par les coefficients de Tchebychev de la fonction, comme la fonction inverse du sinus qui admet une suite de coefficients de Tchebychev hypergéométrique :

$$\arcsin(x) = \sum_{n=0}^{\infty} \frac{4}{\pi(2n+1)^2} T_{2n+1}(x). \tag{6.2}$$

La fonction d'erreur admet aussi un joli développement en série de Tchebychev

$$\operatorname{erf}(x) = 2\sum_{n=0}^{\infty} \frac{4^{-n}(-1)^n}{\sqrt{\pi}(2n+1)n!}\,{}_1F_1\left(\begin{matrix}\frac{1}{2}+n\\ 2n+2\end{matrix}\middle| -1\right) T_{2n+1}(x), \tag{6.3}$$

1. La signification de la notation $\sum'$ est expliquée dans 2.16 page 23

où ${}_1F_1$ est la série hypergéométrique généralisée (voir définition 2.8 page 21).

Pour calculer ces formes closes, une première approche est d'utiliser la récurrence de Tchebychev vérifiée par les coefficients. Le chapitre 5 donne des algorithmes pour calculer ces récurrences. Cette idée permet par exemple d'obtenir la forme close des coefficients de Tchebychev de $\arcsin(x)$, puisque la récurrence est alors à deux termes :

$$\left(n^2+4n+4\right)u_{n+2}-n^2u_n=0.$$

Pour les fonctions exp et erf, les récurrences sont à trois termes. Par exemple pour la fonction erf, la récurrence obtenue est :

$$(n^2+3n)u_n+(2n^3+12n^2+24n+16)u_{n+2}-(n^2+5n+4)u_{n+4}=0. \tag{6.4}$$

Des algorithmes à base de transformations de jauge permettent de réduire cette récurrence à une autre récurrence dont on connait la solution. On en déduit alors base de solution sous forme close (voir [CvHL10]). Afin de déduire de cette base de solution les coefficients de Tchebychev sous formes closes, on doit lier cette base de solution avec les conditions initiales de la récurrence. Pour calculer ces conditions initiales, on doit résoudre l'intégrale donnant les coefficients de Tchebychev sous forme close ; à notre connaissance, il n'existe pas d'algorithme pour effectuer ce calcul.

Le problème qui nous motive dans ce chapitre est le calcul automatique de ces formes closes. Pour résoudre ce problème, on développe une nouvelle méthode pour calculer les coefficients de Tchebychev sans utiliser la récurrence vérifiée par ces coefficients. En plus d'un calcul automatique donnant des formes closes, cette méthode nous donnera aussi un algorithme rapide pour calculer des approximations des coefficients de Tchebychev.

1.1 Idée de base : une généralisation des coefficients de Tchebychev

L'idée de départ de ce chapitre est de généraliser le développement en série d'une fonction f en développant la fonction $f(xt)$ comme :

$$f(xt)=\sum_{k\in\mathbb{N}}{}'c_k(t)T_k(x), \tag{6.5}$$

où les fonctions $c_k(t)$ sont données par l'intégrale :

$$c_k(t)=\frac{2}{\pi}\int_{-1}^{1}\frac{f(xt)T_k(x)}{\sqrt{1-x^2}}dx. \tag{6.6}$$

Les coefficients de Tchebychev de (6.5) se déduisent alors des fonctions $c_k(t)$ en les évaluant en $t=1$.

En partant de la définition intégrale de la fonction $c_k(t)$ et en supposant que la fonction f est analytique en 0, on peut dans un premier temps développer f en série de Taylor dans la formule (6.6) :

$$c_k(t)=\frac{2}{\pi}\int_{-1}^{1}\frac{\sum_{n\in\mathbb{N}}u_nx^nt^nT_k(x)}{\sqrt{1-x^2}}dx.$$

Les fonctions f et $T_k(x)$ étant analytiques en 0, il existe un disque $\mathcal{D}$ contenant 0 et une borne M tels que pour tout $|x| \leqslant 1$ et $t \in \mathcal{D}$,

$$|f(xt)T_k(x)| = \left|\sum_{n\in\mathbb{N}} u_n x^n t^n T_k(x)\right| \leqslant M.$$

La fonction $\frac{1}{\sqrt{1-x^2}}$ est positive et intégrable sur $[-1,1]$; on peut alors appliquer le théorème de convergence dominée pour obtenir :

$$c_k(t) = \frac{2}{\pi}\sum_{n\in\mathbb{N}} u_n \int_{-1}^{1} \frac{x^n T_k(x)}{\sqrt{1-x^2}} dx\ t^n. \tag{6.7}$$

L'équation (6.7) montre que la fonction $c_k(t)$ est le produit d'Hadamard :

$$c_k(t) = f(t) \odot g_k(t), \quad \text{avec } g_k(t) = \sum_{n=0}^{\infty} \int_{-1}^{1} \frac{x^n T_k(x)}{\sqrt{1-x^2}} dx\ t^n. \tag{6.8}$$

On rappelle que le produit d'Hadamard de deux séries formelles $f = \sum_{n\geqslant 0} f_n x^n$ et $g = \sum_{n\geqslant 0} g_n x^n$ est la série formelle $\sum_{n\geqslant 0} f_n g_n x^n$.

Puissances de x en termes de polynômes de Tchebychev. Les coefficients $g_{n,k}$ de la série $g_k(t)$ définie dans l'équation (6.8) sont les coefficients du développement en série de Tchebychev de la fonction x^n. C'est-à-dire :

$$x^n = \sum_{k\in\mathbb{N}} g_{n,k} T_k(x).$$

Le lemme suivant donne une forme close pour leur série génératrice. Ce résultat est sûrement classique, mais ne trouvant pas de référence, on donne une preuve de ce lemme utilisant la récurrence de Tchebychev.

Lemme 6.1. *Pour tout k, la série g_k est donnée par :*

$$g_k(t) = \frac{2t^k}{(1+\sqrt{1-t^2})^k \sqrt{1-t^2}}. \tag{6.9}$$

Nous allons montrer qu'il découle naturellement des méthodes déjà présentées au chapitre 5 et plus généralement de la D-finitude. On montre d'abord un lemme classique que l'on retrouve par exemple dans [MH03, §2.3.1].

Lemme 6.2. *Les puissances de x se développent en séries de Tchebychev comme :*

$$x^n = 2^{1-n} {\sum_{k=0}^{\lfloor n/2\rfloor}}' \binom{n}{k} T_{n-2k}(x), \tag{6.10}$$

c'est-à-dire

$$g_{n,k} = \begin{cases} 2^{1-n}\binom{n}{\frac{k+n}{2}} & \text{lorsque } n+k \quad \text{est pair,} \\ 0 & \text{sinon.} \end{cases} \tag{6.11}$$

Démonstration. La fonction x^n satisfait l'équation :

$$xy'(x) - ny(x) = 0.$$

Les algorithmes du chapitre 5 donnent alors la récurrence vérifiée par les coefficients de Tchebychev de x^n :

$$(-n+k)u_k + (2+n+k)u_{k+2} = 0. \tag{6.12}$$

La suite $g_{n,k}$ décrite dans ce lemme vérifie cette récurrence. Il suffit donc de vérifier des conditions initiales de $g_{n,k}$ pour prouver le lemme.

De la définition des premiers polynômes de Tchebychev, $T_0(x) = 1$ et $T_1(x) = x$, et de la récurrence vérifiée par cette famille de polynômes,

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),$$

on déduit par récurrence que $T_n(x) = 2^{n-1}x^n + R_n$ pour $n > 0$ où R_n est un polynôme de degré $n-2$. On retrouve donc bien pour tout n les conditions initiales $g_{n,n} = 2^{-n+1}$ et $g_{n,n-1} = 0$. □

Démonstration du lemme 6.1. De cette forme close (6.11), on déduit immédiatement une récurrence en n où k n'apparaît que comme paramètre et qui annule la suite $\left(g_{n,k}\right)_{n\in\mathbb{N}}$:

$$(n+1)(n+2)u_n - (n+2+k)(n+2-k)u_{n+2} = 0. \tag{6.13}$$

Cette récurrence se traduit en une équation différentielle vérifiée par la série génératrice de la suite $(g_{n,k})_{n\in\mathbb{N}}$:

$$\left(2t^2+k^2\right)g_k(t) + \left(4t^3-t\right)g_k'(t) + \left(t^4-t^2\right)g_k''(t) = 0. \tag{6.14}$$

Cette équation est singulière en 0, les conditions initiales $g_{l,k} = 0$ pour $l < k$, $g_{k,k} = 2^{1-k}$ et $g_{k+1,k}$ permettent de conclure que l'on a pour solution unique :

$$g_k(t) = \frac{2t^k}{(1+\sqrt{1-t^2})^k\sqrt{1-t^2}}.$$

□

Une conséquence immédiate de l'équation (6.8) page 116 et du lemme 6.1 est la proposition suivante.

Proposition 6.3. *Si f est une fonction analytique dans un voisinage de 0, alors les coefficients de Tchebychev $c_k(t)$ de la fonction $f(xt)$ vérifient l'égalité suivante :*

$$c_k(t) = f(t) \odot \frac{2t^k}{(1+\sqrt{1-t^2})^k\sqrt{1-t^2}}. \tag{6.15}$$

1.2 Plan

À partir de ce produit d'Hadamard, je montre dans la première section de ce chapitre des formules explicites pour les développements en séries de Tchebychev de fonction hypergéométrique. De nombreuses fonctions spéciales sont des fonctions hypergéométriques, on déduit immédiatement de ces formules les développements (6.1), (6.2) et (6.3).

Il est bien connu [Sta80] que le produit d'Hadamard de deux séries D-finies est une série D-finie. Dans la seconde section, je donne un algorithme permettant de calculer une équation différentielle, dépendante de k, vérifiée par les coefficients de Tchebychev $c_k(t)$. Cet algorithme utilise les propriétés de la fonction g_k pour calculer efficacement ce produit.

Si on suppose que la fonction f est D-finie et analytique dans une ellipse englobant le segment $[-1, 1]$, les équations différentielles vérifiées par les fonctions $c_k(t)$ permettent d'évaluer rapidement et à précision arbitraire ces fonctions en $t = 1$. Une bonne référence sur ce sujet est la thèse de Mezzarobba [Mez11], qui expose des algorithmes rapides permettant d'évaluer numériquement des fonctions D-finies en garantissant l'erreur. Dans la dernière section, ces méthodes d'évaluation sont utilisées pour donner des approximations à précision arbitraire des coefficients de Tchebychev. Un algorithme utilisant cette évaluation numérique et la récurrence de Tchebychev calculée au chapitre 5 est présenté ; celui-ci permet d'obtenir n coefficients de Tchebychev d'une fonction D-finie en $\mathcal{O}(n)$ opérations arithmétiques.

Cet algorithme ne se contente pas de dérouler naïvement la récurrence de Tchebychev. En effet une utilisation naïve de la récurrence n'est pas satisfaisante pour calculer les coefficients de Tchebychev (ceci étant une conséquence immédiate du corollaire 5.35 page 110). Dans cette dernière section l'idée d'utiliser la fonction $f(xt)$ plutôt que $f(x)$ est encore exploitée. On obtient de cette manière une récurrence paramétrée en t vérifiée par la suite $(c_k(t))_{k\in\mathbb{N}}$. De cette récurrence, on déduit un algorithme qui permet de calculer une bonne approximation des n premiers coefficients $c_k(1)$ en temps linéaire par rapport à n.

2 Coefficients de Tchebychev de fonctions hypergéométriques

Dans le cas particulier du développement des fonctions hypergéométriques, le produit d'Hadamard nous permet de déduire des formules donnant des formes closes pour les coefficients de Tchebychev. Ces formules se spécialisent à un très grand nombre de fonctions élémentaires et spéciales puisque celles-ci s'expriment comme des fonctions hypergéométriques (quelques exemples tirés de [AS64] sont donnés dans le tableau 6.1).

La plupart des formules présentes dans cette section ne sont pas nouvelles [Luk69, p. 30-31]. Luke déduit ces formules de théorèmes plus généraux sur des développements de fonction hypergéométrique dans des bases de famille de fonctions hypergéométriques. Ces formules sont toutes prouvées par récurrence. Dans cette section, on montre que le produit d'Hadamard permet de prouver ces formules de manière unifiée et constructive. Ce procédé permet aussi de découvrir de nouvelles formules comme le développement de la fonction d'Airy Ai dans la section 2.4. Une autre contribution de cette section est la simplification des preuves par rapport à celles données dans [Luk69].

Fonction	Symbole	Représentation hypergéométrique
Fonction exponentielle	$\exp(x)$	${}_0F_0\left(\begin{matrix}-\\-\end{matrix}\middle\vert x\right)$
Fonction sinus	$\sin(x)$	${}_0F_1\left(\begin{matrix}-\\\frac{3}{2}\end{matrix}\middle\vert \frac{-x^2}{4}\right)$
Fonction cosinus	$\cos(x)$	${}_0F_1\left(\begin{matrix}-\\\frac{1}{2}\end{matrix}\middle\vert \frac{-x^2}{4}\right)$
Fonction d'erreur	$\operatorname{erf}(x)$	$\frac{2x}{\sqrt{\pi}}{}_1F_1\left(\begin{matrix}\frac{1}{2}\\\frac{3}{2}\end{matrix}\middle\vert -x^2\right)$
Fonction arc sinus	$\arcsin(x)$	$x{}_2F_1\left(\begin{matrix}\frac{1}{2}\frac{1}{2}\\\frac{3}{2}\end{matrix}\middle\vert x^2\right)$
Fonction arc tangente	$\arctan(x)$	$x{}_2F_1\left(\begin{matrix}\frac{1}{2}1\\\frac{3}{2}\end{matrix}\middle\vert -x^2\right)$
Fonction sinus intégral	$\operatorname{Si}(x)$	$x{}_1F_2\left(\begin{matrix}\frac{1}{2}\\\frac{3}{2}\frac{3}{2}\end{matrix}\middle\vert -\frac{x^2}{4}\right)$
Fonction de Bessel de 1ère espèce	$\mathrm{J}_\nu(x)$	$\frac{x^\nu}{\nu!2^\nu}{}_0F_1\left(\begin{matrix}-\\\nu+1\end{matrix}\middle\vert -\frac{x^2}{4}\right)$
Fonction de Bessel modifiée de 1ère espèce	$\mathrm{I}_\nu(x)$	$\frac{x^\nu}{\nu!2^\nu}{}_0F_1\left(\begin{matrix}-\\\nu+1\end{matrix}\middle\vert \frac{x^2}{4}\right)$
Polynôme de Gegenbauer d'indice pair	$C_{2n}^{(\lambda)}(x)$	$(-1)^n\binom{n+\lambda-1}{n}{}_2F_1\left(\begin{matrix}-n;n+\lambda\\\frac{1}{2}\end{matrix}\middle\vert x^2\right)$
Polynôme de Gegenbauer d'indice impair	$C_{2n+1}^{(\lambda)}(x)$	$(-1)^n 2\lambda\binom{n+\lambda}{n}x{}_2F_1\left(\begin{matrix}-n;n+\lambda+1\\\frac{3}{2}\end{matrix}\middle\vert x^2\right)$
Polynôme de Laguerre généralisé	$L_n^{(\alpha)}(x)$	$\binom{n+\alpha}{n}{}_1F_1\left(\begin{matrix}-n\\\alpha+1\end{matrix}\middle\vert x\right)$
Polynôme d'Hermite d'indice pair	$H_{2n}(x)$	$(-1)^n\frac{2n!}{n!}{}_1F_1\left(\begin{matrix}-n\\\frac{1}{2}\end{matrix}\middle\vert x^2\right)$
Polynôme d'Hermite d'indice impair	$H_{2n+1}(x)$	$(-1)^n\frac{(2n+2)!}{(n+1)!}x{}_1F_1\left(\begin{matrix}-n\\\frac{3}{2}\end{matrix}\middle\vert x^2\right)$

FIGURE 6.1 – Représentations hypergéométriques de certaines fonctions spéciales

La plupart des fonctions du tableau 6.1 sont des fonctions hypergéométriques évaluées en un carré, multipliées par un coefficient. Une autre fonction spéciale classique, la fonction d'Airy de première espèce Ai, non indiquée dans ce tableau, s'écrit quant à elle comme une combinaison linéaire de fonctions hypergéométriques évaluées en $\frac{x^3}{9}$:

$$\operatorname{Ai}(x)\,(x) = -\frac{1}{2\pi}x\sqrt[6]{3}\Gamma\left(2/3\right){}_0F_1\left(\begin{matrix}-\\4/3\end{matrix}\middle\vert \frac{x^3}{9}\right) + \frac{1}{3\Gamma(2/3)}\sqrt[3]{3}{}_0F_1\left(\begin{matrix}-\\2/3\end{matrix}\middle\vert \frac{x^3}{9}\right). \tag{6.16}$$

Pour cette raison, cette section ne se contente pas de donner des preuves de formules pour les fonctions hypergéométriques mais aussi pour les fonctions hypergéométriques évaluées en des carrés voire des cubes.

Comme le montre la discussion de la section 2.2 ci-dessous, les fonctions hypergéométriques évaluées en des carrés sont plus faciles. La première partie de cette section leur est dédiée. Dans la seconde partie les formes closes pour des fonctions hypergéométriques classiques sont présentées. La dernière partie montre comment aborder des cas comme (6.16).

La section 2.3 donne une application directe des formules exposées dans les sections 2.1 et 2.2 : elle explicite les développements de Tchebychev des éléments d'autres suites clas-

siques de fonctions spéciales. Si P_n et Q_n sont deux familles de fonctions, les coefficients du développement des P_n sur la base (Q_n) sont appelés constantes de connexions. Gegenbauer [Geg72, Geg84] a donné les premières constantes de connexion dans le cas où les familles P_n et Q_n étaient les familles de polynômes portant son nom, Askey [Ask75] a généralisé les constantes de connexion à d'autres familles de fonctions. Dans cette section, des formes closes de ces coefficients lorsque Q_n est la famille des polynômes de Tchebychev sont données directement par les formules de développement de fonction hypergéométrique en série de Tchebychev. On retrouve des formes closes déjà connues de Gegenbauer [Geg72, Geg84] ou de Askey [Ask75] et on corrige une formule sur le développement de polynôme de Gegenbauer en série de Tchebychev [RZG95].

2.1 Formes closes pour les hypergéométriques en un carré

Théorème 6.4 (Luke [Luk69])**.** *Les coefficients $c_k(t)$ du développement de la série de Tchebychev :*

$$x^l {}_pF_q\left(\left.\begin{matrix}a_1;\dots;a_p\\ b_1;\dots;b_q\end{matrix}\right|(xt)^2\right) = {\sum_{k\in\mathbb{N}}}' c_k(t)T_k(x), \tag{6.17}$$

lorsque $q+1 \geqslant p$, sont donnés par la formule :

$$c_{2k+l}(t) = \frac{2^{1-l}}{4^k}\frac{(a_1)_k\dots(a_p)_k}{(b_1)_k\dots(b_q)_k}\frac{t^{2k+l}}{k!}\,{}_{p+2}F_{q+2}\left(\left.\begin{matrix}a_1+k;\dots;a_p+k;k+l/2+1/2;k+l/2+1\\ b_1+k;\dots;b_q+k;2k+l+1;k+1\end{matrix}\right|t^2\right),$$

et

$$c_{2k+l+1} = 0.$$

Démonstration. Le membre gauche de (6.17) a pour développement de Taylor

$$\sum_{n\in\mathbb{N}} u_{2n+l}t^{2n+l} \quad \text{avec} \quad u_{2n+l} = \frac{(a_1)_n\dots(a_p)_n}{(a_1)_n\dots(a_p)_n n!}.$$

Le coefficient $g_{n+k,k}$ est non nul seulement lorsque n est pair. On a donc :

$$c_{2k+l}(t) = t^{2k+l}\sum_{n\in\mathbb{N}} u_{2n+2k+l}g_{2n+2k+l,2k+l}t^{2n},$$

soit en utilisant les formes closes((6.11) page 116) :

$$c_{2k+l}(t) = t^{2k+l}\sum_{n\in\mathbb{N}}\frac{(a_1)_{n+k}\dots(a_p)_{n+k}}{(b_1)_{n+k}\dots(b_q)_{n+k}(n+k)!}\frac{2^{1-2n-2k-l}(2n+2k+l)!}{(n+2k+l)!n!}t^{2n}.$$

En simplifiant les factorielles notamment à l'aide de la formule de duplication de la fonction Γ [Nat10, http://dlmf.nist.gov/5.5.E5] et de la relation entre le symbole de Pochhammer et les factorielles (2.13), il vient :

$$c_{2k+l}(t) = \frac{2^{1-l}}{4^k}\frac{(a_1)_k\dots(a_p)_k}{(b_1)_k\dots(b_q)_k}\frac{t^{2k+l}}{k!}\sum_{n\in\mathbb{N}}\frac{(a_1+k)_n\dots(a_p+k)_n(k+l/2+1/2)_n(k+l/2+1)_n}{(b_1+k)_{n+k}\dots(b_q+k)_{n+k}(2k+l+1)_n(k+1)_n}\frac{t^{2n}}{n!}.$$

Ce qui est bien le développement en série de Taylor de la fonction hypergéométrique de la proposition. □

Exemple 6.5. La fonction $\cos(x)$ est une fonction hypergéométrique ${}_0F_1$ (voir tableau 6.1) évaluée en $(-1/4)x^2$. En remplaçant t par $-\frac{it}{2}$ et l par zéro dans la formule du théorème, on obtient :

$$c_{2k}(t) = (-1)^k\frac{2}{4^{2k}(1/2)_k}\frac{t^{2k}}{k!}{}_2F_3\left(\begin{matrix}k+1/2;k+1\\k+1/2;2k+1;k+1\end{matrix}\middle|-\frac{t^2}{4}\right).$$

Dans cette fonction hypergéométrique, les paramètres $k+1/2$ et $k+1$ se simplifient et la formule de duplication de la fonction Γ donne $4^k(1/2)_k k! = 2k!$. On en déduit que :

$$c_{2k}(t) = (-1)^k\frac{2}{4^k}\frac{t^{2k}}{(2k)!}{}_0F_1\left(\begin{matrix}-\\2k+1\end{matrix}\middle|-\frac{t^2}{4}\right).$$

Cette fonction est la représentation sous forme hypergéométrique de la fonction de Bessel de première espèce $2(-1)^k J_{2k}(t)$ (voir tableau 6.1). On retrouve alors le développement déjà connu de Snyder [Sny66, page 46] :

$$\cos(xt) = 2{\sum_{k\in\mathbb{N}}}'(-1)^k J_{2k}(t)T_{2k}(x).$$

Exemple 6.6. Le tableau 6.2 donne les séries de Tchebychev de fonction spéciale déduites directement du théorème. Comme pour la fonction cos, le développement de la fonction sin est connu de Snyder [Sny66]. Les développements de arccos et arcsin sont donnés lorsque $t = 1$ dans [MH03, §5.2] ; avec cette spécialisation, on retrouve les coefficients hypergéométriques de l'équation (6.2) page 114.

2.2 Formes closes pour les fonctions hypergéométriques

Le second théorème fondamental de ce chapitre traite du cas général :

Théorème 6.7 (Luke [Luk69])**.** *Les coefficients $c_k(t)$ du développement d'une fonction hypergéométrique généralisée en série de Tchebychev,*

$${}_pF_q\left(\begin{matrix}a_1;\dots;a_p\\b_1;\dots;b_q\end{matrix}\middle|xt\right) = {\sum_{k\in\mathbb{N}}}'c_k(t)T_k(x),$$

avec $q+1\geqslant p$ sont donnés par la formule :

$$c_k(t) = \frac{2}{2^k}\frac{(a_1)_k\dots(a_p)_k}{(b_1)_k\dots(b_q)_k}\frac{t^k}{k!}{}_2pF_{2q+1}\left(\begin{matrix}\frac{a_1+k}{2};\frac{a_1+k+1}{2};\dots;\frac{a_p+k}{2};\frac{a_p+k+1}{2}\\\frac{b_1+k}{2};\frac{b_1+k+1}{2};\dots;\frac{b_q+k}{2};\frac{b_q+k+1}{2};k+1\end{matrix}\middle|\frac{t^2}{4^{q-p+1}}\right).$$

Fonction	série de Tchebychev
$\sin(xt)$	$2\sum_{k\in\mathbb{N}}(-1)^k J_{2k+1}(t)T_{2k+1}(x)$
$\operatorname{erf}(xt)$	$\sum_{k\in\mathbb{N}}\frac{2}{\sqrt{\pi}}\frac{(-1)^k}{4^k}\frac{t^{2k+1}}{(2k+1)k!}{}_1F_1\left(\begin{smallmatrix}k+\frac{1}{2}\\2k+2\end{smallmatrix}\middle\vert -t^2\right)T_{2k+1}(x)$
$\arccos(xt)$	$\frac{\pi}{2}T_0(x)-\sum_{k\in\mathbb{N}}\frac{t^{2n+1}(2n)!}{4^{2n}(1+2n)n!^2}{}_2F_1\left(\begin{smallmatrix}k+\frac{1}{2},k+\frac{1}{2}\\2k+2\end{smallmatrix}\middle\vert t^2\right)T_{2k+1}(x)$
$\arcsin(xt)$	$\sum_{k\in\mathbb{N}}\frac{t^{2n+1}(2n)!}{4^{2n}(1+2n)n!^2}{}_2F_1\left(\begin{smallmatrix}k+\frac{1}{2},k+\frac{1}{2}\\2k+2\end{smallmatrix}\middle\vert t^2\right)T_{2k+1}(x)$
$\arctan(xt)$	$\sum_{k\in\mathbb{N}}2\frac{(-1)^k t^{2k+1}\left(1+\sqrt{t^2+1}\right)^{-1-2k}}{1+2k}T_{2k+1}(x)$
$\operatorname{Si}(xt)$	$\sum_{k\in\mathbb{N}}\frac{4^{-k}(-1)^k}{(2k+1)!(2k+1)}{}_1F_1\left(\begin{smallmatrix}k+\frac{1}{2}\\2k+2,k+\frac{3}{2}\end{smallmatrix}\middle\vert -\frac{1}{4}t^2\right)T_{2k+1}(x)$

FIGURE 6.2 – Formes closes pour les séries de Tchebychev de certaines fonctions spéciales

Démonstration. On repart du développement en série de (6.8) page 116

$$c_k(t) = \sum_{n\in\mathbb{N}} u_n g_{n,k} x^k,$$

où u_n sont les coefficients de Taylor de ${}_pF_q\left(\begin{smallmatrix}a_1;\dots;a_p\\b_1;\dots;b_q\end{smallmatrix}\middle\vert xt\right)$.

Comme $g_{n,k}$ est nul lorsque $n<k$, $c_k(t)$ est de valuation k, on a donc :

$$c_k(t) = t^k \sum_{n\in\mathbb{N}} u_{n+k} g_{n+k,k} t^n.$$

En utilisant les formes closes vérifiées par u_k et g_k et le fait que le coefficient $g_{n+k,k}=0$ lorsque n est impair, on a :

$$c_k(t) = t^k \sum_{n\in\mathbb{N}} 2^{1-2n-k}\frac{(a_1)_{2n+k}\dots(a_p)_{2n+k}}{(b_1)_{2n+k}\dots(b_q)_{2n+k}(2n+k)!}\frac{(2n+k)!}{(n+k)!n!}t^{2n}.$$

En simplifiant les factorielles à l'aide de la formule de duplication de la fonction Γ,

$$c_k(t) := \frac{2}{2^k}\frac{(a_1)_k\dots(a_p)_k}{(b_1)_k\dots(b_q)_k}\frac{t^k}{k!}\sum_{n\in\mathbb{N}}4^{n(p-q-1)}\frac{(\frac{a_1+k}{2})_n(\frac{a_1+k+1}{2})_n\dots(\frac{a_p+k}{2})_n(\frac{a_p+k+1}{2})_n}{(\frac{b_1+k}{2})_n(\frac{b_1+k+1}{2})_n\dots(\frac{b_q+k}{2})_n(\frac{b_q+k+1}{2})_n(k+1)_n}\frac{t^{2n}}{n!},$$

ce qui conclut la preuve au vu de (2.12) page 21. □

Exemple 6.8. Le développement de la fonction exp comme fonction hypergéométrique (voir tab 6.1) permet de déduire l'égalité :

$$c_k(t) = \frac{2}{2^k}\frac{t^k}{k!}{}_0F_1\left(\begin{smallmatrix}-\\k+1\end{smallmatrix}\middle\vert \frac{t^2}{4}\right),$$

où $c_k(t)$ sont des coefficients de Tchebychev de $\exp(xt)$. On retrouve la représentation hypergéométrique de la fonction $2\,\mathrm{I}_k(t)$ et le développement classique [Sny66] :

$$\exp(xt) = {\sum_{k\in\mathbb{N}}}' 2\,\mathrm{I}_k(t)T_k(x).$$

2.3 Cas particulier : les constantes de connexion

Une application de ces premières sections est le calcul de certaines formes closes de constantes de connexion. Ces constantes $a_{n,k}$ [Ask75, cours 7] relient deux familles de fonctions $(P_n)_{n\in\mathbb{N}}$ et $(Q_k)_{k\in\mathbb{N}}$ entre elles de la façon suivante :

$$P_n = \sum_{k\in\mathbb{N}} a_{n,k}Q_k.$$

On s'intéresse ici au cas particulier où les polynômes Q_k sont des polynômes de Tchebychev. Le chapitre 8 traite des cas plus généraux. Gegenbauer [Geg84] a donné le développement des polynômes portant son nom :

$$C_n^\lambda(x) = \sum_{k=0}^{n} \frac{(\lambda)_{n-k}\,(\lambda)_k}{(n-k)!k!}T_{|n-2k|}(x). \tag{6.18}$$

Dans le même article il donne une démonstration assez technique. On va montrer comment on peut obtenir ce résultat simplement grâce au théorème 6.4.

Area, Dimitrov, Godoy et Ronveaux [ADGR04] donnent une formule plus générale pour les coefficients de Tchebychev $A_{n,k}(t)$ du polynôme $C_n^\lambda(xt)$:

$$A_{n,k}(t) = \frac{(-1)^{(n-k)/2}}{k!((n-k)/2)!}\frac{t^k}{2^{3n-k}}\frac{(2\lambda)_n(n+2\lambda)_n n!}{(\lambda+1/2)_n(\lambda+(n-k)/2)_{(n-k)/2}(\lambda)_n} \times {}_2F_1\left(\begin{matrix}(k-n)/2;(2\lambda+k+n)/2\\ k+1\end{matrix}\middle| t^2\right),$$

lorsque $n+k$ est pair et 0 sinon. Malheureusement cette formule est fausse (exemple : le polynôme de Gegenbauer évalué en $n=6$, $\lambda=2$ et $t=x=1$ donne 84 contre $1,63\ldots$ en utilisant cette formule.

Une application du théorème 6.4 fournit directement la formule correcte :

$$A_{2n,2k}(t) = (-1)^n\binom{n+\lambda-1}{n}\frac{2}{4^k}\frac{(-n)_k(n+\lambda)_k}{(1/2)_k}\frac{t^{2k}}{k!}{}_4F_3\left(\begin{matrix}-n+k;n+\lambda+k;k+1;k+1/2\\ 1/2+k;2k+1;k+1\end{matrix}\middle| t^2\right),$$

$$A_{2n+1,2k+1}(t) = (-1)^n\binom{n+\lambda}{n}\frac{2\lambda}{4^k}\frac{(-n)_k(n+\lambda+1)_k}{(3/2)_k}\frac{t^{2k+1}}{k!}{}_4F_3\left(\begin{matrix}k-n;n+\lambda+k+1;k+1;k+3/2\\ 3/2+k;2k+2;k+1\end{matrix}\middle| t^2\right),$$

et

$$A_{2n+1,2k}(t) = A_{2n,2k+1}(t) = 0.$$

Ces expressions se simplifient du fait de la présence simultanée de $k+1$ et $k+1/2$ (ou $k+3/2$) dans les deux ensembles de paramètres. De cette simplification vient la proposition suivante.

Proposition 6.9. *Le développement des polynômes de Gegenbauer en série de Tchebychev s'écrit :*

$$C_{2n}^{(\lambda)}(xt) = 2\sum_{k=0}^{n}{}'(-1)^{n+k}\frac{(\lambda)_{n+k}}{(n-k)!}\frac{t^{2k}}{2k!}{}_2F_1\left(\begin{matrix}-n+k; n+\lambda+k;\\ 2k+1\end{matrix}\middle| t^2\right)T_{2k}(x),$$

et

$$C_{2n+1}^{(\lambda)}(xt) = 2\sum_{k=0}^{n}(-1)^{n+k}\frac{(\lambda)_{n+k+1}}{(n-k)!}\frac{t^{2k+1}}{(2k+1)!}{}_2F_1\left(\begin{matrix}-n+k; n+\lambda+k+1\\ 2k+2\end{matrix}\middle| t^2\right)T_{2k+1}(x).$$

On retrouve la formule (6.18) en évaluant en 1 grâce à l'identité de Chu-Vandermonde [Nat10, http://dlmf.nist.gov/15.4.E24] :

$${}_2F_1\left(\begin{matrix}-n; b\\ c\end{matrix}\middle| 1\right) = \frac{(c-b)_n}{(c)_n}$$

Les mêmes idées fournissent des formes closes pour les développements des polynômes de Laguerre généralisées (L_n^{α}) ou des polynômes d'Hermite (H_n) (voir tableau 6.1 page 119).

Proposition 6.10. *Les polynômes orthogonaux classiques suivants se développent en séries de Tchebychev comme :*

$$L_n^{(\alpha)}(xt) = \sum_{k=0}^{n}{}'\binom{n+\alpha}{n}\frac{2}{2^k}\frac{(-n)_k}{(\alpha+1)_k}\frac{t^k}{k!}{}_2F_3\left(\begin{matrix}(-n+k)/2; (-n+k+1)/2\\ (\alpha+k+1)/2; (\alpha+k)/2+1; (k+1)\end{matrix}\middle| \frac{t^2}{4}\right)T_k(x),$$

$$H_{2n}(xt) = \sum_{k=0}^{n}{}'(-1)^n 4^{-k+1/2}\frac{2n!}{n!}\frac{(-n)_k}{(1/2)_k}\frac{t^{2k}}{k!}{}_1F_1\left(\begin{matrix}-n+k\\ 2k+1\end{matrix}\middle| t^2\right)T_{2k}(x),$$

$$H_{2n+1}(xt) = \sum_{k=0}^{n}(-1)^n 4^{-k}\frac{(2n+2)!}{(n+1)!}\frac{(-n)_k}{(3/2)_k}\frac{t^{2k+1}}{k!}{}_1F_1\left(\begin{matrix}-n+k\\ 2k+2\end{matrix}\middle| t^2\right)T_{2k+1}(x).$$

2.4 Généralisation à d'autres fonctions hypergéométriques

Si la récurrence de Taylor associée à une fonction est à deux termes mais d'ordre plus grand que deux, les théorèmes précédents ne donnent pas de forme close pour les coefficients. Cependant l'idée se généralise ; lors de cette thèse, du code Maple a été développé pour calculer des formes closes pour ces coefficients de Tchebychev. Ce code est utilisé par le DDMF [BCD+10]. Un exemple intéressant est donné par le développement de la fonction d'Airy Ai. La récurrence satisfaite par les coefficients de Taylor de cette fonction est à deux termes et d'ordre 3 :

$$u_n - \left(n^2+5n+6\right)u_{n+3} = 0.$$

Grâce à la forme particulière de cette récurrence et aux méthodes utilisées pour la preuve des théorèmes 6.4 et 6.7, on obtient le développement suivant.

$$
\begin{aligned}
&\mathrm{Ai}(x) = \\
&\quad {\sum_{k\in\mathbb{N}}}' \frac{1}{81^k k!}\left(-\frac{3^{2/3}}{144}\frac{1}{\Gamma\left(k+4/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2\,k+4/3;1/2\,k+5/6\\ 4/3;5/3;k+1;k+4/3;k+5/3\end{matrix}\right|\frac{1}{1296}\right)\right.\\
&\qquad \left. +\frac{\sqrt[3]{3}}{3}\frac{2}{\Gamma\left(k+2/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2k+2/3;1/2\,k+1/6\\ 1/3;2/3;k+1;k+2/3;k+1/3\end{matrix}\right|\frac{1}{1296}\right)\right) T_{3k}(x)\\
&\quad +\sum_{k\in\mathbb{N}} \frac{1}{81^k k!}\left(\frac{\sqrt[3]{3}}{36}\frac{1}{\Gamma\left(k+5/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2k+2/3;1/2k+7/6\\ 2/3;4/3;k+1;k+4/3;k+5/3\end{matrix}\right|\frac{1}{1296}\right)\right.\\
&\qquad \left. -\frac{3^{2/3}}{9}\frac{1}{\Gamma\left(k+4/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2k+5/6;1/2k+1/3\\ 1/3;2/3;k+1;k+4/3;k+2/3\end{matrix}\right|\frac{1}{1296}\right)\right) T_{3k+1}(x)\\
&\quad +\sum_{k\in\mathbb{N}} \frac{1}{81^{k+1}(k+1)!}\left(\frac{\sqrt[3]{3}}{24}\frac{1}{\Gamma\left(k+5/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2k+7/6;1/2k+5/3\\ 4/3;5/3;k+2;k+5/3;k+7/3\end{matrix}\right|\frac{1}{1296}\right)\right.\\
&\qquad \left. -\frac{3^{2/3}}{9}\frac{3k+4}{\Gamma\left(k+7/3\right)}{}_2F_5\left(\left.\begin{matrix}1/2k+4/3;1/2k+5/6\\ 2/3;4/3;k+2;k+4/3;k+5/3\end{matrix}\right|\frac{1}{1296}\right)\right) T_{3k+2}(x).
\end{aligned}
$$

3 Calcul du produit d'Hadamard dans le cas D-fini

La théorie de l'holonomie [Zei90] affirme que les fonctions $c_k(t)$ définies par (6.6) sont D-finies si la fonction f est D-finie. Connaissant une équation différentielle vérifiée par f, l'algorithme de création télescopique [Zei90, AZ90, CS98, Chy00] calcule l'équation différentielle vérifiée par chaque $c_k(t)$. Les packages Maple Mgfun [Chy98, Pec09] et Mathematica HolonomicFunctions [Kou10a, Kou10b] permettent d'effectuer ce calcul.

Notre approche dans ce chapitre est plus directe et n'utilise pas la création télescopique. Les propriétés de clôture des fonctions D-finies [Sta80] entraînent que les fonctions c_k (6.8) page 116 sont D-finies comme produits d'Hadamard de fonctions D-finies. On utilise donc le produit d'Hadamard pour calculer l'équation différentielle, paramétrée par k, vérifiée par $c_k(t)$.

3.1 Calcul d'une équation différentielle vérifiée par $c_k(t)$

L'équation différentielle vérifiée par le produit d'Hadamard de deux fonctions D-finies se calcule en effectuant une élimination dans un système linéaire à coefficients polynomiaux à partir des équations différentielles vérifiées par les fonctions. La dimension de ce système linéaire est telle que la complexité de ce calcul peut être élevée. Cependant, le produit d'Hadamard ici est particulier, et il est possible d'obtenir directement un algorithme plus efficace pour ce calcul. Les coefficients de Taylor $g_{n,k}$ de $g_k(x)$ vérifiant une récurrence à

deux termes (6.13),

$$(n+2)(n+1)g_{n,k} - (n+k+2)(n-k+2)g_{n+2,k} = 0,$$

le calcul du produit d'Hadamard est aisé.

Exemple 6.11. Les coefficients de la série de Taylor de $\exp(x)$ vérifient l'équation d'ordre 2 :

$$u_n - (n+1)(n+2)u_{n+2} = 0,$$

La récurrence vérifiée par $\frac{g_{n,k}}{n!}$ s'obtient simplement en multipliant terme à terme les coefficients de cette récurrence et de la récurrence satisfaite par $g_{n,k}$:

$$(n+2)(n+1)c_{n,k} - (n+2)(n+1)(n+k+2)(n-k+2)c_{n+2,k} = 0.$$

Les indices n étant des entiers naturels, on peut simplifier cette équation par le facteur commun $(n+1)(n+2)$, ce qui donne la récurrence vérifiée par les coefficients de Taylor de la fonction de Bessel modifiée $\mathrm{I}_k(t)$. Pour montrer la validité du développement de la fonction exponentielle,

$$\exp(xt) = {\sum_{n=0}^{\infty}}' 2\,\mathrm{I}_k(t)T_n(x),$$

il ne reste plus qu'à vérifier des conditions initiales. Celles-ci seront données dans la prochaine section.

Dans cet exemple, le produit d'Hadamard se calcule simplement une fois que l'on a la récurrence d'ordre 2 qui annule les coefficients de la fonction exp. De façon générale, le calcul de la récurrence vérifiée par le produit d'Hadamard entre une suite u_n et la suite $c_{n,k}$ est plus facile lorsque la récurrence vérifiée par u_n va de deux en deux ; c'est-à-dire qu'elle annule en même temps les suites $(u_0, 0, u_2, 0, u_4, \ldots)$ et $(0, u_1, 0, u_3, \ldots)$. En effet, partant de

$$r_0(n)u_n + r_1(n)u_{n+2} + \cdots + r_d(n)u_{n+2d} = 0, \tag{6.19}$$

et des coefficients de la récurrence (6.12),

$$p_0(n) = (n+2)(n+1) \quad \text{et} \quad p_2(n) = -(n+k+2)(n-k+2),$$

la suite $(g_{n,k}u_n)_{n\in\mathbb{N}}$ satisfait la récurrence :

$$\begin{aligned}\prod_{i=0}^{d-1} p_0(n+2i)r_0(n)g_{n,k}u_n + \cdots \\ \prod_{i=0}^{d-s-1} p_0(n+2i) \prod_{i=d-s}^{d-1} p_2(n+2i)r_s(n)g_{n+2s,k}u_{n+2s} + \cdots \\ \prod_{i=0}^{d-1} p_2(n+2i)r_d(n)g_{n+2d,k}u_{n+2d} = 0.\end{aligned} \tag{6.20}$$

Entrée: une récurrence rec telle que $rec(u_n) = \sum_{n=0}^{d} a_i(n)u_{n+i} = 0$ avec $f(x) = \sum_{n\in\mathbb{N}} u_n x^n$

Sortie: une récurrence rec_2 telle que $rec_2(c_{n,k}) = 0$ avec $c_k(t) = \sum_{n\in\mathbb{N}} c_{n,k}t^n$ et $f(xt) = \sum'_{k\in\mathbb{N}} c_k(t)T_k(x)$

1: Calculer

$$\widetilde{rec} = \sum_{n=0}^{d}(-1)^i a_i(n)u_{n+i} = 0$$

2: Calculer

$$rec_1 = \mathrm{lclm}(rec, \widetilde{rec}) = r_0(n)u_n + r_1(n)u_{n+2} + \cdots + r_d(n)u_{n+2d}$$

3: **renvoyer**

$$rec_2 = \sum_{s=0}^{d}\prod_{i=0}^{d-s-1} p_0(n+2i)\prod_{i=d-s}^{d-1} p_2(n+2i)r_s(n)c_{n+2s,k}u_{n+2s}$$

Algorithme 6.1: Produit d'Hadamard pour les coefficients de Tchebychev

Dans le cas général, il suffit de calculer, à partir de l'équation de récurrence vérifiée par la suite $(u_n)_{n\in\mathbb{N}}$, une récurrence vérifiée par les suites $(u_0, 0, u_2, 0, \ldots)$ et $(0, u_1, 0, u_3, \ldots)$, c'est-à-dire $\frac{1}{2}(u_n + (-1)^n u_n)$ et $\frac{1}{2}(u_n - (-1)^n u_n)$. Si l'équation

$$\sum_{n=0}^{d} a_i(n)u_{n+i} = 0 \tag{6.21}$$

est vérifiée par la suite (u_n), alors l'équation

$$\sum_{n=0}^{d}(-1)^i a_i(n)u_{n+i} = 0, \tag{6.22}$$

est vérifiée par la suite $((-1)^n u_n)$. Le lclm (4.16 page 62) de ces deux récurrences annule les sommes de ces suites. En calculant ce lclm, on obtient alors la récurrence souhaitée. On retrouve ici une des méthodes de [BCLR03].

L'algorithme 6.1 résume l'ensemble des opérations calculant une récurrence annulant la suite $(u_n g_{n,k})_{n\in\mathbb{N}}$ à partir d'une récurrence vérifiée par la suite $(u_n)_{n\in\mathbb{N}}$

Le passage de l'équation différentielle vérifiée par f à une équation différentielle vérifiée par c_k se déduit de cet algorithme et des algorithmes permettant de passer d'une relation de récurrence à l'équation différentielle vérifiée par les séries génératrices des solutions de cette récurrence.

3.2 Conditions initiales

Pour que la fonction c_k soit entièrement déterminée par l'équation différentielle, il ne reste plus qu'à déterminer les conditions initiales de l'équation différentielle ou de façon équivalente celles de la récurrence vérifiée par les coefficients de Taylor de la fonction c_k.

Si la récurrence est d'ordre $2d$ comme (6.20), il faut au moins $2d$ conditions initiales pour caractériser la solution. Il peut en falloir plus. Le nombre de conditions initiales nécessaires dépend aussi des racines du polynôme de tête de la récurrence, c'est-à-dire ici des racines de :

$$\prod_{i=0}^{d-1} p_2(n+2i) r_d(n) g_{n+2d,k}.$$

Le facteur $\prod_{i=0}^{d-1} p_2(n+2i)$ admet d racines : $k-2, k-4, \ldots, k-2d$. Comme la série g_k est de valuation k, la série c_k est de valuation au moins k. Les coefficients $c_{n,k}$ de c_k sont donc nuls pour $n < k$. Si on nomme n_0 la plus grande racine du polynôme r_d, on peut alors caractériser les conditions initiales de la récurrence par :

$$c_{i,k} = \begin{cases} 2^{1-i}\binom{i}{\frac{i+k}{2}} u_i & \text{si } i+k \quad \text{est pair et } i \text{ plus grand que } \max(n_0,k), \\ 0 & \text{sinon} \end{cases}, \tag{6.23}$$

pour i allant de 0 à $2d + \max(n_0, k) - 1$.

Les conditions initiales de l'équation différentielle vérifiée par c_k se déduisent immédiatement de cette équation.

4 Approximation des coefficients de Tchebychev

Pour calculer le $i^{\text{ème}}$ coefficient de Tchebychev d'une fonction que l'on sait évaluer numériquement, on peut utiliser l'intégrale issue du produit scalaire des polynômes de Tchebychev

$$\frac{2}{\pi} \int_{-1}^{1} \frac{f(x) T_k(x)}{\sqrt{1-x^2}} dx.$$

Cette méthode n'est pas efficace. Par exemple Maple ne parvient à calculer numériquement que les neuf premiers coefficients de Tchebychev de la fonction d'Airy Ai de cette façon.

Les formes closes données dans la section précédente permettent d'évaluer efficacement les coefficients de Tchebychev en utilisant par exemple des algorithmes numériques dédiés à l'évaluation des fonctions hypergéométriques. Les formules de la section 2 ne donnent des formes closes que pour une certaine classe de fonctions D-finies ; l'utilisation des formes closes pour l'évaluation numérique est donc limitée.

Une autre approche pour calculer numériquement les coefficients de Tchebychev $c_k(1)$, où les fonctions $c_k(t)$ sont définies par (6.6) page 115, est donnée par Thacher [Tha64] et utilise le produit d'Hadamard (6.8) page 116. Sa méthode consiste à calculer les coefficients u_n

du développement de la fonction en série de Taylor en 0, puis pour tout k la somme qui approxime $c_k(1)$

$$\sum_{n=0}^{N} g_{n,k} u_n,$$

où $g_{n,k}$ est défini par (6.11) et N est un entier choisi par l'utilisateur. Dans cette section, on se propose d'affiner cette méthode lorsque la fonction est D-finie. On pourra déterminer explicitement l'entier N en fonction de la précision souhaitée et calculer rapidement les coefficients.

Cette section se décompose en deux parties. La première est consacrée à l'évaluation rapide d'*un* coefficient à précision arbitraire en utilisant des algorithmes dédiés à l'évaluation numérique de fonction D-finie. Dans la seconde partie se trouve la contribution de cette section. On propose un algorithme calculant d coefficients de Tchebychev à précision ϵ en $\mathcal{O}(d + \log(\epsilon^{-1}))$ opérations arithmétiques. Cet algorithme se base sur la récurrence vérifiée par les coefficients de Tchebychev $(c_k(t))_{k\in\mathbb{N}}$ (§ 5).

Nous supposons dans cette section que les opérations arithmétiques entre les nombres réels ou complexes sont effectuées en arithmétique exacte (c'est-à-dire en général rationnelle) dans un sous-corps $\mathbb{K}$ de $\mathbb{C}$, que l'on considère effectif. Les calculs effectués ici ne sont pas stables numériquement, les algorithmes n'ont donc aucun intérêt si on remplace l'arithmétique exacte par de l'arithmétique flottante.

Comme dans le reste de cette thèse, la complexité utilisée ici est la complexité arithmétique. En d'autre termes, nous n'attribuons un coût unitaire qu'aux opérations dans $\mathbb{K}$.

4.1 Calcul rapide d'un coefficient de Tchebychev

Mezzarobba a développé le package Maple NumGfun [2] [Mez10] disponible comme sous-package de gfun [SZ94, http://algo.inria.fr/libraries/papers/gfun.html] qui permet d'évaluer numériquement une fonction en un point à partir de l'équation différentielle vérifiée par celle-ci et des conditions initiales. L'évaluation d'une fonction par ce package est garantie.

L'exemple suivant montre comment, en utilisant la théorie de la première section et le package gfun, calculer un coefficient de Tchebychev d'une fonction D-finie donnée.

Exemple 6.12. La fonction $\mathrm{erf}(x)$ satisfait l'équation différentielle :

```
> deq1 := 2*x*diff(y(x), x)+diff(y(x), x, x):
```

On affecte à la variable *deq*2 l'équation différentielle (6.14) page 117 vérifiée par $g_k(x)$:

2. La version publique actuelle de NumGfun ne traite pas des équations différentielles singulières à l'origine. Comme la série $c_k(t)$ est de valuation k ((6.23) page 128), l'équation différentielle l'annulant admet une singularité apparente en 0. Pour cette raison, dans cette section, la version publique de NumGfun n'est pas utilisée et remplacée par une version en développement qui traite les équations admettant une singularité apparente en 0.

```
> deq2 := (2*x^2+k^2)*y(x)+(4*x^3-x)*diff(y(x), x)+(x^4-x^2)*diff(y(x), x,
x):
```

On effectue le produit d'Hadamard entre $deq1$ et $deq2$:

```
> deq3 := eval(hadamardproduct(deq1, deq2, y(x)), [_C=0]):
```

Sachant que la fonction c_k est au moins de valuation k, pour plus de commodité pour la suite, on affecte à la variable $deq3$, l'équation différentielle vérifiée par $\frac{c_k(x)}{x^k}$ (ce qui ne change pas le résultat de l'évaluation en 1).

```
> deq3 := collect(expand(eval(deq3, y= proc(x) x^k*y(x) end proc)), [y, diff(y(x),
x), diff(y(x), x, x), diff(y(x), x, x, x)]):
```

En utilisant la procédure analytic_continuation du package NumGfun, on évalue $c_1(1)$ avec 200 chiffres garantis. Cette procédure utilise des conditions initiales symboliques, on doit donc effectuer ce calcul en deux étapes. La première étape utilise la procédure avec l'aide des conditions initiales symboliques et la seconde étape évalue le résultat obtenu avec les conditions initiales de l'équation différentielle (6.23) page 128.

```
> analytic_continuation(eval(deq3, k=1), y(x), [0,1], 200):
> evalf[200](eval(%, {_C[0]=0, _C[1]=2/sqrt(Pi)}));
```

$$0.90434633683079727479088401848515229290965150861477458941 9\ldots$$

On peut aussi calculer 10000 chiffres du $51^{\text{ème}}$ coefficient en 14 secondes :

```
> st := time():
> analytic_continuation(eval(deq3, k=51), y(x), [0,1], 10000):
> evalf[10000](eval(%, {_C[1]=eval(diff(erf(x),x$51),x=0)*2^(-50)/51!, _C[0]=0}));
> time()-st;
```

$$5.22625984410981191673592537174705582306120 6\ldots \times 10^{-61}$$

$$14.336$$

Cet exemple montre comment obtenir rapidement l'évaluation numérique d'*un* coefficient de Tchebychev en Maple à partir de l'équation différentielle vérifiée par la série.

Souvent, on a besoin de calculer plusieurs coefficients de Tchebychev. Dans ce cas, on utilise la récurrence de Tchebychev pour accélérer les calculs. Dans la prochaine section, on verra comment calculer et utiliser une récurrence vérifiée par la suite indexée par k :

$$\sum_{n=0}^{N} c_{n,k}, \tag{6.24}$$

$c_{n,k}$ étant le $n^{\text{ème}}$ coefficient de Taylor de c_k.

L'intérêt de cette suite est que, sous certaines conditions d'analycité, les coefficients tendent vers $c_k(1)$ quand N tend vers l'infini. Pour utiliser la récurrence vérifée par cette suite, on a besoin de ses conditions initiales. On doit donc évaluer cette somme pour les premiers k, ce que l'algorithme 6.2 effectue.

En partant du principe que la fonction c_k est analytique dans le disque unité, on sait alors que la somme (6.24) converge vers $c_k(1)$ quand N tend vers l'infini. Un algorithme de Mezzarobba et Salvy [MS10] permet de quantifier cette convergence puisque celui-ci

Entrée: *deq* une équation différentielle n'admettant pas de singularité dans le disque unité et munie des conditions initiales à valeurs dans $\mathbb{K}$ donnant une solution unique, un entier k et une erreur cible ϵ
Sortie: Une approximation dans $\mathbb{K}$ de $\frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx$ avec une erreur $\leqslant \epsilon$
1: Calculer rec_k, la récurrence (6.20) page 126 à partir de l'algorithme 6.1
2: Calculer les conditions initiales de la récurrence à partir de (6.23)
3: Calculer N telle que la suite $c_{n,k}$, annulée par rec et vérifiant les conditions initiales calculées précédemment, vérifie $|\sum_{n=N+1}^{\infty} c_{n,k}| < \epsilon$ (Utiliser l'algorithme de [MS10])
4: **renvoyer** $\sum_{n=0}^{N} c_{n,k}$ calculée en déroulant rec

Algorithme 6.2: Calcul du $k^{\text{ème}}$ coefficient de Tchebychev

borne, à l'aide d'une forme close, la suite

$$\sum_{n=N+1}^{\infty} |u_n|,$$

en fonction de N à partir de la récurrence vérifiée par la suite $(c_{n,k})_{k\in\mathbb{N}}$. Cet algorithme permet donc d'obtenir une borne sur la différence entre $c_k(1)$ et la somme (6.24).

La récurrence vérifiée par cette suite permet aussi de calculer rapidement la somme (6.24). De ces résultats on déduit un algorithme pour calculer une approximation de $c_k(1)$ en garantissant l'erreur.

Proposition 6.13. *Pour toute fonction f solution d'équation différentielle linéaire à coefficients dans $\mathbb{K}[x]$, n'admettant pas de singularité dans le disque unité, munie de conditions initiales dans $\mathbb{K}$, tout entier k et tout $\epsilon > 0$, l'algorithme 6.2 calcule en $\mathcal{O}(k + \log(\epsilon^{-1}))$ opérations arithmétiques la somme $\sum_{n=0}^{N} c_{n,k}$ où $c_{n,k}$ est défini par :*

$$c_k(t) = \sum_{n\in\mathbb{N}} c_{n,k}t^n = \frac{2}{\pi}\int_{-1}^{1}\frac{f(xt)T_k(x)}{\sqrt{1-x^2}}dx,$$

et N est tel que $|c_k(1) - \sum_{n=0}^{N} c_{n,k}| < \epsilon$.

Démonstration. Hadamard [Had99] montre que le produit d'Hadamard de deux fonctions f et g_k n'a d'autres points singuliers que ceux que l'on obtient en multipliant les affixes des différents points singuliers de f par celles des différents points singuliers de c_k. L'équation différentielle annulant g_k a le point 1 comme singularité la plus proche de 0. On en déduit que l'équation différentielle annulant c_k n'admet pas de singularité dans le disque unité. On a donc l'égalité :

$$c_k(1) = \sum_{n\in\mathbb{N}} c_{k,n} = \frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx.$$

La complexité du calcul de la récurrence à partir de l'équation différentielle avec l'algorithme 6.1 page 127 dépend seulement de la forme de cette équation différentielle et

est donc indépendante de k et ϵ. En revanche le calcul des conditions initiales dépend de k, puisqu'il faut calculer les coefficients u_i du développement de Taylor pour i allant de 0 à k. En utilisant la récurrence vérifiée par la suite u_i (venant de la D-finitude de f) des coefficients de Taylor de f, ce calcul s'effectue en $\mathcal{O}(k)$ opérations arithmétiques. Mezzarobba [Mez11] montre que le nombre N de termes suffisant pour obtenir une précision ϵ de c_k par $\sum_{n=0}^{N} c_{k,n}$ est $\frac{\log_2(\epsilon^{-1})}{\log_2(\rho)} + o\left(\log_2\left(\epsilon^{-1}\right)\right)$, où ρ la plus petite singularité de l'équation différentielle vérifiée par c_k. Cette singularité dépend de la singularité dominante de l'équation différentielle vérifiée par f que l'on suppose ici constante. On a donc $N = \mathcal{O}(\log(\epsilon^{-1}))$.

Du fait de la valuation supérieure à k de la fonction c_k, on a pour tout N l'égalité

$$\sum_{n=0}^{N} c_{n,k} = \sum_{n=0}^{N-k} c_{n+k,k}.$$

La récurrence vérifiée par la suite $(c_{n+k,k})_{n\in\mathbb{N}}$ s'obtient en effectuant, pour tous les coefficients polynomiaux de la récurrence vérifiée par $(c_{n,k})_{n\in\mathbb{N}}$, le changement de variable $n = n + k$. De cette récurrence et des conditions initiales, on peut déduire avec l'algorithme de Mezzarobba et Salvy exprime une forme close $p(N)$ telle que :

$$\left|\sum_{n=N-k+1}^{\infty} c_{n+k,k}\right| < p(N).$$

Le calcul de la forme close p dépend uniquement de l'équation différentielle et des conditions initiales, donc s'effectue en $\mathcal{O}(1)$ opérations arithmétiques. Comme p dépend uniquement de l'équation différentielle, et est donc indépendant de ϵ et N, calculer un N tel que $p(N) < \epsilon$ s'effectue en $\mathcal{O}(\log(\epsilon^{-1}))$ opérations.

La récurrence et les conditions initiales définissent une unique solution. En la déroulant, on calcule l'approximation souhaitée. Calculer N coefficients à partir d'une récurrence linéaire et les additionner entre eux s'effectue en un nombre d'opérations linéaire en N.

La complexité de cet algorithme ne dépend donc que du calcul de N, des k coefficients u_i et des N coefficients $c_{n,k}$. □

L'exemple suivant illustre le calcul des coefficients de Tchebychev de la fonction d'Airy $\mathrm{Ai}(x)$ à l'aide de l'algorithme ci-dessus.

Exemple 6.14. Le calcul du 9ème coefficient de Tchebychev de la fonction $\mathrm{Ai}(x)$. En utilisant l'algorithme 6.1 page 127, on calcule la récurrence *rec* vérifiée par la suite $(c_{n,k})_{k\in\mathbb{N}}$.

```
> rec;
```

$$rec := -(n+2)(n+1)(n+4)(n+3)(n+6)(n+5)u(n) +$$
$$(n+k+2)(n-k+2)(n+4+k)(n+4-k)(n+6+k)(n+6-k)\left(91n^2+216n+180+n^4+16n^3\right)u(n+6)$$

En utilisant la procédure bound_rec_tail du package NumGfun, on majore le reste de la suite par une forme close. Pour calculer les conditions initiales, on utilise ici encore la formule (6.23).

```
> ic :=seq(op([u(i)=eval(diff(AiryAi(x)*2^(-i+1)/i!*binomial(i, (i+9)/2),
x$i), x=0), u(i+1)=0]), i=9..14, 2):
> rec := {eval(rec, k=9), seq(u(i)=0, i=0..8), ic}:
> sol := bound_rec_tail(rec, u(n)):
```

Il suffit d'évaluer cette borne pour trouver un N tel que le reste est plus petit que 10^{-101}, c'est ce qu'effectue la boucle suivante :

```
> i :=1:
> l := 0:
> ev := evalf(eval(sol, n=1)):
> for l from 0 while abs(ev)>10^(-101) do
> i := i*2;
> ev := evalf(eval(sol, n=i));
> od:
> i;
```

256

Pour calculer l'approximation, on utilise la procédure fnth_term disponible dans NumGfun avec l'option *series* qui permet de calculer rapidement une approximation en arithmétique flottante de la somme des premiers termes de la suite $(c_{9,k})_{k\in\mathbb{N}}$ à précision fixée. On souhaite ici calculer les 256 premiers termes à précision 100 chiffres. Ce qu'effectue la commande suivante :

```
> k1 := 9:
> fnth_term(rec, u(n), 256, 100, 'series');
```

$$0.0000001063392639\ldots$$

On peut calculer aussi ce coefficient en utilisant l'intégrale et comparer les 100 premiers chiffres des deux résultats.

```
> evalf[120](2/Pi*Int(f(x)*orthopoly[T](k1, x)/sqrt(1-x^2), x=-1..1)-%);
```

$$-3.09917579327244 * 10^{-112}$$

On retrouve bien les mêmes 100 premiers chiffres.

L'évaluation du coefficient s'effectue en 0.848 secondes (0.5 secondes pour le calcul de la borne qui valide le résultat et le reste pour le calcul du coefficient) alors que le calcul numérique des 100 premières décimales de l'intégrale en Maple s'effectue en 1.918 secondes sans garantie du résultat.

4.2 Utilisation de la récurrence de Tchebychev pour calculer les M premiers coefficients

Le problème qui nous préoccupe dans cette partie est le calcul efficace des M premiers coefficients de Tchebychev d'une fonction D-finie f. Ces coefficients se calculent naturellement en itérant M fois l'algorithme 6.2. La complexité de ce calcul est alors $\mathcal{O}(M + M\log(\epsilon^{-1}))$ opérations arithmétiques. Je montre maintenant comment ce calcul s'effectue en $\mathcal{O}(\log(\epsilon^{-1}))$ opérations arithmétiques en utilisant la récurrence de Tchebychev.

Le problème qui a déjà été évoqué dans l'introduction est qu'en déroulant naïvement la récurrence, les approximations des coefficients deviennent très mauvaises. Ceci est une

conséquence du théorème 5.35 page 110. En effet, la dimension de l'espace des solutions convergentes d'une récurrence de Tchebychev d'ordre $2s$ est s. En déroulant directement l'algorithme avec des conditions initiales non exactes (par rapport aux conditions initiales d'une solution convergente), on calculera probablement les coefficients d'une solution divergente de la récurrence. On peut observer ce phénomène sur l'exemple suivant.

Exemple 6.15. On souhaite calculer les coefficients de Tchebychev de la fonction exp en utilisant la récurrence qu'ils vérifient. On utilise une bonne approximation numérique des conditions initiales et on déroule la récurrence. De cette manière on obtient vite des résultats faux, comme le montre la session Maple suivante.

```
> Digits := 100:
> u_0 := evalf(BesselI(0,1)): u_1:= evalf(BesselI(1,1)):
> rec := -u(n)+(2*n+2)*u(n+1)+u(n+2):
> v := gfun[rectoproc]({rec, u(0)=u_0, u(1)=u_1}, u(n)):
> v(62), evalf[10](BesselI(62,1));
```

$$400.0, 6.917787740000000 \times 10^{-105}$$

Dans cette session, on travaille avec 100 chiffres de précision. Les coefficients `u_0` et `u_1` sont des approximations des deux premiers coefficients de Tchebychev de la fonction exp, `rec` est la récurrence satisfaite par ces coefficients et `v` est la suite obtenue en déroulant cette récurrence avec comme conditions initiales `u_0` et `u_1`. Le résultat retourné est le $62^{\text{ème}}$ terme de la suite v, qui est censé représenter une approximation du $62^{\text{ème}}$ coefficient de Tchebychev de cette fonction. Le résultat obtenu est très loin du résultat espéré (une approximation de $\mathrm{I}_{62}(1)$). La seule raison de cette différence est la structure de la récurrence de Tchebychev qui fait que l'on ne peut jamais obtenir une bonne approximation des coefficients en utilisant cette méthode. L'utilisation de la récurrence de cette manière n'est donc pas satisfaisante.

En revanche, la proposition 6.16 montre que la suite indexée par k de terme général :

$$\sum_{n=0}^{N} c_{n,k}, \tag{6.25}$$

vérifie pour tout N une récurrence inhomogène dérivée de la récurrence de Tchebychev, et dont on peut faire usage numériquement en gardant une bonne complexité.

Proposition 6.16. *Soit f une fonction D-finie et analytique en 0. Les coefficients de Tchebychev $c_k(t)$ de $f(xt)$ définis par (6.6) page 115 vérifient une récurrence du type :*

$$\left(p_{0,0}(k) + p_{0,1}(k)t + \cdots + p_{0,l}(k)t^l\right) c_k(t) + \cdots + \left(p_{d,0}(k) + \cdots + p_{d,l}(k)t^l\right) c_{k+d}(t) = 0, \tag{6.26}$$

où les coefficients $p_{i,j}$ sont des polynômes en k et l est un entier naturel. La suite

$$\sum_{n=0}^{N} c_{n,k},$$

où $c_{n,k}$ est le $n^{ème}$ coefficient de Taylor de $c_k(t)$, est alors solution de la récurrence inhomogène :

$$\begin{aligned}\big(p_{0,0}(k)+\cdots+p_{0,l}(k)\big)\sum_{n=0}^{N}c_{n,k}+\cdots+\big(p_{d,0}(k)+\cdots+p_{d,l}(k)\big)\sum_{n=0}^{N}c_{n,k+d}=\\ \big(p_{0,1}(k)+\cdots+p_{0,l}(k)\big)\,c_{N,k}+\cdots+\big(p_{d,1}(k)+\cdots+p_{d,l}(k)\big)\,c_{N,k+d}\\ +\big(p_{0,2}(k)+\cdots+p_{0,l}(k)\big)\,c_{N-1,k}+\cdots+\big(p_{d,2}(k)+\cdots+p_{d,l}(k)\big)\,c_{N-1,k+d}\\ \ddots\\ +p_{0,l}(k)c_{N-l+1,k}+\cdots+p_{d,l}(k)c_{N-l+1,k+d}.\end{aligned}\tag{6.27}$$

Démonstration. L'équation différentielle en x vérifiée par $f(xt)$ est à coefficients polynomiaux en x et t. Selon le chapitre 5, il existe une récurrence à coefficients polynomiaux en t et en k qui annule $c_k(t)$. Il existe donc un entier l (le maximum des degrés en t des polynômes de la récurrence) tel que la récurrence s'écrit comme (6.26). Pour tout entier N, l'équation (6.26) implique l'égalité :

$$\begin{aligned}\left(p_{0,0}(k)\sum_{n=0}^{N}c_{n,k}t^n+p_{0,1}(k)\sum_{n=0}^{N-1}c_{n,k}t^{n+1}+\cdots+p_{0,l}(k)\sum_{n=0}^{N-l}c_{n,k}t^{n+l}\right)+\cdots\\ +\left(p_{d,0}(k)\sum_{n=0}^{N}c_{n,k+d}t^n+\cdots+p_{d,l}(k)\sum_{n=0}^{N-l}c_{n,k+d}t^{n+l}\right)=0.\end{aligned}$$

En effet, le membre gauche de cette équation est le membre gauche de l'équation (6.26) tronqué à l'ordre t^{N+1}. En évaluant t en 1 dans cette équation puis en isolant la somme $\sum_{n=0}^{N}c_{n,k}$, on obtient :

$$\begin{aligned}\big(p_{0,0}(k)+\cdots+p_{0,l}(k)\big)\sum_{n=0}^{N}c_{n,k}+\cdots+\big(p_{d,0}(k)+\cdots+p_{d,l}(k)\big)\sum_{n=0}^{N}c_{n,k+d}=\\ p_{0,1}(k)c_{N,k}+\cdots+p_{0,l}(k)\sum_{n=N-l+1}^{N}c_{n,k}+\\ \ddots\\ +p_{d,1}(k)c_{N,k+d}+\cdots+p_{d,l}(k)\sum_{n=N-l+1}^{N}c_{n,k+d}.\end{aligned}\tag{6.28}$$

En réarrangeant les termes de la partie gauche de cette équation, on retrouve l'équation (6.27). □

Pour tout N fixé, la suite $(c_{N,k})_{k\in\mathbb{N}} = (u_N g_{N,k})_{k\in\mathbb{N}}$ vérifie la même récurrence que la suite $(g_{N,k})_{k\in\mathbb{N}}$ (6.12) page 117. Le membre droit de l'équation (6.27) se calcule donc efficacement en utilisant cette récurrence. On déduit alors un algorithme pour calculer rapidement les développements de coefficients (6.25) en utilisant la récurrence (6.27).

Entrée: *deq* une équation différentielle n'admettant pas de singularité dans le disque unité et munie des conditions initiales à valeurs dans $\mathbb{K}$ donnant une solution unique f, un entier M et une erreur cible ϵ

Sortie: Des approximations dans $\mathbb{K}$ de $\frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx$ pour k allant de 0 à M avec une erreur $\leqslant \epsilon$.

1: Calculer rec_k, la récurrence (6.20) page 126 à partir de l'algorithme 6.1.
2: Calculer les conditions initiales de rec_k à partir de (6.23). Soit $(c_{n,k})_n n \in \mathbb{N}$, la solution de cette récurrence et des conditions initiales.
3: Calculer N tel que $|\sum_{n=N+1}^{\infty} c_{n,0}| < \epsilon$ et $|\sum_{n=N+1}^{\infty} c_{n,1}| < \epsilon$ (utiliser les algorithmes de [MS10])
4: Calculer la récurrence de Tchebychev associée à *deq* et identifier les polynômes $p_{i,j}$ tels que la récurrence soit sous la forme (6.26) (avec d l'ordre de la récurrence et l le degré des polynômes en t)
5: Calculer les coefficients $\sum_{n=0}^{N} c_{n,k}$ pour k allant de 0 à $d-1$ avec l'algorithme 6.2
6: Calculer les coefficients $c_{N,0}, c_{N-1,0}, \ldots, c_{N-l+1,0}, c_{N,1}, \ldots, c_{N-l+1,1}$ en utilisant rec_k et les conditions initiales calculées à l'étape 2
7: Calculer les coefficients $(c_{N,2}, \ldots, c_{N,M}), \ldots, (c_{N-l+1,2}, \ldots, c_{N-l+1,M})$ en utilisant la récurrence (6.12) page 117 et les conditions initiales calculées à l'étape précédente
8: **renvoyer** $\sum_{n=0}^{N} c_{n,k}$ pour k allant de 0 à M en utilisant l'équation (6.27)

Algorithme 6.3: Calcul efficace des M premiers coefficients de Tchebychev

Proposition 6.17. *Pour toute fonction f solution d'équation différentielle linéaire à coefficients dans $\mathbb{K}[x]$, n'admettant pas de singularité dans le disque unité, munie de conditions initiales dans $\mathbb{K}$, tout entier k et tout $\epsilon > 0$, l'algorithme 6.3 calcule en $\mathcal{O}(M + \log(\epsilon^{-1}))$ opérations arithmétiques les sommes $\sum_{n=0}^{N} c_{n,k}$ pour k allant de 0 à M où $c_{n,k}$ est défini par :*

$$c_k(t) = \sum_{n\in\mathbb{N}} c_{n,k}t^n = \frac{2}{\pi}\int_{-1}^{1}\frac{f(xt)T_k(x)}{\sqrt{1-x^2}}dx,$$

et N est tel que $|c_k(t) - \sum_{n=0}^{N} c_{n,k}| < \epsilon$ pour tout $k \in \{0, \ldots, M\}$.

Démonstration. La correction de cet algorithme est une conséquence de la proposition 6.16. Seule l'étape 3 est à vérifier ; on doit montrer que l'entier N calculé vérifie bien $\left|\sum_{n=N+1}^{\infty} c_{n,k}\right| \leqslant \epsilon$ pour tout k. Cette propriété est vérifiée, en montrant les inégalités :

$$\sum_{n=N+1}^{\infty}|c_{n,0}| \leqslant \sum_{n=N+1}^{\infty}|c_{n,2k}| \quad \text{et} \quad \sum_{n=N+1}^{\infty}|c_{n,1}| \leqslant \sum_{n=N+1}^{\infty}|c_{n,2k+1}|,$$

pour tout $k \in \mathbb{N}$ et $N \in \mathbb{N}$. Comme la suite $(c_{n,k})_{k\in\mathbb{K}}$ est définie comme le produit de u_n avec la suite $(g_{n,k})_{k\in\mathbb{N}}$; le fait que $|g_{n,k}| \leqslant |g_{n,k+2i}|$ suffit à démontrer ces inégalités. On a selon la formule (6.12) page 117 :

$$\frac{g_{n,k+2i}}{g_{n,k}} = \frac{((n-k)/2)!}{((k+n)/2+1)!}, \quad n \geqslant k.$$

Ce quotient est bien inférieur à 1.

Pour les mêmes raisons que dans la preuve de la proposition 6.13 page 131, l'étape 1 s'effectue en temps constant et l'étape 5 en $\mathcal{O}(M)$ opérations arithmétiques. Le calcul de borne est effectué seulement pour $k \in \{0, 1\}$. Le calcul de l'entier N à l'étape 3 s'effectue donc en $\mathcal{O}(\log(\epsilon^{-1}))$ opérations arithmétiques. Selon [Mez11], l'entier N est dans $\mathcal{O}\left(\log(\epsilon^{-1})\right)$.

Les autres étapes de cet algorithme calculent des coefficients à l'aide de récurrences, le nombre de coefficients à calculer étant M (étape 7) ou N (étapes 5, 6 et 8). Ces calculs s'effectuent donc en $\mathcal{O}(N + M)$ opérations ce qui est bien la complexité recherchée. □

Remarque 6.18. Pour $\epsilon > 0$, le calcul des approximants $\sum_{n=0}^{N} c_{n,k}$ de tous les coefficients de Tchebychev $c_k(1)$ avec une erreur $\leqslant \epsilon$ s'effectue en $\mathcal{O}(\log(\epsilon^{-1}))$.

En effet les coefficients $c_{n,k}$ sont nuls pour $n < k$, les approximants $\sum_{n=0}^{N} c_{n,k}$ sont donc nuls pour $k > N$. Dans l'algorithme 6.3 on peut donc ne pas mettre M dans l'entrée et calculer N coefficients.

Exemple 6.19. Cet exemple est la suite de l'exemple 6.14.

On souhaite maintenant calculer plusieurs coefficients de Tchebychev de la fonction d'Airy Ai avec une erreur de 10^{-100}. On utilise encore pour cette opération $N = 256$. La procédure NumChebyCoeff que j'ai développée lors de cette thèse permet d'effectuer ce calcul. On définit dans un premier temps une équation différentielle annulant la fonction d'Airy.

```
> deq := gfun[holexprtodiffeq](AiryAi(x), y(x)):
```

Le calcul des 10 premiers coefficients s'effectue efficacement en utilisant cette procédure :

```
> nb_coeffs:=10: N:=256:
> st:=time():
> L:=NumChebyCoeff(deq, y(x), nb_coeffs, N):
> time()-st;
```

$$0.462$$

Le nombre retourné par `time()-st` est le temps qu'il faut pour évaluer la fonction. Il faut donc ici seulement 1/2 secondes pour l'évaluer. On peut vérifier que la borne N est correcte :

```
> evalf[200](L[9]-
>   2/Pi*Int(AiryAi(x)*orthopoly[T](9,x)/sqrt(1-x^2), x=-1..1));
```

$$-1.503000000 \times 10^{-203}$$

Dans cet algorithme une grosse partie du temps est passée dans le calcul des récurrences et des conditions initiales, ce qui explique que le calcul de 256 termes de la récurrence ne soit pas beaucoup plus long que le calcul des 10 premiers termes :

```
> nb_coeffs:=256:
> st:=time():
> L2:=NumChebyCoeff(deq, y(x), nb_coeffs, N):
> time()-st;
```

$$0.615$$

5 Conclusion

En considérant les coefficients de Tchebychev de la fonction $f(xt)$, on obtient des formes closes pour les coefficients de Tchebychev des fonctions hypergéométriques et un algorithme pour le calcul numérique de ces coefficients.

Dans ce chapitre, le calcul numérique des coefficients est restreint aux fonctions analytiques dans le disque unité. Une piste, pour améliorer cette méthode aux fonctions analytiques non pas dans le disque unité mais sur le segment $[-1,1]$, serait d'adapter l'algorithme 6.3 avec une méthode de prolongement analytique afin d'évaluer les coefficients de Tchebychev $c_k(t)$ en 1.

Une autre idée pour calculer les coefficients de Tchebychev de fonction ayant des singularités plus proches de 0 est développée dans le chapitre 7.

L'utilisation du produit d'Hadamard peut se généraliser à d'autres séries comme celles discutées dans le chapitre 7. On pourrait alors obtenir de nouvelles formes closes et aussi des algorithmes de calcul numérique pour les développements dans d'autres familles de fonctions.

En plus de ce produit d'Hadamard, l'existence des formules donnant des formes closes repose sur la récurrence à deux termes (6.12) page 117 vérifiée par les coefficients $g_{n,k}$. Un grand nombre de familles de fonctions admettent des coefficients du même type que $g_{n,k}$ qui vérifient une récurrence à deux termes. On a par exemple les développements de x^n en séries de Gegenbauer, Laguerre et Hermite suivants [Nat10, http://dlmf.nist.gov/18.18.iv] :

$$(2x)^n = n! \sum_{\ell=0}^{\lfloor n/2 \rfloor} \frac{\lambda+n-2\ell}{\lambda} \frac{1}{(\lambda+1)_{n-\ell}\,\ell!} C_{n-2\ell}^{(\lambda)}(x)\,,$$

$$x^n = (\alpha+1)_n \sum_{\ell=0}^{n} \frac{(-n)_\ell}{(\alpha+1)_\ell} L_\ell^{(\alpha)}(x)\,,$$

$$(2x)^n = \sum_{\ell=0}^{\lfloor n/2 \rfloor} \frac{(-n)_{2\ell}}{\ell!} H_{n-2\ell}(x)\,.$$

On peut alors généraliser ce procédé donnant des formes closes pour l'ensemble de ces familles de fonctions.

Chapitre 7

Utilisation des séries de Tchebychev pour l'approximation uniforme sur un segment

Résumé

Un large éventail de méthodes numériques existe pour calculer des approximations polynomiales de solutions d'équations différentielles basées sur les séries de Tchebychev ou sur l'interpolation polynomiale en des points de Tchebychev. Dans ce chapitre nous considérons l'application de telles méthodes dans le contexte du calcul rigoureux, où nous avons besoin d'un moyen pour obtenir des garanties sur l'exactitude du résultat, en prenant en considération à la fois les erreurs de troncature et d'arrondi.

Nous avons vu précédemment que les coefficients d'une série de Tchebychev D-finie étaient solutions d'une récurrence. Néanmoins, dérouler cette récurrence ne suffit pas en général à obtenir les coefficients, en raison d'une part de la difficulté d'accès aux conditions initiales, d'autre part de la présence de nombreuses solutions divergentes.

Dans ce chapitre, nous montrons comment ces récurrences peuvent quand même être utilisées pour calculer une bonne approximation uniforme de fonction D-finie accompagnée d'une borne d'erreur, le tout en complexité arithmétique linéaire. Notre approche se fonde sur une méthode numérique classique due à Clenshaw, combinée à une technique d'arithmétique des intervalles pour les solutions d'équations différentielles.

Ce chapitre représente une version préliminaire d'un travail en commun avec Mioara Joldeş et Marc Mezzarobba [BJM]. Une version est aussi disponible dans leurs thèses [Jol11, Mez11].

Sommaire

1 Introduction

1.1 Contexte

Le problème qui nous occupe dans ce chapitre est le suivant :

Problème 7.1. Soient y une fonction D-finie spécifiée par une équation différentielle avec des coefficients polynomiaux, une suite de conditions initiales et un entier d. Comment calculer les coefficients du polynôme :

$$p(x) = \sum_{n=0}^{d}{}' c_n T_n(x)$$

ainsi qu'une borne B « assez fine » telle que $|y(x) - p(x)| \leqslant B$ pour tout $x \in [-1, 1]$.

Une première motivation pour étudier ce problème vient de l'évaluation d'une fonction y en de nombreux points vivant dans un intervalle, le plus souvent à précision modérée. Les exemples incluent le tracé de graphes, l'intégration numérique, le calcul d'approximation *minimax* d'une fonction en utilisant l'algorithme de Remez. Une façon standard pour répondre à ce besoin a recours à des approximations polynomiales de y. Si nous voulons manipuler des fonctions D-finies arbitraires, il est naturel de demander de bonnes approximations polynomiales uniformes de ces fonctions sur un intervalle, avec des bornes d'erreurs rigoureuses, afin que l'ensemble des calculs puisse aboutir à un résultat rigoureux.

Outre l'évaluation numérique aisée, de telles approximations fournissent une représentation commode des fonctions continues sur laquelle on peut définir toute une arithmétique, avec addition, multiplication, composition et intégration. Comparée à la représentation exacte des fonctions D-finies par des équations différentielles, cette représentation est seulement approchée mais plus générale. Pour différentes raisons, il est naturel dans ce contexte d'écrire les polynômes sur la base des Tchebychev plutôt que sur la base monomiale. En particulier, les troncatures qui interviennent durant les opérations arithmétiques sur les approximants préservent de bonnes propriétés d'approximation uniforme. Le logiciel Chebfun de Trefethen *et al.*[Tre07, DBT08] est un système populaire de calcul numérique basé sur cette idée.

Dans un contexte encore plus général, Epstein, et ont développé un formalisme de calcul sur les fonctions mathématiques appelé *ultra-arithmetic* [EMR82a, EMR82b, KM84] qui se veut à celle-ci ce que l'arithmétique en virgule flottante est aux nombres réels. Diverses *séries de Fourier généralisées*, incluant les séries de Tchebychev, y jouent le rôle des flottants. Les objets de base sont alors des séries tronquées à coefficients des intervalles, accompagnées de bornes rigoureuses sur les erreurs de troncatures. Cette approche a été relancée avec l'introduction des « ChebModels » dans un travail récent de Brisebarre et Joldeş [BJ10]. Une deuxième motivation à la suite du problème 7.1 est l'utilisation des fonctions D-finies quelconques comme « fonctions de bases » aux feuilles et aux nœuds des arbres d'expressions qui sont évalués en utilisant les ChebModels.

Finalement, le principal attrait de l'*ultra-arithmetic* et des techniques apparentées est peut-être la possibilité de résoudre rigoureusement des équations fonctionnelles par des méthodes d'inclusion. Les équations différentielles linéaires à coefficients polynomiaux sont parmi les plus simples auxquelles s'applique cette approche. Un troisième objectif de ce chapitre est de contribuer à l'étude de la complexité des méthodes d'inclusion, du point de vue du calcul formel, en prenant comme prototype cette famille de problèmes.

1.2 Données

Dans ce chapitre, on fixe une équation différentielle linéaire homogène à coefficients polynomiaux

$$L \cdot y = a_r y^{(r)} + a_{r-1} y^{(r-1)} + \cdots + a_0 y = 0, \quad a_i \in \mathbb{Q}[x]. \tag{7.1}$$

Quitte à effectuer un changement de variables, on recherche une approximation polynomiale d'une solution y de (7.1) sur le segment $[-1, 1]$. La norme uniforme sur cet intervalle est notée $\|\cdot\|_\infty$. On suppose aussi que $a_r(x) \neq 0$ pour $x \in [-1, 1]$, de sorte que toutes les solutions de (7.1) sont analytiques sur $[-1, 1]$, et que sont données r conditions au bord indépendantes (dans le sens que les applications $\lambda_i : \ker L \to \mathbb{C}$ sont linéairement indépendantes)

$$\lambda_i(y) = \ell_i, \quad 1 \leqslant i \leqslant r, \tag{7.2}$$

chacune de la forme $\lambda_i(y) = \sum_{j=1}^{q} \mu_j y^{(r_j)}(x_j)$ avec $x_j \in [-1; 1]$ et $r_j \leqslant r$, telle que la fonction y est l'unique solution de (7.1) satisfaisant (7.2). Le cas de conditions initiales données en dehors de l'intervalle de développement peut être ramené à nos hypothèses par des algorithmes de prolongement analytique.

Modèle de complexité Nous supposons dans ce chapitre que les opérations arithmétiques entre les nombres réels ou complexes sont effectuées en arithmétique exacte (c'est-à-dire en général rationnelle). Cette hypothèse est faite pour plus de simplicité : la rigueur des calculs n'est pas affectée si l'arithmétique exacte est remplacé par l'arithmétique à virgule flottante dans l'algorithme 7.1 et par l'arithmétique des intervalles dans l'algorithme 7.4 (et l'algorithme 7.2, même si cela implique quelques ajustements). Cependant, nous n'analysons pas l'effet des erreurs d'arrondi sur la qualité du polynôme p (voir problème 7.1) ou celle de la borne d'erreur B. Au moins dans les cas simples, nous nous attendons à ce que l'algorithme 7.1 exhibe une stabilité comparable aux méthodes basées sur les récurrences déroulées « à reculons » (voir [Wim84]). Certaines expériences qui montrent un comportement numérique satisfaisant sont données dans §5.

1.3 Résumé des résultats et plan du chapitre

Nous décrivons une méthode de calcul d'approximations polynomiales des fonctions D-finies avec les caractéristiques suivantes. Soit y une fonction D-finie analytique sur le segment $[-1, 1]$, et notons $\|\cdot\|_\infty$ la norme uniforme sur ce segment. Pour un degré d donné, notre méthode renvoie un polynôme p de degré d ainsi qu'une borne rigoureuse et

fine B sur la différence $\|y-p\|_\infty$. Le polynôme p est obtenu comme une approximation du développement en série de Tchebychev de y tronquée à l'ordre d.

Sous des hypothèses convenables, la complexité arithmétique du procédé est linéaire en $d+\log(\epsilon^{-1})$, où ϵ est lié à la précision de calcul des coefficients de p ainsi qu'à la borne B. Si la fonction y à approcher n'est pas elle-même un polynôme de degré inférieur ou égal à d, on attend « en général » dans un sens volontairement vague,

$$\max_x |y(x)-p(x)| \approx \epsilon \approx 2^{-\Theta(d)}$$

de sorte que les termes d et $\log(\epsilon^{-1})$ de la borne de complexité sont du même ordre de grandeur.

À titre de comparaison, en supposant que le développement en série de Taylor de y converge, ses troncatures successives fournissent des approximations polynomiales de y d'erreur bornée par 2^{-e}, $e=\Omega(d)$, sur tout compact inclus dans leur disque de convergence, et peuvent être calculées en $\mathcal{O}(d)=\mathcal{O}(e)$ opérations arithmétiques. Nos approximations ne sont pas meilleures selon ce critère : le temps nécessaire pour obtenir l'erreur inférieure à 2^{-e} reste $\Omega(e)$. En revanche, il ne suffit pas que y soit analytique sur $[-1,1]$ pour que son développement en série entière à l'origine converge sur ce segment. De façon liée, la constante que cache l'écriture $e=\Omega(d)$ dans le cas des séries de Taylor peut devenir arbitrairement grande quand y varie, même si y admet en fait de bien meilleures approximations de degré d.

Cependant, la constante cachée dans la relation $e=\Omega(d)$ peut être arbitrairement petite en fonction de y, même si des meilleurs approximations polynomiales de degré d existent réellement.

L'objet de notre étude est ainsi d'égaler le coût linéaire de l'approximation par séries de Taylor, tout en calculant des approximations dont la qualité est proche de l'optimum, uniformément en y. Dans l'état actuel de ce travail, nous n'énonçons cependant pas de lien précis entre la borne B produite par notre méthode et l'erreur optimale (« minimax »).

Le choix du modèle de complexité arithmétique pour un algorithme où interviennent des calculs numériques multi-précision pourrait surprendre. Observons cependant que toutes les opérations d'arithmétique de base sur les rationnels et les nombres à virgule flottante de taille bornée par n peuvent être effectuées en $\mathcal{O}(n(\log n)^{\mathcal{O}(1)})$ opérations [BZ10]. Sauf si des annulations exceptionnelles se produisent, la taille maximale en bit des nombres que l'on manipule est à peu près la même que celles des coefficients de p (à savoir $\mathcal{O}(d\log d)$ lorsque ceux-ci sont représentés par des rationnels). On s'attend ainsi à une complexité binaire quasi-linéaire en la taille totale du polynôme d'approximation calculé.

Notre algorithme procède en deux étapes. On calcule d'abord un polynôme d'approximation candidat, à partir du développement en série de Tchebychev de la fonction f. Nous ne tentons pas de contrôler rigoureusement les erreurs à ce stade. Nous validons dans un second temps ce polynôme en utilisant les méthodes d'inclusion.

Ce chapitre est organisé comme suit. Dans §2, nous donnons un algorithme pour calculer les coefficients de Tchebychev d'une fonction D-finie à partir de la récurrence qu'ils

vérifient (voir 5). Cet algorithme rappelle la variante de Fox et Parker [FP68, Chap. 5] de l'algorithme de Clenshaw [Cle57]. En général, sa sortie est une approximation de qualité indéterminée. Sous quelques hypothèses simplificatrices **H1** et **H2**, nous montrons que le polynôme calculé peut être rendu arbitrairement proche du développement de Tchebychev de y tronqué au degré d. Dans la section §3, nous étudions le développement en série de Tchebychev de fonction rationnelle. Plus important encore, nous calculons une borne sur l'erreur dans §4, avec une méthode d'inclusion pour les équations différentielles, afin de valider la sortie du premier algorithme et obtenir une borne B. La section 5 présente quelques résultats expérimentaux.

2 Calcul des coefficients

2.1 L'algorithme de Clenshaw revisité

On se propose dans cette section de donner un algorithme pour calculer les coefficients de Tchebychev d'une fonction D-finie en utilisant la récurrence qu'ils vérifient. Dans les chapitres précédents, on a vu que l'on ne pouvait pas utiliser la récurrence en la déroulant « naïvement ». L'exemple 6.15 page 134 illustre cette impossibilité.

Dans le chapitre 6, un algorithme permettant ce calcul est proposé. Ici, on ne l'utilise pas, on en présente un nouveau pour le calcul des coefficients. Un des intérêts de ce nouvel algorithme est qu'il ne prend pas en entrée uniquement des équations différentielles non singulières dans le disque unité.

Cet algorithme reprend et généralise une méthode due à Clenshaw en 1957 [Cle57], reformulée pour mettre en évidence le rôle (déjà observé par Fox et Parker [Fox62, FP68]) qu'y joue la récurrence de Tchebychev. L'ensemble de ces méthodes et algorithmes sont à rapprocher de la méthode de Miller pour le calcul d'une solution minimale d'une récurrence et de ses généralisations [BCM+52, Wim84].

Un cas particulier simple de la méthode de Miller est le calcul de la solution la plus convergente d'une récurrence d'ordre 2 dont une base de solution est (α^n, β^n) avec $\alpha > \beta$. Pour tout couple u_0, u_1, il existe un couple a, b tel que $u_0 = a+b, u_1 = a\alpha+b\beta$. La solution u_n que l'on obtient en déroulant la récurrence avec les conditions initiales u_0 et u_1 est la suite $a\alpha^n + b\beta^n$. Pour n grand on a alors $u_n \sim a\alpha^n$ et la solution calculée est alors plus « proche » d'une solution rapide que de la solution la plus lente. L'idée de Miller est de calculer les coefficients $u_N, u_{N-1}, \ldots, u_0$ d'une suite récurrente linéaire « à reculons », à partir de conditions initiales u_N, u_{N-1} prises au hasard au voisinage de $N \gg 0$. On peut encore calculer a et b tel que $u_{N-1} = a+b$ et $u_N = a\alpha+b\beta$. La solution obtenue en déroulant « à reculons » est alors $u_n = \frac{a}{\alpha^{N-1}}\alpha^n + \frac{b}{\beta^{N-1}}\beta^n$. C'est alors le plus rapidement décroissant des comportements possibles des solutions de la récurrence qui domine. Les termes calculés sont donc proches de ceux d'une solution minimale de la récurrence. La méthode de Miller généralise cette idée pour des récurrences d'ordres quelconques.

L'intérêt est double : premièrement, cette méthode est bien plus stable numériquement que le calcul de $u_0, u_1, \ldots$ pour n croissant qui serait au contraire parasité par une contri-

bution issue de la solution dominante après la moindre erreur d'arrondi. Deuxièmement, on parvient ainsi à caractériser et approcher la solution minimale *via* son comportement asymptotique, sans avoir besoin de connaître au départ les conditions initiales correspondantes.

Plus généralement, si l'on calcule ainsi s solutions test linéairement indépendantes, on s'attend à ce que leurs restrictions à $[\![0, n]\!]$ pour $n < N$ engendrent un espace vectoriel « proche » de celui qu'on obtiendrait avec « les s plus convergentes » des solutions de base données par le théorème de Perron-Kreuser.

En utilisant cette idée et les résultats de la section 5 on en déduit l'algorithme 7.1 pour calculer une approximation (non prouvée) des coefficients de Tchebychev d'une solution d'équation différentielle. Cet algorithme déroule « à reculons » $s + k$ solutions test linéairement indépendantes de la récurrence de Tchebychev associée à l'équation (7.1) page 142. Il en cherche ensuite une combinaison linéaire qui est symétrique (c'est-à-dire $u_n = u_{-n}$) et qui satisfait les conditions au bord (7.2). Il prend en entrée à la fois le degré d du polynôme recherché, et un paramètre N indiquant à partir de quel rang calculer les solutions test. Nous étudierons plus loin la façon dont la qualité de l'approximation qu'il renvoie évolue avec N. En pratique, adopter simplement $N = d + s$ donne des résultats satisfaisants.

Proposition 7.2. *L'algorithme 7.1 s'exécute en $\mathcal{O}(N)$ opérations.*

La preuve de convergence en section suivante relie, dans les cas simples, le choix de N à la qualité de l'approximation de $\pi_d(y)$ par p. Sous ces hypothèses de convergence, l'étape 7 est justifiée par le lemme suivant.

Lemme 7.3. *En utilisant les notations de l'algorithme 7.1, le système*

$$\begin{cases} \lambda_k(y) = \ell_k, & 1 \leqslant k \leqslant r \\ b_{-s}(n)y_{n-s} + \cdots + b_s(n)y_{n+s} = 0, & n \in [\![r, s-1]\!] \end{cases} \tag{7.6}$$

restreint à l'espace des solutions convergentes de l'opérateur de récurrence P, admet une unique solution u_n. La série $\sum u_n T_n(x)$ est la solution de l'équation différentielle (7.1) et des conditions au bord (7.2) page 142.

Démonstration. Il est clair que la suite des coefficients de Tchebychev de la solution de l'équation différentielle 7.1 munie des conditions au bord (7.2) est solution du système (7.6). Il reste à montrer l'unicité de la solution.

Par la proposition 5.29 page 107, on peut prouver, comme dans la preuve du corollaire 5.31 page 108, que si une solution y_n de la récurrence n'est pas symétrique, alors cette solution n'est pas une solution du système

$$b_{-s}(n)y_{n-s} + \cdots + b_s(n)y_{n+s} = 0, \ n \in [\![r, s-1]\!].$$

Nous en déduisons que l'espace des solutions de ce système restreint à l'espace des solutions convergentes de P est l'espace des solutions convergentes et symétriques de P.

Entrée: Un opérateur L d'ordre r et des conditions au bord $\lambda_1(y) = \ell_1, \dots, \lambda_r(y) = \ell_r$ comme dans (7.2), un degré $d > s$ (où s est le demi-ordre de L), et un entier $N \geqslant d$.
Sortie: Une approximation polynomiale $\tilde{y}(x) = \sum_{n=-d}^{d} \tilde{y}_n T_n(x)$ de la solution y de $L \cdot y = 0$ satisfaisant les conditions au bord.

1: calculer la récurrence de Tchebychev $P = \sum_{k=-s}^{s} b_k(n) S^k$ (utiliser l'algorithme 5.6 page 102) associée à L
2: $\mathbf{S} = \{n \mid b_{-s}(n) = 0\}$ et $\mathbf{I} = \mathbf{S} \cup [\![N, N+s-1]\!]$
3: **pour** $i \in \mathbf{I}$ **faire**
4: En utilisant la relation de récurrence « à reculons », calculer les coefficients $t_{i,N}, \dots, t_{i,0}$ de la suite $(t_{i,n})_{n\in\mathbb{N}}$ telle que $P \cdot t_i = 0$ en partant des conditions initiales pour $i \in [\![N, N+s-1]\!]$

$$\begin{cases} t_{i,i-s} &= 1 \\ t_{i,j-s} &= 0, \quad j \in [\![N, N+2s-1]\!] \setminus \{i\} \cup \mathbf{S} \end{cases} \tag{7.3}$$

et pour $i \in \mathbf{S}$

$$\begin{cases} t_{i,i-s} &= 1 \\ t_{i,j-s} &= 0, \quad j \in [\![i+s+1, N+2s-1]\!] \cup (\mathbf{S} \cap \{n < i\}) \end{cases} \tag{7.4}$$

5: **fin pour**
6: poser $\tilde{y}_n = \sum_{i\in\mathbf{I}} \eta_i t_{i,|n|}$ pour $|n| \leqslant N$, et $\tilde{y}_n = 0$ pour $|n| > N$ (d'où $\eta_i = \tilde{y}_i$), et $\tilde{y}(x) = \sum_{n=-N}^{N} \tilde{y}_n T_n(x)$
7: résoudre pour $(\eta_i)_{i\in\mathbf{I}}$ le système linéaire

$$\begin{cases} \lambda_k(\tilde{y}) = \ell_k, & 1 \leqslant k \leqslant r \\ b_{-s}(n)\tilde{y}_{n-s} + \cdots + b_s(n)\tilde{y}_{n+s} = 0, & n \in [\![r, s-1]\!] \cup \mathbf{S} \end{cases} \tag{7.5}$$

8: **renvoyer** $\sum_{n=-d}^{d} \tilde{y}_n T_n(x)$

Algorithme 7.1: Algorithme de Clenshaw revisité

Il reste à montrer que le système

$$\lambda_k(y) = \ell_k, \ 1 \leqslant k \leqslant r$$

restreint à l'espace des solutions convergentes symétriques de P admet une unique solution. Le théorème 5.24 page 105 dit que si y est une solution convergente de P, alors $\sum' y_n T_n(x)$ est une solution de l'équation différentielle L. Nous savons qu'il existe une unique solution de l'équation différentielle avec les conditions au bord représentés par ce système, nous en déduisons l'unicité de la solution du système (7.6). □

Remarque 7.4. Il est concevable qu'existent des équations différentielles auxquelles l'algorithme 7.1 est inapplicable, au sens où il échoue *pour tout N assez grand*. Cela se produit si l'espace engendré par les $s+k$ solutions test intersecte systématiquement l'espace des solutions symétriques avec dimension strictement plus grande que r.

Nous n'avons à ce jour ni d'exemple explicite de ce phénomène ni preuve qu'il n'arrive jamais. Cependant, lorsque les hypothèses de la section suivante sont vérifiées, la terminaison sans échec découle de la preuve de convergence. Une manière de remédier au problème en général serait de tirer au hasard des conditions initiales $(t_{i,n})_{n\in\mathbf{i}}$ qui définissent les solutions tests. En vue de la preuve de convergence, nous nous en tenons ici à la version où les conditions initiales sont fixées à l'identité.

2.2 Convergence

Nous montrons maintenant que notre algorithme converge, sous deux hypothèses simplificatrices **H1** et **H2**. La preuve est inspirée de l'analyse de l'algorithme de Miller généralisé [Zah76, Wim84].

Lorsque ces hypothèses ne sont pas satisfaites, notre analyse ne dit rien sur la qualité des approximants qu'il est possible d'obtenir par l'algorithme 7.1. Celui-ci — s'il n'échoue pas — calcule tout de même une certaine approximation de $\pi_d(y)$ (le meilleur approximant polynomial de y d'ordre d), qu'il demeure possible de valider en calculant une borne d'erreur par l'algorithme de §4.

Nos hypothèses sont les suivantes :

H1 Les racines complexes des équations caractéristiques d'une même arête du polygone de Newton (voir §5.3) de l'opérateur P sont simples et de module deux à deux distincts.

H2 L'opérateur P n'a pas de singularité de queue aux indices $n \leqslant s$. Autrement dit, en reprenant les notations de l'algorithme 7.1, $\mathbf{S} = \varnothing$.

L'hypothèse **H1** permet d'appliquer la forme forte du théorème de Perron-Kreuser (voir §5.3) : soit

$$e_{-s}, \dots, e_{-1}, e_1, \dots, e_s$$

une base de germes à l'infini de solutions de la récurrence $P \cdot y = 0$ telles que :

$$\forall i,\ \frac{e_{i,n+1}}{e_{i,n}} \sim \alpha_i n^{\kappa_i},$$

où les α_i et κ_i sont donnés par le polygone de Newton comme rappelé en §5.3. Ces germes se prolongent en des suites solutions de la récurrence sur $\mathbb{N}$ (mais pas nécessairement solutions symétriques ou même solutions sur $\mathbb{Z}$) d'après l'hypothèse **H2**. Cette dernière est là essentiellement pour alléger les notations. Elle entraîne que l'équation $L \cdot y = 0$ n'a pas de solution polynomiale.

Proposition 7.5. *Supposons les hypothèses **H1** et **H2** vérifiées. Avec les notations de l'algorithme 7.1, on fixe L ainsi que les conditions au bord $\lambda_i(y) = \ell_i$, posons $y(x) = \sum'_{n\in\mathbb{N}} y_n T_n(x)$ et soient $y_n^{(N)} = \tilde{y}_n$, $|n| \leqslant N$, les approximations des coefficients de Tchebychev calculés par l'algorithme7.1 (exécuté en arithmétique exacte), vus comme des fonctions du paramètre d'entrée N. Quand $N \to \infty$ on a*

$$\max_{n=-N}^{N} (y_n^{(N)} - y_n) = \mathcal{O}(N^t e_{1,N})$$

pour un certain t indépendant de N.

Cette proposition nous dit que pour un degré d fixé et quand N tend vers l'infini, le polynôme calculé converge au moins exponentiellement vite vers la série de Tchebychev tronquée à l'ordre d de y.

Notre preuve fait appel au lemme suivant.

Lemme 7.6. *Soient des suites $(e_{0,n})_n, \dots, (e_{s-1,n})_n$ telles que*

$$\frac{e_{i,n+1}}{e_{i,n}} \sim_{n\to+\infty} \alpha_i n^{\kappa_i} \quad (\alpha_i \in \mathbb{C} \setminus \{0\}, \kappa_i \in \mathbb{Q})$$

avec $\kappa_0 \leqslant \kappa_1 \leqslant \cdots \leqslant \kappa_{s-1}$ et $\kappa_i = \kappa_j \to \alpha_i \neq \alpha_j$. Le déterminant de Casorati

$$C(n) = \begin{vmatrix} e_{0,n} & e_{1,n} & \cdots & e_{s-1,n} \\ e_{0,n+1} & & & e_{s-1,n+1} \\ \vdots & & & \vdots \\ e_{0,n+s-1} & e_{1,n+s-1} & \cdots & e_{s-1,n+s-1} \end{vmatrix}$$

satisfait alors

$$C(n) \sim_{n\to\infty} e_{0,n}e_{1,n+1}\cdots e_{s-1,n+s-1} \prod_{\substack{i<j \\ \kappa_i=\kappa_j}} \Big(\frac{\alpha_i}{\alpha_j} - 1\Big).$$

Démonstration. Posons $C(n) = e_{0,n}e_{1,n+1}\cdots e_{s-1,n+s-1}C'(n)$. Alors

$$C'(n) = \det\Big(\frac{e_{j,n+i}}{e_{j,n+j}}\Big)_{0\leqslant i,j<s} = \sum_{\sigma\in\mathfrak{S}_s} \varepsilon(\sigma) \prod_{j=0}^{s-1} \frac{e_{j,n+\sigma(j)}}{e_{j,n+j}},$$

où le terme d'indice σ de la somme croît comme $n^{\sum_{j=0}^{s-1}(\sigma(j)-j)\kappa_j}$. Les termes dominants sont ceux pour lesquels $\sum_{j=0}^{s-1}\sigma(j)\kappa_j = \sum_{j=0}^{s-1} j\kappa_{\sigma(j)}$ est maximal, c'est-à-dire, ceux pour lesquels les $\kappa_{\sigma(j)}$ sont en ordre croissant. En utilisant les crochet de Iverson, *i. e.* $[P] = 1$ si P est vraie et $[P] = 0$ sinon, cela se traduit par

$$\begin{aligned} C'(n) &= \sum_{\sigma\in\mathfrak{S}_s} \varepsilon(\sigma) \prod_{j=0}^{s-1} [\kappa_{\sigma(j)} = \kappa_j] \frac{e_{j,n+\sigma(j)}}{e_{j,n+j}} + \mathcal{O}(1) \\ &= \det\Big([\kappa_i = \kappa_j]\frac{e_{j,n+i}}{e_{j,n+j}}\Big)_{0\leqslant i,j<s} + \mathcal{O}(1) \\ &= \prod_{\kappa\in\{\kappa_i\}} \det\Big(\frac{e_{j,n+i}}{e_{j,n+j}}\Big)_{\kappa_i=\kappa_j=\kappa} + \mathcal{O}(1). \end{aligned}$$

Chaque déterminant du dernier produit a la forme

$$\det\Big(\frac{e_{j,n+i}}{e_{j,n+j}}\Big) = \det\Big(n^{(j-i)\kappa}\frac{e_{j,n+i}}{e_{j,n+j}}\Big) \to_{n\to\infty} \det(\alpha_j^{i-j}) = \prod_{\substack{i<j \\ \kappa_i=\kappa_j=\kappa}} \Big(\frac{\alpha_i}{\alpha_j} - 1\Big) \neq 0$$

d'où le résultat. □

Démonstration de la proposition 7.5. Commençons par décrire la sortie de l'algorithme. La suite $(y^{(N)})_{n=-N}^{N}$ calculée s'étend en une solution $(y_n^{(N)})_{n\in\mathbb{Z}}$ de $P\cdot y^{(N)}=0$ caractérisée par les conditions $y_N^{(N)}=\cdots=y_{N+s-1}^{(N)}=0$ de l'étape 4, et du système linéaire (7.5) résolu à l'étape 7. En écrivant les formes linéaires $\lambda_1,\dots,\lambda_r:\mathcal{C}\to\mathbb{C}$ (voir définition 2.14 page 24 pour $\mathcal{C}$) qui expriment les conditions au bord (7.2) comme $\lambda_i(y)=\sum_{n=0}^{\infty}{}'\lambda_{i,n}y_n$, on en définit les « troncatures » $\lambda_i^{(N)}(y)=\sum_{n=0}^{N}{}'\lambda_{i,n}y_n$ qui ont un sens sans supposer y convergente. En abusant un peu des notations, nous appliquerons les λ_i et $\lambda_i^{(N)}$ indifféremment à des fonctions, des séries de Tchebychev formelles ou des suites interprétées comme des suites de coefficients de Tchebychev. On introduit de plus les formes linéaires $\lambda_{r+1}=\lambda_{r+1}^{(N)},\dots,\lambda_s=\lambda_s^{(N)}$ pour écrire les $s-r$ dernières équations (*i. e.*, les contraintes de symétrie des développements de Tchebychev) dans la même forme que les r premières. Ainsi le système (7.5) se réécrit comme

$$\lambda_i^{(N)}(y^{(N)})=\sum_{n=0}^{N}{}'\lambda_{i,n}y_n^{(N)}=\ell_i,\quad 1\leqslant i\leqslant s, \tag{7.7}$$

où l'on a aussi posé $l=\ell_{r+1}=\cdots=\ell_s=0$

Soit maintenant

$$\Delta^{(N)}=\begin{vmatrix} e_{1,N} & \cdots & e_{s,N} & e_{-1,N} & \cdots & e_{-s,N}\\ \vdots & & \vdots & \vdots & & \vdots\\ e_{1,N+s-1} & \cdots & e_{s,N+s-1} & e_{-1,N+s-1} & \cdots & e_{-s,N+s-1}\\ \lambda_1^{(N)}(e_1) & \cdots & \lambda_1^{(N)}(e_s) & \lambda_1^{(N)}(e_{-1}) & \cdots & \lambda_1^{(N)}(e_{-s})\\ \vdots & & \vdots & \vdots & & \vdots\\ \lambda_s^{(N)}(e_1) & \cdots & \lambda_s^{(N)}(e_s) & \lambda_s^{(N)}(e_{-1}) & \cdots & \lambda_s^{(N)}(e_{-s}) \end{vmatrix}, \tag{7.8}$$

et soit $\Delta_j^{(N)}$ le même déterminant où l'on a remplacé la colonne en e_j par $(0,\dots,0,\ell_1,\dots,\ell_s)^{\mathrm{T}}$. D'après les règles de Cramer, la suite $(y_n^{(N)})_n$ se décompose sur la base $(e_j)_{j=-s}^{s}$ de $\ker P\subset\mathbb{C}^{\mathbb{Z}}$ sous la forme

$$y^{(N)}=\sum_{k=-s}^{s}\gamma_k^{(N)}e_k,\qquad \gamma_k^{(N)}=\frac{\Delta_k^{(N)}}{\Delta^{(N)}}. \tag{7.9}$$

La « véritable » suite des coefficients de Tchebychev de la fonction y est donnée par

$$y=\sum_{k=1}^{s}\gamma_k e_k,\qquad \gamma_k=\frac{\Delta_k}{\Delta}, \tag{7.10}$$

où $\Delta=\det(\lambda_i(e_j))_{1\leqslant i,j\leqslant s}$ et Δ_j dénote le déterminant Δ avec la $j^{\text{ème}}$ colonne remplacée par $(\ell_1,\dots,\ell_s)^{\mathrm{T}}$.

Notre objectif est maintenant de prouver que $\gamma_k^{(N)}\to\gamma_k$ aussi vite que $N\to\infty$. On décompose $\Delta^{(N)}$ en quatre blocs carrés comme suit :

$$\Delta^{(N)}=\begin{vmatrix} A & B\\ C & D\end{vmatrix}.$$

Les blocs modifiés dans $\Delta_k^{(N)}$ sont dénotés A_k, B_k, C_k, D_k. On remarque que ces matrices dépendent de N, mais nous abandonnons l'indice explicite pour des raisons de lisibilité.

Les blocs A et B (par le lemme 7.6) aussi bien que C ($\det C \to \Delta \neq 0$ quand $N \to \infty$) ne sont pas singuliers pour N grand. La formule du complément de Shur implique que

$$\Delta^{(N)} = -\det(B)\det(C)\det(I - C^{-1}DB^{-1}A).$$

Posons $\mathbf{e}_j = (e_{j,N}, \dots, e_{j,N+s-1})^{\mathrm{T}}$, le coefficient d'indice (i,j) dans la matrice $B^{-1}A$ satisfait

$$\begin{aligned}
(B^{-1}A)_{i,j} &= \frac{\det(\mathbf{e}_{-1}, \dots, \mathbf{e}_{-i+1}, \mathbf{e}_j, \mathbf{e}_{-i-1}, \dots, \mathbf{e}_{-s})}{\det B} \\
&= \frac{(-1)^{i-1}\det(\mathbf{e}_j, \mathbf{e}_{-1}, \dots, \widehat{\mathbf{e}_{-i}}, \dots, \mathbf{e}_{-s})}{\det(\mathbf{e}_{-1}, \dots, \mathbf{e}_{-s})} \\
&= \mathcal{O}\left(\frac{e_{j,N}e_{-1,N+1}\dots e_{-i+1,N+i-1}e_{-i-1,N+i}\dots e_{-s,N+s-1}}{e_{-1,N}e_{-2,N+1}\dots e_{-s,N+s-1}}\right) \\
&= \mathcal{O}\left(N^{\kappa_{-1}+\dots+\kappa_{-i+1}-(i-1)\kappa_i}\frac{e_{j,N}}{e_{-i,N}}\right) \\
&= \mathcal{O}\left(\frac{e_{j,N}}{e_{-i,N}}\right)
\end{aligned}$$

lorsque $N \to \infty$ par le lemme 7.6. Les hypothèses sur les conditions au bord (7.2) entraînent que $\lambda_{i,n} = O_{n\to\pm\infty}(n^r)$ dans (7.7), donc que les coefficients de D satisfont

$$D_{i,j} = \lambda_i^{(N)}(e_{-j}) = \mathcal{O}(N^r e_{-j,N}).$$

On en tire l'estimation $(DB^{-1}A)_{i,j} = \mathcal{O}(N^r e_{j,N})$ pour la $j^{\text{ème}}$ colonne de $DB^{-1}A$. Comme

$$C_{i,j} = \lambda_i^{(N)}(e_j) = \lambda_i(e_j) + \mathcal{O}(N^r e_{j,N}),$$

on obtient aussi $(C^{-1}DB^{-1}A)_{i,j} = \mathcal{O}(N^r e_{j,N})$ d'où finalement

$$\begin{aligned}
\Delta^{(N)} &= -\det(B)\det(C)(1 - \operatorname{tr}(C^{-1}DB^{-1}A) + \mathcal{O}(\|C^{-1}DB^{-1}A\|^2)) \\
&= -\det(B)(\Delta + \mathcal{O}(N^r e_{1,N})).
\end{aligned}$$

Passons aux déterminants modifiés Δ_k. Pour $k > 0$, le même raisonnement que ci-dessus (sauf que C_k peut maintenant être singulier) conduit à [1]

$$\begin{aligned}
\Delta_k^{(N)} &= -\det(B)\det(C_k - DB^{-1}A_k) \\
&= -\det(B)(\det(C_k) + \mathcal{O}(N^r e_{1,N})) \\
&= -\det(B)(\Delta_k + \mathcal{O}(N^r e_{1,N})),
\end{aligned}$$

1. nous avons même $\Delta_1^{(N)} = \Delta_1 + \mathcal{O}(N^r e_{2,N})$

d'où

$$\gamma_k^{(N)} = \frac{\Delta_k^{(N)}}{\Delta^{(N)}} = \gamma_k + \mathcal{O}(N^r e_{1,N}), \qquad k > 0. \tag{7.11}$$

Dans le cas $k < 0$, écrivons

$$\Delta_k^{(N)} = -\det(C)\det(B_k - AC^{-1}D_k).$$

Les bornes naturelles des coefficients sur A et D entraînent $(C^{-1}D_k)_{i,j} = \mathcal{O}(N^r e_{-j,N+s-1})$ et de là

$$(AC^{-1}D_k)_{i,j} = \mathcal{O}(N^r e_{1,N} e_{-j,N+s-1}) = o(e_{-j,N}),$$

de sorte que l'on a

$$(B_k + AC^{-1}D_k)_{i,j} \sim e_{-j,N+i-1}, \qquad j \neq -k.$$

Pour $j = -k$ cependant, la $j^{\text{ème}}$ colonne de B_k est nulle et celle de D_k est constante, donc

$$(B_k + AC^{-1}D_k)_{i,j} = \mathcal{O}(e_{1,N}), \qquad j = -k.$$

En écrivant sous le signe $\widehat{\ }$ la colonne que l'on supprime de la matrice, il s'ensuit que

$$\begin{aligned}\det(B_k + AC^{-1}D_k) &= \mathcal{O}(e_{-1,N+s-1}\cdots\widehat{e_{k,N+s-1}}\cdots e_{-s,N+s-1}\ e_{1,N}) \\ &= \mathcal{O}\left(\frac{N^\tau \det(B)}{e_{k,N}} e_{1,N}\right)\end{aligned}$$

avec $\tau \leqslant \sum_{j\neq k}(s-j)\kappa_{-j}$, et

$$\begin{aligned}\gamma_k^{(N)} = \frac{\Delta_k^{(N)}}{\Delta^{(N)}} &= \frac{-\det(B)\det(C)N^\tau}{-\det(B)\det(C)(1+\mathcal{O}(e_{1,N}))}\frac{e_{1,N}}{e_{k,N}} \\ &= \mathcal{O}\left(N^\tau \frac{e_{1,N}}{e_{k,N}}\right), \qquad k < 0.\end{aligned} \tag{7.12}$$

En combinant (7.9), (7.10) avec (7.11), (7.12) il vient finalement

$$y_n^{(N)} = y_n + \mathcal{O}\left(N^{\max(r,\tau)} e_{1,N} \sum_{k=1}^{s}\left(e_{k,n} + \frac{e_{-k,n}}{e_{-k,N}}\right)\right)$$

quand $N \to \infty$, uniformément en n. □

2.3 Variantes

Une autre méthode populaire pour le calcul approché des séries de Tchebychev est la méthode τ de Lánczos [Lan38, Lan56]. Il a été observé par Fox [Fox62] et plus tard dans une plus grande généralité (et dans un langage différent) par El Daou, Ortiz et Samara [EDOS93] que les deux méthodes sont en fait équivalentes, dans le sens où on peut les exprimer dans un même cadre et les ajuster de telle façon qu'elles produisent le même résultat. Nous

allons maintenant expliquer comment l'utilisation de la récurrence de Tchebychev s'inscrit dans ce point de vue. Cela jette une lumière différente sur l'algorithme 7.1 et indique comment la récurrence de Tchebychev peut être utilisée dans le contexte de la méthode τ.

Comme dans la section précédente, nous considérons une équation différentielle linéaire $L\cdot y = 0$ d'ordre r, avec des coefficients polynomiaux, et une solution pour laquelle on recherche une approximation polynomiale de degré d. Supposons pour simplifier que l'équation n'a pas de solution polynomiale, *i .e.*, $(\ker L)\cap\mathbb{C}[x] = \{0\}$.

En quelques mots, la méthode τ fonctionne comme suit. La première étape est de calculer $L\cdot p$ où p est un polynôme de degré d avec des coefficients indéterminés. Comme $(\ker L)\cap\mathbb{C}[x] = \{0\}$, le résultat a un degré au moins égal à d. On introduit des inconnues additionnelles $\tau_{d+1},\dots,\tau_{d+m}$ en nombre choisi de sorte que le système

$$\begin{aligned} L\cdot p &= \tau_{d+1}T_{d+1} + \cdots + \tau_{d+m}T_{d+m} \\ \lambda_i(p) &= \ell_i \qquad\qquad (1\leqslant i\leqslant r) \end{aligned} \tag{7.13}$$

ait une solution (unique de préférence). La sortie de l'algorithme est la valeur de p obtenue en résolvant le système ; c'est une solution exacte de la projection $\pi_d(L\cdot y) = 0$ de l'équation différentielle d'origine.

Notons $p = \sum_{n=0}^{d}{}' p_nT_n$ et étendons la suite (τ_n) en posant $\tau_n = 0$ pour $n\notin[\![d+1, d+m]\!]$. Il suit que (7.13) entraîne que $P\cdot(p_n) = \frac{1}{2}Q\cdot(\tau_n)$ où P et Q sont des opérateurs de récurrence donnés par le théorème 5.24 page 105. En notant $\operatorname{Supp} u = \{n|u_n\neq 0\}$, on voit aussi de par la forme explicite de Q que $\operatorname{Supp}(Q\cdot\tau)\subset[\![d, d+m+1]\!]$. Ainsi les coefficients p_n du résultat de la méthode τ sont donnés par la récurrence de Tchebychev, en partant d'un petit nombre de conditions initiales données au voisinage de l'indice $|n| = d$.

Inversement, considérons le polynôme $\tilde y$ calculé par l'algorithme 7.1 et soit $v = \sum_n{}' v_nT_n = L\cdot\tilde y$. On a $P\cdot\tilde y = Q\cdot v$ par le théorème 5.24. Mais la définition de $\tilde y$ dans l'algorithme implique aussi que $(P\cdot\tilde y)_n = 0$ quand $|n|\leqslant N-s$ (comme les $\tilde y_n$, $|n|\leqslant N$ sont des combinaisons linéaires de suites $(t_{i,n})_{|n|\leqslant N}$ calculées en utilisant la récurrence $P\cdot t_i = 0$) ou $|n| > N+s$ (comme $\tilde y_n = 0$ pour $|n| > N$), de sorte que $\operatorname{Supp}(Q\cdot v)\subset[\![N-s, N+s-1]\!]$. Il peut être vérifié que l'opérateur de récurrence P associé à $L = (\frac{\mathrm{d}}{\mathrm{d}x})^r$ par l'algorithme de Paszkowski est $P = \delta_r(n)$: en effet, en utilisant les notations du chapitre 5, il doit satisfaire $Q^{-1}P = I^{-r}$. Ainsi la relation $\delta_r(n)\cdot u = Q\cdot v$ est équivalente à $u^{(r)} = v$, où

$$v(x) = \frac{\mathrm{d}^r}{\mathrm{d}x^r}\sum_{|n|>r}\frac{(P\cdot\tilde y)_n}{\delta_r(n)}T_n(x) = \sum_{N-s\leqslant|n|<N+s}\frac{(P\cdot\tilde y)_n}{\delta_r(n)}T_n^{(r)}(x). \tag{7.14}$$

On observe que la sortie $\tilde y(x)$ de l'algorithme 7.1 satisfait une équation différentielle inhomogène de la forme $L\cdot\tilde y = \tau_{N-s}T_n^{(r)}(x) + \cdots + \tau_{N+s-1}T_n^{(r)}(x)$. (Cependant, le support de la suite v_n n'est en général pas creux .)

Ce point de vue nous amène également à l'observation suivante. À titre de comparaison, on rappelle que la meilleure complexité arithmétique connue pour la conversion d'un polynôme arbitraire de degré d de la base de Tchebychev à la base monomiale (ou inversement) est $\mathcal{O}(\mathsf{M}(n))$, où M représente le coût d'une multiplication polynomiale (voir §3).

Proposition 7.7. *L'expression sur la base monomiale du polynôme $\tilde{y}(x)$ retourné par l'algorithme 7.1 peut-être calculé en $\mathcal{O}(d)$ opérations arithmétiques.*

Démonstration. Comme déjà mentionné, le développement en série de Taylor d'une fonction D-finie satisfait une récurrence linéaire à coefficients polynomiaux. Dans le cas où la fonction est solution d'une équation inhomogène $L \cdot u = v$, l'opérateur de récurrence ne dépend pas de v, et le membre droit de la récurrence est la suite des coefficients de v. Comme $\tilde{y}$ satisfait $L \cdot \tilde{y} = v$ où v est donné par (7.14). les coefficients $(P \cdot \tilde{y})_n/\delta_r(n)$ de (7.14) sont calculés aisément à partir des quelques derniers coefficients de Tchebychev de $\tilde{y}$. On en déduit les coefficients de v_n en temps linéaire en appliquant de manière répétée la relation de récurrence inhomogène

$$T'_{n-1}(x) = -T'_{n+1}(x) + 2xT'_n(x) + 2T_n(x) \tag{7.15}$$

obtenue de l'équation de dérivation (2.9) page 20, et finalement ceux du développement de $\tilde{y}$ dans la base monomiale en utilisant les relations de récurrence qu'ils satisfont. □

Ce résultat de complexité sera utile pour le développement de fraction rationnelle en série de Tchebychev dans le chapitre suivant.

3 Développements de Tchebychev des fractions rationnelles

Cette section est consacrée au même problème que dans le reste du chapitre, seulement restreint au cas où $y(x)$ est une faction rationnelle. On s'intéresse au calcul d'une relation de récurrence vérifiée par les coefficients y_n de la série de Tchebychev de la fonction y, à l'utilisation de cette récurrence pour obtenir une bonne approximation polynomiale uniforme de $y(x)$ on $[-1, 1]$, et à la validation de cette approximation. Tout cela sera utile lors de l'étape de validation de notre algorithme principal.

Notre outil principal est le changement de variable $x = \frac{1}{2}(z+z^{-1})$ suivi d'une décomposition en éléments simples. Des idées similaires ont été utilisées dans le passé avec des objectifs légèrement différents des nôtres, comme le calcul de y_n en forme close [EF89, Mat06]. En effet, la suite $(y_n)_{n\in\mathbb{N}}$ vérifie une récurrence à coefficients *constants*. Calculer cette récurrence ou une forme close de y_n sont des problèmes essentiellement équivalents. Cependant, nous aurons besoin par la suite de quelques résultats sur le coût des algorithmes que nous n'avons pas trouvés dans la littérature. Notre principal souci de ce point de vue est d'éviter les conversions de la base de Tchebychev vers la base monomiale ou inversement, afin d'aboutir à une complexité arithmétique linéaire. Nous avons également besoin d'une borne d'erreur simple du résultat.

L'idée centrale de cette section est la relation entre la série de Tchebychev de $y(x)$ et la série de Laurent de $y\left(\frac{x+x^{-1}}{2}\right)$ (voir §5.3).

Par commodité dans cette section, les séries de Tchebychev seront indexées sur $\mathbb{Z}$, en rappelant que pour toute suite de coefficients de Tchebychev (y_n), nous avons l'égalité

(voir 2.16 page 23) :

$$\sum_{n\in\mathbb{N}}{}'y_nT_n(x) = \sum_{n\in\mathbb{Z}} y_nT_n(x).$$

3.1 Récurrence et expression explicite

Soit $y(x) = a(x)/b(x) \in \mathbb{Q}[x]$ une fraction rationnelle sans pôle sur $[-1,1]$. On note $(y_n)_{n\in\mathbb{Z}}$, $(a_n)_{n\in\mathbb{Z}}$ and $(b_n)_{n\in\mathbb{Z}}$ les suites symétriques des coefficients de Tchebychev de y, a et b.

Proposition 7.8. *La suite des coefficients de Tchebychev $(y_n)_{n\in\mathbb{Z}}$ satisfait la relation de récurrence à coefficients constants $b(\frac{1}{2}(S+S^{-1}))\cdot(y_n) = (a_n)$.*

Démonstration. Ce n'est que le cas limite $r = 0$ du théorème 5.24 page 105, mais une preuve directe est vraiment simple : on écrit

$$\sum_{i=-\infty}^{\infty} b_i z^i \sum_{n=-\infty}^{\infty} y_n z^n = \sum_{n=-\infty}^{\infty}\Big(\sum_{i=-\infty}^{\infty} b_i y_{n-i}\Big)z^n = \sum_{n=-\infty}^{\infty} a_n z^n, \quad x = \frac{z+z^{-1}}{2},$$

et l'on identifie par l'extraction des coefficients de z^i. □

Cette récurrence souffre des mêmes problèmes liés à l'existence de solutions divergentes et à l'accès aux conditions initiales que ceux discutés dans §5.3. Cependant, on peut séparer explicitement les puissances positives de z des puissances négatives dans le développement de Laurent

$$\hat{y}(z) = y\Big(\frac{z+z^{-1}}{2}\Big) = \sum_{n=-\infty}^{\infty} y_n z^n, \qquad \rho^{-1} < |z| < \rho, \tag{7.16}$$

par décomposition en éléments simples. Du point de vue du calcul, mieux vaut prendre comme point de départ la factorisation sans carré du dénominateur de $\tilde{y}$:

$$\beta(z) = z^{\deg b} b\left(\frac{z+z^{-1}}{2}\right) = \beta_1(z)\beta_2(z)^2\cdots\beta_k(z)^k \tag{7.17}$$

et écrire la décomposition en éléments simples sur les complexes de $\hat{y}(z)$ sous la forme

$$\hat{y}(z) = q(z) + \sum_{i=1}^{k}\sum_{\beta_i(\zeta)=0}\sum_{j=1}^{i}\frac{h_{i,j}(\zeta)}{(\zeta-z)^j}, \qquad q(z) = \sum_n q_n z^n \in \mathbb{Q}[z], \quad h_{i,j}\in\mathbb{Q}(z). \tag{7.18}$$

Les $h_{i,j}$ peuvent être calculés efficacement en utilisant l'algorithme de Bronstein-Salvy [BS93] (voir aussi [GS96]).

On obtient une identité de la forme (7.16) en développant en série à l'origine les éléments simples associés aux pôles ζ avec $|\zeta| > 1$, et à l'infini ceux pour lesquels $|\zeta| < 1$. Le développement à l'infini de

$$\frac{h_{i,j}(\zeta)}{(\zeta-z)^j} = \frac{(-1)^j z^{-j} h_{i,j}(\zeta)}{(1-\zeta^{-1}z^{-1})^j}$$

ne contribue pas aux coefficients de z^n, $n \geqslant 0$ dans le développement complet en série de Laurent. Il suit par unicité du développement en série de Laurent de $\hat{y}$ dans l'anneau $\rho^{-1} < |z| < \rho$ que [2]

$$\sum_{n=0}^{\infty} y_n z^n = q(z) + \sum_{i=1}^{k} \sum_{\substack{\beta_i(\zeta)=0 \\ |\zeta|>1}} \sum_{j=1}^{i} \frac{h_{i,j}(\zeta)}{(\zeta - z)^j}. \tag{7.19}$$

Par extraction de coefficient dans (7.19) et utilisation de la symétrie de $(y_n)_{n\in\mathbb{Z}}$ on obtient une forme explicite de y_n en termes des racines $b(\frac{1}{2}(z+z^{-1}))$.

Proposition 7.9. *Les coefficients de Tchebychev $y(x) = \sum_{n=-\infty}^{\infty} y_{|n|} T_n(x)$ sont donnés par la formule*

$$y_n = q_n + \sum_{i=1}^{k} \sum_{j=1}^{i} \sum_{\substack{\beta_i(\zeta)=0 \\ |\zeta|>1}} \binom{n+j-1}{j-1} h_{i,j}(\zeta)\zeta^{-n-j} \qquad (n \geqslant 0) \tag{7.20}$$

où les $q_n \in \mathbb{Q}$, $\beta_i \in \mathbb{Q}[z]$ et $h_{i,j} \in \mathbb{Q}(z)$ sont définis par les équations (7.17) et (7.18).

Observons que l'équation (7.19) fournit implicitement une récurrence d'ordre $\deg b$ pour $(y_n)_{n\in\mathbb{N}}$, au lieu de $2\deg b$ par celle qui résulte de la proposition 7.8, mais la nouvelle récurrence est à coefficients algébriques et non rationnels en général. Par ailleurs, nous pouvons maintenant borner l'erreur commise en tronquant la série de Tchebychev de y.

Proposition 7.10. *Soit $y \in \mathbb{Q}(x)$ une fraction rationnelle sans pôle dans le disque elliptique E_ρ (voir (2.19) page 23). En reprenant toutes les notations des équations (7.17) et (7.18), pour tout $d \geqslant \deg q$, on a*

$$\| \sum_{n>d} y_n T_n \|_\infty \leqslant \sum_{i=1}^{k} \sum_{j=1}^{i} \sum_{\substack{\beta_i(\zeta)=0 \\ |\zeta|>1}} \frac{|h_{i,j}(\zeta)|(d+2)^{j-1}}{(|\zeta|-1)^j} |\zeta|^{-d-1} = \mathcal{O}(d^{\deg b}\rho^{-d}).$$

Démonstration. On a $\| \sum_{n>d} y_n T_n \|_\infty \leqslant \sum_{n>d} |y_n|$ car $\|T_n\|_\infty \leqslant 1$ pour tout n. En utilisant l'inégalité

$$\sum_{n>d} \binom{n+j-1}{j-1} t^{n+j} \leqslant (d+2)^{j-1} t^{d+1} \sum_{n=0}^{\infty} \binom{n+j-1}{j-1} t^{n+j} = \frac{(d+2)^{j-1} t^{d+j+1}}{(1-t)^j}$$

2. Pour prévenir toute confusion, il vaut peut-être la peine de relever que dans l'expression

$$\tilde{y}(z) = q(z) + \sum_{i=1}^{k} \sum_{\substack{\beta_i(\zeta)=0 \\ |\zeta|>1}} \sum_{j=1}^{i} \left(\frac{h_{i,j}(\zeta)}{(\zeta - z)^j} + \frac{h_{i,j}(\zeta^{-1})}{(\zeta^{-1} - z)^j} \right)$$

la série de Laurent d'un terme isolé de la forme $\frac{h_{i,j}(\zeta)}{(\zeta-z)^j} + \frac{h_{i,j}(\zeta^{-1})}{(\zeta^{-1}-z)^j}$ n'est *pas* symétrique pour $j > 1$, même si $q(z) = 0$.

pour $t < 1$, l'expression explicite de la proposition 7.9 établit la borne

$$\begin{aligned}\sum_{n>d}|y_n| &\leqslant \sum_{n>d}\sum_{i=1}^{k}\sum_{j=1}^{i}\sum_{\substack{\beta_i(\zeta)=0\\|\zeta|>1}}\binom{n+j-1}{j-1}|h_{i,j}(\zeta)||\zeta|^{-n-j}\\ &\leqslant \sum_{i=1}^{k}\sum_{j=1}^{i}\sum_{\substack{\beta_i(\zeta)=0\\|\zeta|>1}}\frac{|h_{i,j}(\zeta)|(d+2)^{j-1}}{(|\zeta|-1)^j}|\zeta|^{-d-1}.\end{aligned}$$

L'estimation asymptotique provient du fait que $|\zeta| > 1$ entraîne $|\zeta| > \rho$ lorsque $b(\frac{1}{2}(\zeta+\zeta^{-1})) = 0$. □

3.2 Calcul

Reste à s'assurer que les résultats précédents induisent effectivement un algorithme de complexité arithmétique linéaire. Nous rappelons d'abord un résultat sur la division euclidienne entre deux polynômes exprimés dans la base de Tchebychev. L'objet de la section 4.2 est la présentation d'un algorithme rapide pour effectuer cette opération (3.15 page 51). Cet algorithme n'est en fait pas utile pour le propos qui nous préoccupe ici à savoir la division d'un polynôme a par un polynôme de degré constant b. L'algorithme naïf nous suffit pour obtenir une complexité optimale. Le lemme suivant découle directement de l'algorithme 3.14 page 49.

Lemme 7.11. *La division avec reste $a = bq + r$ $(\deg r < \deg b)$ où $a, b, q, r \in \mathbb{Q}[x]$ sont représentés dans la base monomiale peut être calculé en $\mathcal{O}(\deg a)$ opérations arithmétiques à b fixé.*

On peut à ce stade énoncer l'algorithme 7.2 qui prend en entrée à la fois une fraction rationnelle dont les degrés du numérateur et du dénominateur sont supposés bornés, et un polynôme de degré de l'ordre de celui attendu en sortie, mais déjà écrit dans la base de Tchebychev.

Proposition 7.12. *L'algorithme 7.2 est correct. Il s'exécute en $\mathcal{O}(d + \log(\epsilon^{-1}))$, tous les autres paramètres étant fixés.*

Démonstration. On montre dans un premier temps la borne d'erreur $\|\tilde{y} - y\|_\infty \leqslant \epsilon$ puis dans un second l'analyse de complexité de l'algorithme.

Validité de la borne d'erreur. Soit $A = \{\zeta : \rho_- \leqslant |\zeta| \leqslant \rho_+\}$ et

$$M_0 = \sup_{\zeta\in A}|h'_{i,j}(\zeta)|,\ M_1 = \sup_{\zeta\in A}|\zeta^{-1}h_{i,j}(\zeta)|.$$

Pour tout $\zeta \in A$,

$$\left|\left(h_{i,j}(\zeta)\zeta^{-n-j}\right)'\right| \leqslant (M_0 + (n+j)M_1)\,|\zeta|^{-n-j} \leqslant (n+j)(M_0+M_1)\rho_-^{-n-j}. \tag{7.21}$$

Entrée: Les coefficients de Tchebychev d'un polynôme $f = \sum_{k=-d}^{d} f_k T_k(x)$ de degré d, une fraction rationnelle $a/b \in \mathbb{Q}(x)$. Une précision cible ϵ.

Sortie: Les coefficients de Tchebychev d'un polynôme $\tilde{y}(x)$ tels que $\left\| \tilde{y} - \frac{fa}{b} \right\|_\infty < \epsilon$.

1: Développer a et b dans la base des polynômes de Tchebychev.
2: Calculer le produit de polynômes af dans la base de Tchebychev
3: Calculer le quotient q et le reste r de la division de af par b
4: Calculer la décomposition en éléments simples de $\hat{w}(z) = w(x) = r(x)/b(x)$ où $x = \frac{z+z^{-1}}{2}$, en utilisant l'algorithme de Bronstein-Salvy, et en déduire celle de $\hat{y}(z) = q(x) + w(x)$ (cf. (7.18))
5: Déterminer $d' \geqslant \deg q$ tel que $\left\| \sum_{k>d'} y_k T_k \right\|_\infty < \frac{\epsilon}{4}$
6: Calculer ρ_- et ρ_+ tels que $\beta(\zeta) = 0 \wedge |\zeta| > 1 \Rightarrow 1 < \rho_- \leqslant |\zeta| \leqslant \rho_+$ (β est défini par (7.17))
7: Calculer $M \geqslant \sum_{i=1}^{k} \sum_{j=1}^{i} j(\deg \beta_i) \sup_{\rho_- \leqslant |\zeta| \leqslant \rho_+} \left(|h'_{i,j}(\zeta)| + |\zeta^{-1} h_{i,j}(\zeta)| \right) {\rho_-}^{-j}$ (les β_i sont définis par (7.17) et les $h_{i,j}$ sont définis par (7.18))
8: $\epsilon' := \min\left(\rho_- - 1, M^{-1} \left(1 - \rho_-^{-1}\right)^{D+1} \frac{\epsilon}{4} \right)$, avec $D = \deg b$
9: Calculer les approximations $\tilde{\zeta} \in \mathbb{Q}[i]$ des racines ζ de β_i tels que $\left|\tilde{\zeta} - \zeta\right| < \epsilon'$;
10: **pour** $0 \leqslant n \leqslant d'$ **faire**
11: $\quad \tilde{y}_n = q_n + \mathrm{Re}\left(\sum_{i=1}^{k} \sum_{j=1}^{i} \sum_{\substack{\beta_i(\zeta) = 0 \\ |\zeta| > 1}} \binom{n+j-1}{j-1} h_{i,j}(\tilde{\zeta}) \tilde{\zeta}^{-n-j} \right)$
12: **fin pour**
13: **renvoyer** $\tilde{y}(x) = \sum_{n=-d'}^{d'} \tilde{y}_n T_n(x)$

Algorithme 7.2: Calcul des coefficients de Tchebychev d'une fraction rationnelle

Par la proposition 7.9, la condition $|\zeta - \tilde{\zeta}| < \rho_- - 1$ issue de l'étape 8 implique que $\left[\zeta, \tilde{\zeta}\right] \subset A$, et en utilisant (7.21), on a :

$$
\begin{aligned}
|y_n - \tilde{y}_n| &\leqslant \sum_{i=1}^{k} \sum_{j=1}^{i} \sum_{\substack{\beta_i(\zeta) = 0 \\ |\zeta| > 1}} \binom{n+j-1}{j-1} \left| h_{i,j}(\zeta) \zeta^{-n-j} - h_{i,j}(\tilde{\zeta}) \tilde{\zeta}^{-n-j} \right| \\
&\leqslant \sum_{i=1}^{k} \sum_{j=1}^{i} \sum_{\substack{\beta_i(\zeta) = 0 \\ |\zeta| > 1}} \binom{n+j-1}{j-1} \left(M_0 + (n+j) M_1 \right) |\zeta|^{-n-j} \left| \tilde{\zeta} - \zeta \right| \\
&\leqslant \sum_{i=1}^{k} \sum_{j=1}^{i} j(\deg \beta_i) \binom{n+j}{j} \left(M_0 + M_1 \right) {\rho_-}^{-n-j} \epsilon' \\
&\leqslant M \binom{n+D}{D} {\rho_-}^{-n} \epsilon'.
\end{aligned}
$$

Ainsi,

$$\|y_n - \tilde{y}_n\|_\infty \leqslant \sum_{n=-d'}^{d'} |y_n - \tilde{y}_n| + 2\left\|\sum_{n>d'} y_n T_n\right\|_\infty \leqslant \frac{2M\epsilon'}{(1-\rho_-^{-1})^{D+1}} + 2\frac{\epsilon}{4} \leqslant \epsilon.$$

Analyse de complexité. Il est facile de voir que les étapes 1 et 4 à 8 ainsi que chaque itération de la boucle finale en faisant attention de retenir les valeurs de $\tilde{\zeta}^{-n}$ d'une itération sur l'autre ont un coût constant (en le degré n et ϵ). Les étapes 2 et 3 prennent un temps linéaire en n, en utilisant les résultats de la section 4 et du lemme 7.11 et ne dépendent pas de ϵ. Enfin, pour l'étape 9, il est connu [Pan96, Théorème 1.1(d)] que les racines d'un polynôme avec des coefficients entiers peuvent être approchées avec une précision absolue η en $\mathcal{O}(\eta^{-1})$ opérations arithmétiques. On note au passage que la complexité binaire de l'algorithme est elle-même quasiment linéaire en ϵ^{-1}. Les dépendances en le degré du polynôme de ces deux complexités sont polynomiales de petit degré. Comme M ne dépend ni de ϵ ni de d, on a $\epsilon' = \Omega(\epsilon)$ et ainsi l'étape 9 est en $\mathcal{O}(\epsilon^{-1})$ opérations. □

4 Bornes d'erreurs / Validation

Dans cette section, on suppose déjà calculé (voir algorithme 7.1) un polynôme $p(x) = \sum_{n=0}^{d}{}' \tilde{y}_n T_n(x)$ de degré d, présumé fournir une approximation sur $[-1,1]$ d'une fonction D-finie y, spécifiée par une équation différentielle et des conditions au bord. Nous nous limitons au cas où l'équation est accompagnée de conditions initiales à l'origine. Comme énoncé par le problème 7.1 de l'introduction, nous sommes maintenant intéressés par le calcul d'une borne B « fine » telle que $|y(x) - p(x)| \leqslant B$ pour tout $x \in [-1,1]$.

L'idée principale pour le calcul de cette borne est d'utiliser l'arithmétique des intervalles pour des équations différentielles ordinaires (voir les rapports donnés dans [NJC99, Cor94] par exemple). On note qu'en général, ces méthodes apparaissent habituellement dans le cadre de calculs par intervalles sur des développements de Taylor en une ou plusieurs variables, mais ici nous présentons une adaptation pour valider les solutions d'équations différentielles dans la base de Tchebychev (Kaucher, Miranker et d'autres [EMR82b, KM84, KM88] utilisaient déjà ces méthodes pour les séries de Tchebychev). Par souci de simplicité, nous présentons d'abord notre méthode pour les équations différentielles d'ordre 1, ensuite nous montrons que nous pouvons utiliser cette méthode pour donner un algorithme plus général pour des équations différentielles d'ordres quelconques.

4.1 Equation différentielle d'ordre 1

Avant d'énoncer l'algorithme de validation pour des équations différentielles d'ordres quelconques, voyons à quoi il ressemble dans le cas d'une équation d'ordre 1. Considérons l'équation :

$$y'(x) = a(x)y(x). \qquad y(0) = y_0,$$

Entrée: Une équation différentielle L d'ordre 1 (écrite sous la forme $L \cdot y = y' - a \cdot y$, avec $a \in \mathbb{Q}(x)$), une condition initiale $y(t_0) = y_0$, une approximation polynomiale $p(x) = \sum_{n=0}^{d} {}'\tilde{y}_n T_n(x)$ de y, un paramètre de précision $\epsilon > 0$
Sortie: : Une borne d'erreur B telle que $\|y - p\|_\infty \leqslant B$

1: $P_0 := p$
2: Calculer j tel que $\gamma_j(t) := \|a\|_\infty^j \cdot \frac{|t-t_0|^j}{j!} < 1$;
3: **pour** $i = 1 \dots j$ **faire**
4: Utiliser l'algorithme 7.2 pour calculer une approximation polynomiale de $P_{a,i}$ pour $a \cdot P_{i-1}$ tel que $\|P_{a,i} - a \cdot P_{i-1}\|_\infty \leqslant \epsilon$
5: Calculer $P_i(t) := y_0 + \int_{t_0}^{t} P_{a,i}(x)\mathrm{d}x$;
6: **fin pour**
7: Calculer $\alpha_j(t) := \epsilon \cdot \sum_{k=1}^{j} \|a\|_\infty^{k-1} \frac{|t-t_0|^k}{k!}$, $\beta_j(t) := P_j(t) - P_0(t)$;
8: **renvoyer** $R_j^\star = \frac{\|\alpha_j\|_\infty + \|\beta_j\|_\infty}{1 - \|\gamma_j\|_\infty}$

Algorithme 7.3: Algorithme de validation pour les équations d'ordre 1

où a est une fraction rationnelle sans pôles sur $[-1, 1]$. En introduisant l'opérateur

$$\tau : y \mapsto \left(x \mapsto y_0 + \int_0^x a(t)y(t)\mathrm{d}t \right), \tag{7.22}$$

le problème 7.1 se met sous la forme de point fixe $\tau(y) = y$. Soit y l'unique solution.

Supposons un instant disposer, en plus du polynôme d'approximation p, d'une borne d'erreur candidate R. Comment vérifier par le calcul que $\|p - y\|_\infty \leqslant R$? Plaçons-nous dans l'espace de Banach des fonctions analytiques sur un ouvert $\Omega \subset \mathbb{C}$ contenant $[-1, 1]$, muni de la norme de convergence uniforme, et notons

$$\mathcal{B}(g, R) = \{f : \|f - g\|_\infty \leqslant R\}$$

la boule fermée centrée en g et de rayon R. C'est un sous-espace métrique complet. On vérifie de plus que

$$\left\|\tau^j(f) - \tau^j(g)\right\|_\infty \leqslant \gamma_j \|f - g\|_\infty, \qquad \text{où } \gamma_j = \frac{\|a\|_\infty^j}{j!}.$$

Comme $\gamma_j \to 0$ quand $j \to \infty$, si la boule $\mathcal{B}(p, R)$ est stable par τ^i pour un certain i, la restriction de τ^i à $\mathcal{B}(p, R)$ possède un itéré contractant, et admet donc un unique point fixe. La solution y appartient alors à $\mathcal{B}(p, R)$.

Concrètement, on explicite le calcul de la borne R par cette idée avec l'algorithme 7.3.

Proposition 7.13. *Étant donnés une équation différentielle linéaire à coefficients polynomiaux d'ordre* 1*, une condition initiale, un polynôme d'approximation* p *de degré* d *de l'unique solution* y *de l'équation différentielle et une précision cible* $\epsilon > 0$*, l'algorithme 7.3 calcule une borne majorante* B *de* $\|y-p\|_\infty$ *en* $\mathcal{O}(d+\log_2(\epsilon^{-1}))$ *opérations arithmétiques.*

On prouve d'abord la correction de l'algorithme. L'idée de cet algorithme est d'itérer (en partant d'une approximation initiale p) l'opérateur de point fixe (7.22) à notre approximation dans le but de construire une suite convergente de borne supérieure de $\|y-p\|_\infty$, dont la limite est la borne B retournée.

Lemme 7.14. *En utilisant les notations de l'algorithme 7.3, pour toute borne majorante* $R > 0$*,* $\|y-p\|_\infty \leqslant R$*,* $S_{k,i}(R)$ *définie, pour tout entier* i*, par*

$$\begin{aligned} S_{0,i}(R) &= \|\alpha_i\|_\infty + \|\beta_i\|_\infty + \|\gamma_i\|_\infty R, \\ S_{k,i}(R) &= \|\alpha_i\|_\infty + \|\beta_i\|_\infty + \|\gamma_i\|_\infty S_{k-1,i}(R), \ k \in \mathbb{N}^*, \end{aligned} \tag{7.23}$$

est aussi une borne supérieure de $\|y-p\|_\infty$*.*

Démonstration. Soit p un polynôme et R une borne supérieure de $\|y-p\|_\infty$. En utilisant (7.22) on construit des approximations successives $P_i(t)$ comme décrit aux étapes 4 et 8 de l'algorithme7.3 avec l'initialisation $P_0 = p$. On calcule aussi par simple majoration d'intégrale une borne de $|y(t) - P_i(t)|$:

$$R_i(t) := \epsilon \cdot \sum_{k=1}^{i} \|a\|_\infty^{k-1} \frac{|t-t_0|^k}{k!} + \|a\|_\infty^i \cdot R \cdot \frac{|t-t_0|^i}{i!}, \quad t \in [-1,1], \quad i \geqslant 1. \tag{7.24}$$

On vérifie la validité de cette borne par une récurrence et l'inégalité triangulaire suivante :

$$|y(t) - p(t)| \leqslant |y(t) - P_i(t)| + |P_i(t) - p(t)|.$$

Il s'ensuit que

$$\|y-p\|_\infty \leqslant \|\alpha_i\|_\infty + \|\beta_i\|_\infty + \|\gamma_i\|_\infty \cdot R.$$

Par une récurrence sur k, on déduit le lemme. □

Lemme 7.15. *Pour tout entier* j *tel que* $\|\gamma_j(t)\|_\infty < 1$*, on a la limite suivante :*

$$\lim_{k\to\infty} S_{k,j}(R) = \frac{\|\alpha_j\|_\infty + \|\beta_j\|_\infty}{1 - \|\gamma_j\|_\infty}.$$

Démonstration. En utilisant le lemme 7.14, une récurrence sur k et l'équation (7.23), on prouve l'égalité :

$$S_{k,j}(R) = \sum_{i=0}^{k-1} ((\|\alpha_j\|_\infty + \|\beta_j\|_\infty) \cdot \|\gamma_j\|_\infty^i) + \|\gamma_j\|_\infty^k R.$$

Par convergence des séries géométriques et l'inégalité $\|\gamma_j(t)\|_\infty < 1$, on déduit :

$$\lim_{k\to\infty} S_{k,j}(R) = \frac{\|\alpha_j\|_\infty + \|\beta_j\|_\infty}{1 - \|\gamma_j\|_\infty}.$$

□

Démonstration de la proposition 7.13. L'algorithme 7.3 calcule un entier j tel que $\|\gamma_j\|_\infty < 1$, par les lemmes précédents, il s'ensuit que $\|y - p\|_\infty \leqslant R_j^*$ et ainsi la valeur retournée par 7.3 est correcte.

Les dépendances de cet algorithme en d et ϵ sont données par l'appel de l'algorithme 7.2.

Par l'analyse de complexité de cet algorithme, on effectue $\mathcal{O}(d + \log(\epsilon))$ opérations arithmétiques. Le nombre d'appels de cet algorithme dépend seulement des coefficients polynomiaux de l'équation différentielle. La complexité de cet algorithme est bien $\mathcal{O}(d + \log(\epsilon))$ opérations arithmétiques. □

4.2 Finesse de la borne calculée

On cherche à comparer la borne d'erreur calculée avec la vraie erreur de notre approximation calculée, c'est-à-dire $\|y - p\|_\infty$. Le lemme 7.16 nous donne une idée de la différence entre les deux erreurs.

Lemme 7.16. *Soient* $\varepsilon^* = \|y - p\|_\infty$, $B = \frac{\|\alpha_j\|_\infty + \|\beta_j\|_\infty}{1 - \|\gamma_j\|_\infty}$. *On a :*

$$\varepsilon^* \leqslant B \leqslant \frac{2\,\|\alpha_j\|_\infty + (1 + \|\gamma_j\|_\infty)\varepsilon^*}{1 - \|\gamma_j\|_\infty}.$$

Démonstration. On a $\|\beta_j\|_\infty = \|P_j - p\|_\infty \leqslant \|y - P_j\|_\infty + \|y - p\|_\infty$. Par (7.24) avec $R = \varepsilon^*$. On a donc : $\|\beta_j\|_\infty \leqslant \|\alpha_j\|_\infty + (1 + \|\gamma_j\|_\infty)\varepsilon^*$.

□

Remarque 7.17. Comme $\|\alpha_j\|_\infty$ peut être aussi petit que désiré, on suppose que $\|\alpha_j\|_\infty \leqslant \mu_\epsilon \varepsilon^*$. On peut calculer la norme uniforme avec une erreur relative aussi petite que l'on souhaite :

$$\left|\frac{\varepsilon^* - B}{\varepsilon^*}\right| \leqslant \frac{2(\mu_M + \|\gamma_j\|_\infty)}{1 - \|\gamma_j\|_\infty}.$$

4.3 Équation différentielle d'ordre quelconque

L'algorithme 7.3 se généralise à une équation d'ordre quelconque. Avant d'énoncer cet algorithme on énonce deux lemmes qui montrent que l'on peut effectuer des opérations de l'algorithme 7.4 en bonne complexité.

Lemme 7.18. *On peut calculer la dérivée ou une primitive d'un polynôme de degré au plus d écrit sur la base de Tchebychev en $\mathcal{O}(d)$ opérations arithmétiques.*

Démonstration. Si $f = \sum_n{}' c_n T_n(x)$ et $f' = \sum_n{}' c'_n T_n(x)$, on a $2nc_n = c'_{n-1} + c'_{n+1}$ d'après la relation (2.9) page 20. Les primitives se calculent par application directe de cette formule. Pour dériver f, supposé de degré d, on part du fait que les coefficients c'_n du polynôme dérivé sont nuls lorsque $|n| \geqslant d$, et on calcule les coefficients restants, pour $|n|$ décroissant, en utilisant la formule d'intégration vue comme une relation de récurrence inhomogène. □

Lemme 7.19. *Soit $f = \sum_{n=0}^{d}{}' c_n T_n$ un polynôme de degré d donné sur la base de Tchebychev. On peut calculer $M \geqslant 0$ tel que $\|f\|_\infty \leqslant M \leqslant \sqrt{d}\,\|f\|_\infty$ en $\mathcal{O}(d)$ opérations.*

Démonstration. On dispose de l'encadrement

$$\|f\|_2 \leqslant \|f\|_\infty \leqslant \sum_{n=0}^{d}{}' |c_n| \leqslant \sqrt{d}\|f\|_2$$

où

$$\|f\|_2 = \left(\frac{2}{\pi}\int_{-1}^{1} \frac{f(t)^2}{\sqrt{1-t^2}}dt\right)^{1/2} = \left(|c_0|^2 + 4\sum_{n=1}^{d}|c_n|^2\right)^{1/2}$$

donc la somme des modules de c_n fournit la borne cherchée. □

Proposition 7.20. *Soient L une équation différentielle d'ordre r (écrite sous la forme $L\cdot y = y^{(r)} - \left(a_{r-1}y^{(r-1)} + \cdots + a_0 y\right)$, avec $a_i \in \mathbb{Q}(x)$) et une suite de conditions initiales $y^{(i)}(t_0) = y_0^{(i)}$, y la solution unique de $L\cdot y = 0$ avec ces conditions initiales, $p(x) = \sum_{n=0}^{d}\tilde{y}_n T_n(x)$ une approximation polynomiale de y de degré d et un paramètre de précision ϵ. L'algorithme 7.4 calcule une borne B telle que $\|y-p\|_\infty \leqslant B$ en $\mathcal{O}(d + \log(\epsilon^{-1}))$ opérations arithmétiques.*

Démonstration. On prouve cette proposition en deux parties. D'abord la correction de l'algorithme puis l'analyse de complexité.

Correction de l'algorithme. La généralisation de l'algorithme 7.3 est issue de la généralisation de l'application de (7.22), pour valider une approximation de $y^{(r-1)}$:

$$\tau(u^{(r-1)})(t) = y_0^{(n-1)} + \int_{t_0}^{t}\left(a_{r-1}(s)u^{(r-1)}(s) + \cdots + a_0(s)u(s)\right)\mathrm{d}s, \tag{7.25}$$

où $a_i \in \mathbb{Q}(x)$.

Comme on s'est donné une approximation polynomiale p de y, on peut calculer numériquement ses $r-1$ dérivées $P_{r-1,0} = p^{(r-1)}$. On calcule d'abord lors des étapes 5 à 13 des bornes d'erreurs de $R_j^\star \geqslant \left\|y^{(r-1)} - P_{r-1,0}\right\|_\infty$ (ceci est discuté par la suite). À partir de là, on peut déduire, par $r-1$ intégrations successives de $P_{r-1,0}$, le polynôme $P_{0,0}$ et valider la borne d'erreur $\|P_{0,0} - y\|_\infty \leqslant R_j^\star \cdot \frac{|t-t_0|^{r-1}}{(r-1)!}$.

Ainsi, on peut borner l'erreur $\|y-p\|_\infty \leqslant \|y - P_{0,0}\|_\infty + \|p - P_{0,0}\|_\infty$.

Entrée: : Une équation différentielle L d'ordre 1 (écrite sous la forme $L \cdot y = y^{(r)} - \left(a_{r-1}y^{(r-1)} + \cdots + a_0 y\right)$, avec $a_i \in \mathbb{Q}(x)$), une suite de conditions initiales $y^{(i)}(t_0) = y_0^{(i)}$, une approximation polynomiale $p(x) = \sum_{n=0}^{d}{}' \tilde{y}_n T_n(x)$ de y, un paramètre de précision $\epsilon > 0$.

Sortie: : Une borne d'erreur B telle que $\|y - p\|_\infty \leqslant B$, où y désigne la solution de $L \cdot y = 0$ satisfaisant les conditions initiales données.

1: $P_{r-1,0} := p^{(r-1)}$; $A := \max\{\|a_i\|_\infty, i = r-1 \dots 0\}$; $Q := \left\| \sum_{i=0}^{r-1} \frac{|t-t_0|^i}{i!} \right\|_\infty$

2: Calculer j tel que $\gamma_j(t) := A^j \cdot Q^j \cdot \frac{|t-t_0|^j}{j!} < 1$.

3: **pour** $i = 0 \dots j-1$ **faire**

4: **pour** $k = r-2 \dots 0$ **faire**

5: Calculer $P_{k,i}(t) := y^{(k)}(t_0) + \int_{t_0}^{t} P_{k+1,i}(u)\mathrm{d}u$

6: **fin pour**

7: **pour** $k = r-1 \dots 0$ **faire**

8: Avec l'algorithme 7.2 calculer les approximations polynomiales $P_{a_k,i}$ pour $a_k \cdot P_{k,i}$ telles que $\|P_{a_k,i} - a_k \cdot P_{a_k,i}\|_\infty \leqslant \epsilon$

9: **fin pour**

10: Calculer $P_{r-1,i+1}(t) := y^{(r-1)}(t_0) + \int_{t_0}^{t} \sum_{k=0}^{r-1} P_{a_k,i}(u)\mathrm{d}u$

11: **fin pour**

12: Calculer $\alpha_j(t) := r\epsilon \cdot \sum_{k=1}^{j} A^{k-1}Q^{k-1}\frac{|t-t_0|^k}{k!}$, $\beta_j(t) := P_{r-1,j}(t) - P_{r-1,0}(t)$

13: Calculer $R_j^* = \frac{\|\alpha_j\|_\infty + \|\beta_j\|_\infty}{1 - \|\gamma_j\|_\infty}$ (R_j^* est déjà une borne d'erreur de $\left\|y^{(r-1)} - P_{r-1,0}\right\|_\infty$)

14: **renvoyer** $B = R_j^* \cdot \frac{|t-t_0|^{r-1}}{(r-1)!} + \|P_{0,0} - p\|_\infty$

Algorithme 7.4: Algorithme de validation pour les équations d'ordre quelconque

Maintenant, examinons en détail les étapes 1 à 13. La preuve de la correction est similaire à celle de l'algorithme 7.3. Notons que lorsque l'ordre de l'équation est plus grand que 1, on ne travaille pas seulement avec une approximation polynomiale de y, mais aussi avec des approximations polynomiales des dérivées $y', \dots, y^{(r-1)}$. Le seul changement significatif est que dans ce cas, (7.24) devient :

$$R_{r-1,j}(t) := rM \cdot \sum_{k=1}^{j} \left(A^{k-1}Q^{k-1} \cdot \frac{|t-t_0|^k}{k!} \right) + A^j \cdot Q^j \cdot R_{r-1} \cdot \frac{|t-t_0|^j}{j!}, \quad j \geqslant 1 \tag{7.26}$$

où A, Q, M sont définis dans l'algorithme 7.4, et $R_{r-1} > 0$ est une borne d'erreur entre $y^{(r-1)}$ et $p^{(r-1)}$.

Analyse de complexité. L'analyse de complexité de cet algorithme est similaire à celle de l'algorithme 7.3. Le nombre d'itérations des boucles ne dépend ni de ϵ ni de d. Chaque

Exemple	degré. d	borne R	$\|y-p\|_\infty$	$\|y-p^*\|_\infty$	$\log_{10} R/\|y-p\|_\infty$
	30	$2,0\cdot10^{-47}$	$3,4\cdot10^{-52}$	$3,4\cdot10^{-52}$	4,8
7.21	60	$7,3\cdot10^{-97}$	$1,9\cdot10^{-97}$	$1,9\cdot10^{-97}$	0,58
	90	$4,4\cdot10^{-142}$	$1,2\cdot10^{-142}$	$1,1\cdot10^{-142}$	0,57
	30	$7,8\cdot10^{-41}$	$5,9\cdot10^{-44}$	$5,6\cdot10^{-44}$	3,1
7.22	60	$4,5\cdot10^{-99}$	$8,7\cdot10^{-103}$	$8,5\cdot10^{-103}$	3,7
	90	$3,5\cdot10^{-164}$	$3,0\cdot10^{-168}$	$3,0\cdot10^{-168}$	4,1
	30	$1,5\cdot10^{-7}$	$4,1\cdot10^{-8}$	$2,8\cdot10^{-8}$	0,57
7.23	60	$2,6\cdot10^{-15}$	$7,3\cdot10^{-16}$	$4,9\cdot10^{-16}$	0,56
	90	$3,2\cdot10^{-23}$	$9,0\cdot10^{-24}$	$6,0\cdot10^{-24}$	0,56

FIGURE 7.1 – Qualité des bornes validées par les algorithmes 7.3 et 7.4. La colonne $\|y-p\|_\infty$ indique la véritable erreur maximale entre l'approximant p et la fonction y (celle-là même qu'on peut lire sur les graphes de la figure 7.2). La colonne $\|y-p^*\|_\infty$ donne une valeur approchée de l'erreur commise par le meilleur approximant de degré d de y, calculé grâce à Sollya [CJL10].

ligne demande $\mathcal{O}(d+\log(\epsilon^{-1}))$ opérations. En effet, les seuls calculs non triviaux sont l'appel à l'algorithme 7.2 qui s'effectue en $\mathcal{O}(d+\log(\epsilon^{-1}))$ opérations arithmétiques selon la proposition 7.12 page 156, le calcul des dérivées et des primitives d'un polynôme de Tchebychev qui s'effectue en la bonne complexité par le lemme 7.18 et le calcul d'une borne fine de $\|\beta_j\|_\infty$ à l'étape 13, ce qui ne pose pas de problème d'après le lemme 7.19.

□

5 Expériences

Nous avons implémenté en Maple une variante des algorithmes de ce chapitre qui mélange arithmétique rationnelle et arithmétique d'intervalles. Les calculs par intervalles reposent sur le module intpakX [Kra06]. Notre implémentation, à l'état de prototype, n'est pas encore disponible publiquement.

Nous nous limitons ici à illustrer sur quelques exemples simples la qualité des approximations et des bornes d'erreur que fournissent les méthodes de ce chapitre.

Pour les trois exemples suivants, nous traçons dans chaque graphe l'erreur entre la solution exacte et le polynôme calculé en utilisant l'algorithme 7.1 pour des degrés $d \in \{30, 60, 90\}$. Dans le tableau 7.1 on donne les temps et les bornes d'erreurs calculées par les algorithmes 7.3 et 7.4.

Exemple 7.21. Cet exemple est adapté de Kaucher et Miranker [KM84]. Il s'agit de la fonction

$$y(x) = \frac{\exp(x/2)}{\sqrt{x+16}}$$

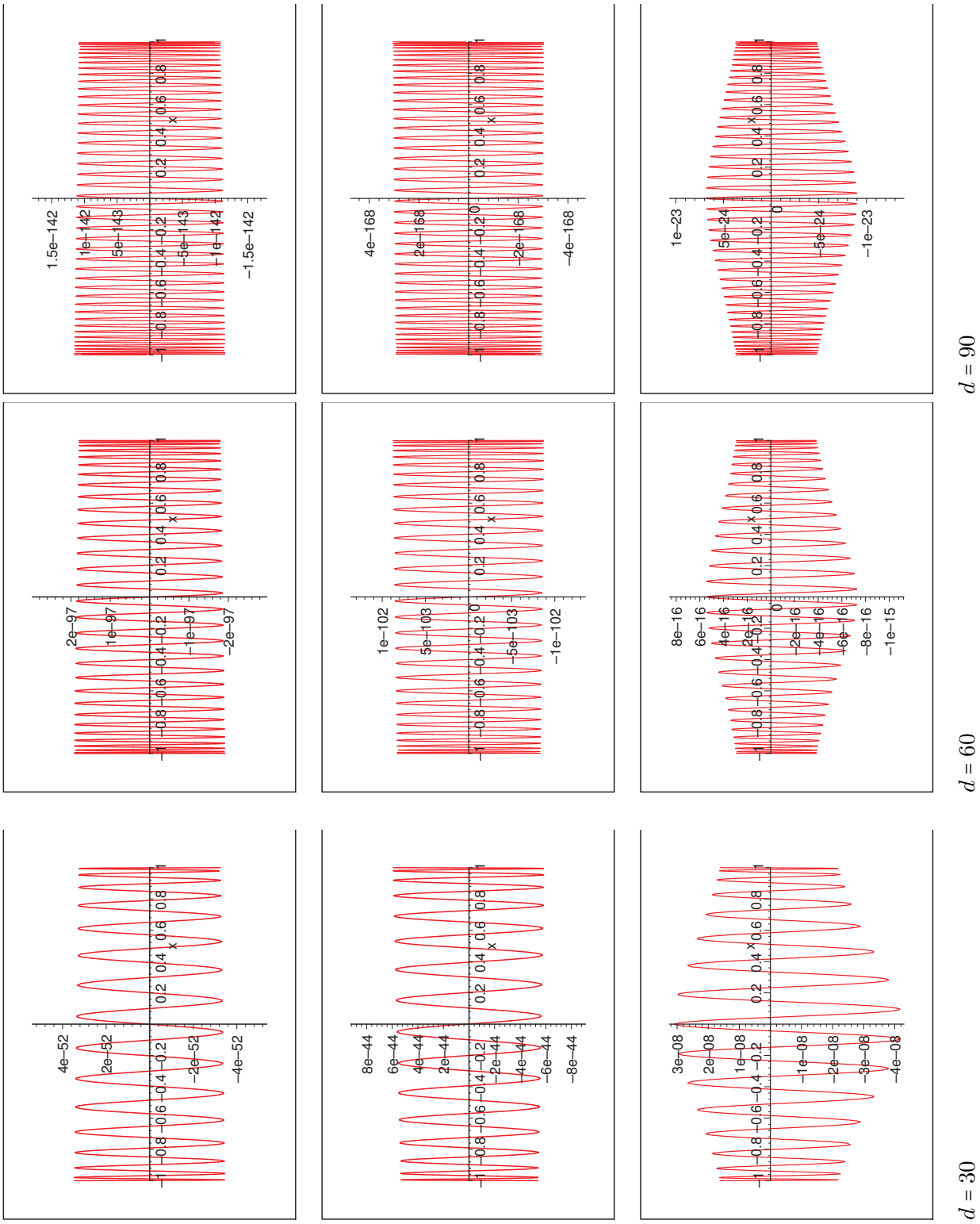


FIGURE 7.2 – Graphes des différences $y - p$ entre approximations p de degré $d \in \{30, 60, 90\}$ calculés par l'algorithme 7.1. Les oscillations d'amplitude presque uniforme sont caractéristiques des approximations polynomiales proches de l'optimum, en vertu du théorème d'oscillation de la Vallée Poussin [Che98]

solution de l'équation différentielle

$$(-x-15)y(x)+(32+2x)y'(x)=0, \quad y(0)=1/4 \tag{7.27}$$

Exemple 7.22. L'exemple de cette équation d'ordre 4 vient de [Ged77a].

$$y^{(4)}(x)-y(x)=0, \quad y(0)=3/2,\, y'(0)=-1/2, y''(0)=-3/2,\, y'''(0)=1/2, \tag{7.28}$$

qui a comme solution exacte

$$y(x)=3/2\cos(x)-1/2\sin(x). \tag{7.29}$$

Exemple 7.23. Ce dernier exemple est une équation d'ordre 1 qui possède des points singuliers imaginaires $x=\pm i\sqrt{2}/2$ relativement proches du segment $[-1,1]$. On peut remarquer qu'avec ce genre de singularité, une approximation sur le segment $[-1,1]$ par une série de Taylor en 0 tronquée n'est pas possible.

$$4xy(x)+(1+4x^2+4x^4)y'(x)=0, \quad y(0)=\exp(1), \tag{7.30}$$

avec la solution exacte

$$y(x)=\exp(1/(1+2x^2)). \tag{7.31}$$

Notons aussi que la condition initiale n'est pas représentable exactement en arithmétique rationnelle, ce qui rend indispensable l'utilisation d'un minimum d'arithmétique des intervalles.

On constate que $\|p-y\|_\infty$ est extrêmement proche de l'erreur optimale accessible avec un polynôme de même degré. La borne prouvée R est à peine pessimiste, surtout quand l'ordre augmente.

Chapitre 8

Séries de Fourier généralisées solutions d'équations différentielles

Résumé

Les *séries de Fourier généralisées* sont des développements en série dans certaines bases de fonctions comme les polynômes orthogonaux classiques, les fonctions de Bessel ou encore dans la base monomiale. Lorsqu'elles sont solutions d'équations différentielles linéaires et sous certaines conditions, les coefficients de ces séries vérifient une récurrence linéaire à coefficients polynomiaux.

Ce chapitre donne une condition suffisante pour que les coefficients vérifient une telle récurrence. Dans le cas où la condition est vérifiée, un algorithme pour calculer cette récurrence est donné. Ce travail est commun avec Bruno Salvy.

Sommaire

1 Introduction

Les séries de Neumann [Wat44, §16] sont des développements en série de fonction analytique dans la base des fonctions de Bessel $\mathrm{J}_n(x)$. Le nom de ces séries rend hommage à Carl Gottfried Neumann qui fut le premier à les définir dans [Neu67]. Un joli exemple de telle série est donné Hansen [Han75, formule 57.2.10 page 380] :

$$-\frac{x}{2}\,\mathrm{J}_1(x) = \sum_{n\in\mathbb{Z}} (-1)^n |n|\,\mathrm{J}_{2n}(x).$$

L'objet de ce chapitre est de montrer comment calculer algorithmiquement un tel développement. Pour cet exemple, le point de départ est une paire d'équations vérifiées par la famille $(\mathrm{J}_n(x))$:

$$x\,(\mathrm{J}_{n+1}(x) + \mathrm{J}_{n-1}(x)) = 2n\,\mathrm{J}_n\,(x)\,, \tag{8.1}$$

$$2\,\mathrm{J}'_n(x) = -\,\mathrm{J}_{n+1}(x) + \mathrm{J}_{n-1}(x). \tag{8.2}$$

Pour une suite (u) de coefficients de série de Neumann quelconque et un opérateur différentiel L, la notation (u^L) représente la suite des coefficients de la série de Neumann $L\cdot\sum_{n\in\mathbb{Z}} u_n\,\mathrm{J}_n(x)$.

On observe alors que la suite (u) des coefficients d'une série de Neumann quelconque est liée à la fois à la suite (u^x) (où l'exposant x est l'opérateur de multiplication par x) par la relation suivante déduite de (8.1) :

$$\frac{1}{n+1}u^x_{n+1} + \frac{1}{n-1}u^x_{n-1} = 2u_n, \tag{8.3}$$

et à la suite (u^{∂_x}) par la relation suivante déduite de (8.2) :

$$2u_n^{\partial_x} = u_{n+1} - u_{n-1}. \tag{8.4}$$

En effet l'équation (8.1) mène à la suite d'égalités suivante :

$$\begin{aligned}\sum_{n\in\mathbb{Z}} u_n x\,\mathrm{J}_n(x) &= \sum_{n\in\mathbb{Z}} u^x_n\,\mathrm{J}_n(x)\\ &= \sum_{n\in\mathbb{Z}} u^x_n \frac{1}{2n} x\,(\mathrm{J}_{n+1}(x) + \mathrm{J}_{n-1}(x))\\ &= \sum_{n\in\mathbb{Z}} \left(\frac{1}{2(n+1)}u^x_{n+1} + \frac{1}{2(n-1)}u^x_{n-1}\right) x\,\mathrm{J}_n(x).\end{aligned}$$

Pour en déduire (8.3), il faut pouvoir extraire les coefficients de J_n. Comme la suite des fonctions de Bessel est paire en n ($\mathrm{J}_n(x) = \mathrm{J}_{-n}(x)$), la suite de fonctions $x\,\mathrm{J}_n(x)$ l'est aussi. On a alors l'égalité suivante [1]

$$\sum_{n\in\mathbb{Z}} u_n x\,\mathrm{J}_n(x) = \sum_{n\in\mathbb{N}}{}' 2u_n x\,\mathrm{J}_n(x).$$

1. Le terme $\sum'$ est similaire à celui défini par la notation (2.16) page 23

La suite $\mathrm{J}_n(x)$ pour $n \in \mathbb{N}$ est à valuations strictement croissantes, la suite $x\,\mathrm{J}_n(x)$ pour $n \in \mathbb{N}$ l'est aussi. La suite des séries $x\,\mathrm{J}_n(x)$ est donc une famille libre, ce qui permet d'extraire les coefficients d'une série développée dans celle-ci. L'égalité précédente implique donc bien la relation entre les suites (u) et (u^x) souhaitée. De la même façon, en utilisant la relation (8.2), on montre la relation entre (u) et (u^{∂_x}).

Ces deux relations (8.3) et (8.4) vont nous permettre d'obtenir le développement en série de Neumann de $-x\,\mathrm{J}_1$. On part de l'équation différentielle satisfaite par $-x\,\mathrm{J}_1$:

$$x\left(-\frac{x}{2}\,\mathrm{J}_1(x)\right)'' - \left(-\frac{x}{2}\,\mathrm{J}_1(x)\right)' + x\left(-\frac{x}{2}\,\mathrm{J}_1(x)\right) = 0, \tag{8.5}$$

et on construit progressivement une relation entre $(u^{x\partial_x^2-\partial_x+x})$ et (u).

Pour commencer, la relation (8.3) suivie de l'application de l'égalité (8.4) itérée deux fois donnent :

$$\frac{2}{n+1}u_{n+1}^{x\partial_x^2} + \frac{2}{n-1}u_{n-1}^{x\partial_x^2} = 4u_n^{\partial_x^2} = 2u_{n+1}^{\partial_x} - 2u_{n-1}^{\partial_x} = u_{n+2} - 2u_n + u_{n-2}.$$

La relation (8.4) donne ensuite l'égalité :

$$\frac{2}{n+1}u_{n+1}^{\partial_x} + \frac{2}{n-1}u_{n-1}^{\partial_x} = \frac{1}{n+1}u_{n+2} - \frac{2}{(n-1)(n+1)}u_n - \frac{1}{n-1}u_{n-2}.$$

Par linéarité $u^{L_1+L_2} = (u^{L_1} + u^{L_2})$ pour tous L_1 et L_2, donc on peut combiner les égalités précédentes pour obtenir :

$$\frac{2}{n+1}u_{n+1}^{x\partial_x^2-\partial_x+x} + \frac{2}{n-1}u_{n-1}^{x\partial_x^2-\partial_x+x} = \\ u_{n+2} - 2u_n + u_{n-2} - \frac{1}{n+1}u_{n+2} + \frac{2}{(n+1)(n-1)}u_n + \frac{1}{n-1}u_{n-2} + 2u_n,$$

qui se réarrange en

$$2(n-1)u_{n+1}^{x\partial_x^2-\partial_x+x} + 2(n+1)u_{n-1}^{x\partial_x^2-\partial_x+x} = (n^2+n)u_{n-2} + (2n^2-4)u_n + (n^2-n)u_{n+2}. \tag{8.6}$$

Mais au vu de (8.5), la suite $(u^{x\partial_x^2-\partial_x+x})$ est nulle. Les coefficients de la série de Neumann de $x\,\mathrm{J}_1(x)$ vérifient donc la récurrence

$$(n^2+n)u_{n-2} + (2n^2-4)u_n + (n^2-n)u_{n+2} = 0,$$

qui admet bien comme solution la suite de terme général $(-1)^{n/2}\frac{|n|}{2}$ pour n pair et 0 sinon. Comme la solution est hypergéométrique, on peut aussi utiliser l'algorithme de Petkovšek [Pet92] pour résoudre cette récurrence et retrouver la formule souhaitée (on devra dans ce cas donner des conditions initiales à la récurrence).

Le fait que cette solution soit hypergéométrique implique aussi qu'il existe une récurrence à deux termes qui annule les coefficients de cette série de Neumann. En fait on peut calculer

Entrée: une équation différentielle, une famille de fonctions $\psi_n(x)$ avec des relations du même type que (8.1) et (8.2) (voir p.168)
Sortie: la récurrence vérifiée par les coefficients du développement en série $\sum u_n\psi_n(x)$ de toute solution de l'équation différentielle développable dans la base des ψ_n.

Algorithme 8.1: Algorithme de calcul de récurrence

cette récurrence directement en allant plus loin dans les calculs. En effet l'égalité (8.6) s'écrit

$$\left(6+5n+n^2+2(2+4n+n^2)S_n^2+(2+3n+n^2)S_n^4\right)\cdot u=\left(2(n+3)S_n+2(n+1)S_n^2\right)\cdot u^{x\partial_x^2-\partial_x+x}.$$

Le gcld (calculé par l'algorithme 4.3 page 58) entre les opérateurs des deux membres de cette égalité est non trivial et vaut $\left(n+3+(n+1)S_n^2\right)$:

$$\left(n+3+(n+1)S_n^2\right)\left(n+2+nS_n^2\right)\cdot u=\left(n+3+(n+1)S_n^2\right)S_n u^{x\partial_x^2-\partial_x+x}.$$

Un argument prouvé dans ce chapitre permet de diviser à gauche par ce gcld tout en conservant la relation d'égalité et donc d'obtenir une récurrence d'ordre 2 qui annule bien les coefficients

$$(n+2)\,u_n+nu_{n+2}=0.$$

Le but de ce chapitre est de généraliser le procédé développé sur cet exemple afin de calculer les récurrences satisfaites par les coefficients d'une large classe de séries de Fourier généralisées. Cet algorithme est résumé par *la boîte* 8.1

Le package Maple gfsRecurrence que j'ai écrit lors de cette thèse permet de calculer ces récurrences en utilisant les algorithmes développés dans ce chapitre. Par exemple le calcul de la récurrence des coefficients de la série de Neumann de $-\frac{x}{2}\,\mathrm{J}_1(x)$ en Maple se fait à l'aide du code suivant :

```
> with(gfsRecurrence);
```

$$[diffeqToGFSRec]$$

```
> deq := gfun[holexprtodiffeq](-x/2*BesselJ(1,x), y(x));
```

$$deq := \left\{\left(\frac{d^2}{dx^2}y(x)\right)x-\frac{d}{dx}y(x)+y(x)\,x, y(0)=0, \left(D^{(2)}\right)(y)(0)=-1/2\right\}$$

```
> diffeqToGFSRec(deq, y(x), u(n), functions=BesselJ(n,x), normalize);
```

$$(n+2)\,u(n)+nu(n+2)$$

On a vu que le calcul de ces récurrences pouvait servir à calculer ou vérifier des formes closes de coefficients de séries de Fourier généralisées. Des travaux [FW61, Luk59, LC61, Ver66, WL62] sont consacrés à donner et prouver les formes closes de coefficients de séries de Fourier généralisées lorsque les bases des fonctions sont hypergéométriques. Un grand nombre d'entre elles sont donc désormais accessibles simplement et de manière unifiée par le calcul formel.

Exemple 8.1 (Fields-Wimp([FW61]))**.**

$$
{}_{p+r}F_{q+s}\left(\begin{matrix}a_p,\ c_r\\ b_q,\ d_s\end{matrix}\middle| zx\right) =
$$
$$
\sum_{n=0}^{\infty}\frac{(a_p)_n(\alpha)_n(\beta)_n(-z)^n}{(b_q)_n(\gamma+n)_n n!}\ {}_{p+2}F_{q+1}\left(\begin{matrix}n+\alpha,\ n+\beta,\ n+a_p\\ 2n+\gamma+1,\ n+b_q\end{matrix}\middle| z\right){}_{r+2}F_{s+2}\left(\begin{matrix}-n,\ n+\gamma,\ c_r\\ \alpha,\ \beta,\ d_s\end{matrix}\middle| x\right).
$$

Dans cette formule, a_p signifie la suite $a_1, \ldots, a_p$. Le terme ${}_{p+r}F_{q+s}$ est la notation de la série hypergéométrique généralisée (voir définition 2.8 page 2.8). L'utilisation des récurrences permet de prouver simplement ces formules pour p, r, q, s fixés.

Exemple 8.2. Le problème de connexion [Tha64, Ask75, RZG95, Szw92] entre deux familles de polynômes est le développement en série des polynômes de la première famille dans la base qui forme la seconde. Par exemple les polynômes de Laguerre généralisés satisfont :

$$
L_k^{(a+b)}(x) = \sum_{n=0}^{k}\binom{b+k-n-1}{k-n}L_n^{(a)}(x). \tag{8.7}
$$

Pour obtenir automatiquement cette formule, on peut calculer la récurrence satisfaite par les coefficients en utilisant le code Maple suivant. On calcule d'abord l'équation différentielle qui annule la fonction $L_k^{(a+b)}$.

```
> deq := k*y(x)+(-x+1+a+b)*(diff(y(x), x))+x*(diff(diff(y(x), x), x));
```

$$
deq := ky(x) + (-x+1+a+b)\frac{d}{dx}y(x) + x\frac{d^2}{dx^2}y(x)
$$

Les procédures du package GFSRecurrence permettent de calculer la récurrence vérifiée par les coefficients du développement en série de Laguerre à partir de cet opérateur :

```
> rec := diffeqToGFSRec(deq, y(x), u(n), functions=LaguerreL(n,a,x), const=a,
normalize);
```

$$
rec := (n-k)\,u(n) + (-1+k-n+b)\,u(n+1)
$$

Cette récurrence est d'ordre 1, et admet comme solution les suites hypergéométriques de la forme :

$$
u_n = \alpha\binom{b+k-n-1}{k-n},
$$

où α est une constante. Il ne reste plus qu'à déterminer une condition initiale pour montrer que $\alpha = 1$. Le développement des polynômes de Laguerre dans la base monomiale [Nat10, http://dlmf.nist.gov/18.5.E12] :

$$
L_k^a(x) = \sum_{l=0}^{k}(-1)^l\frac{(a+l+1)_{k-l}}{(k-l)!l!}x^l,
$$

montre que le terme associé au monôme de degré k ne dépend pas de a. Lorsque $n = k$, le terme $\binom{b+k-n-1}{k-n}$ est égal à 1, ce qui nous fixe $\alpha = 1$ et conclut la preuve de la formule (8.7).

Exemple 8.3 ([Geg72])**.** Le même type de calcul fournit par exemple la relation classique sur les polynômes de Gegenbauer.

$$C_n^\lambda(x) := \sum_{k=0}^{\lfloor n/2 \rfloor} \frac{\Gamma(\nu)(n-k+\nu)\Gamma(k+\lambda-\nu)\Gamma(n-k+\lambda)}{\Gamma(\lambda)\Gamma(\lambda-\nu)k!\Gamma(n-k+\nu+1)} C_{n-2k}^\nu(x).$$

Une autre application importante de ces récurrences est le calcul numérique rapide des coefficients de ces séries. On a vu dans le chapitre 7 comment utiliser les récurrences pour calculer les coefficients d'une série de Tchebychev. On peut de la même façon utiliser ces récurrences pour calculer les coefficients d'autres séries de Fourier généralisées.

Historiquement, les premières récurrences calculées ont été celles que vérifient les coefficients de série de Taylor solution d'équation différentielle au XIX$^{\text{e}}$ siècle. À ma connaissance, c'est Clenshaw [Cle57] qui a pour la première fois utilisé les récurrences vérifiées par les coefficients d'autres séries, il s'agissait alors de séries de Tchebychev. Il n'a pas donné d'algorithme pour le calcul des récurrences mais a utilisé ces récurrences implicitement pour calculer numériquement les coefficients d'une série de Tchebychev solution d'une équation différentielle. Le chapitre 5 présente un historique des algorithmes pour le calcul des récurrences de Tchebychev. Stanisław Lewanowicz avait dès son premier article sur le sujet [Lew76], généralisé l'algorithme de Paszkowski [Pas75] sur le calcul des récurrences de Tchebychev au calcul des récurrences vérifiées par les coefficients de développement dans la base des polynômes de Gegenbauer avant de donner toute une série d'articles [Lew79, Lew83, Lew86, Lew91, Lew92] qui adaptent ces algorithmes aux séries de Jacobi. Les polynômes de Jacobi sont une généralisation des polynômes de Gegenbauer qui eux même généralisent les polynômes de Tchebychev. André Ronveaux, Alejandro Zarzo Altarejos et Eduardo Godoy Malvar [RZG95] proposent une méthode pour calculer les récurrences vérifiées par les coefficients de séries dans des bases de polynômes orthogonaux classiques. C'est-à-dire qu'en plus des polynômes de Jacobi, leurs méthodes s'appliquent aux polynômes de Laguerre et de Hermite par exemple. Stanisław Lewanowicz et Paweł Woźny [LW04] donnent une généralisation des algorithmes de Lewanowicz aux familles de polynômes orthogonaux semi-classiques [Ron79]. Ces familles sont telles que les polynômes dérivés sont aussi quasi-orthogonaux (c'est-à-dire que le produit scalaire entre deux polynômes dont la différence des indices est strictement plus grande que 1 est nul) ; la famille des dérivées d'une famille de polynômes orthogonaux classiques est une famille de polynômes orthogonaux, donc les polynômes orthogonaux semi-classiques généralisent les polynômes orthogonaux classiques. Plus récemment Luc Rebillard et Helena Zakrajšek [RZ07] ont proposé un algorithme pour calculer les récurrences des coefficients de solution d'équation différentielle développée en série dans une base de polynômes hypergéométriques [NU88] qui généralise encore les algorithmes précédents, les polynômes hypergéométriques étant une généralisation des polynômes orthogonaux semi-classiques.

L'algorithme proposé dans ce chapitre permet de calculer les récurrences autant pour les coefficients de série de Neumann que pour les coefficients des développements dans des

bases de polynômes hypergéométriques et donc couvre tous les exemples traités par les algorithmes précédents.

La première partie de ce chapitre donne la définition des séries de Fourier généralisées. Une théorie permettant d'obtenir l'algorithme 8.1 et plusieurs algorithmes alternatifs est proposée dans la section suivante. La dernière section montre comment abaisser l'ordre de l'opérateur de récurrence calculée.

2 Séries de Fourier généralisées

Les séries de Fourier généralisées qui vont nous intéresser sont des développements en séries de la forme :

$$\sum_{n\in\mathbb{Z}} c_n\psi_n(x), \tag{8.8}$$

où (ψ_n) est une famille de fonctions presque-orthogonales au sens suivant.

Définition 8.4. Une famille de fonctions *presque-orthogonales* $(\psi_n(x))$ sur un corps $\mathbb{K}$ est une famille de fonctions de variable x et d'indice n dont l'idéal annulateur dans l'anneau $\mathbb{K}[n,x]\langle S_n,\partial_x\rangle$, des opérateurs de décalage et de dérivation à coefficients polynomiaux en x et en n, est engendré par les deux équations suivantes que l'on nomme (mult_x) comme multiplication par x et (diff_x) comme différentiation par rapport à x :

$$\mathcal{X}_2\cdot(x\cdot\psi_n) = \mathcal{X}_1\cdot(\psi_n)\,, \tag{mult_x}$$
$$\mathcal{D}_2\cdot\left(\psi_n'\right) = \mathcal{D}_1\cdot(\psi_n)\,, \tag{diff_x}$$

où $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{D}_1$ et $\mathcal{D}_2$ sont des opérateurs de récurrence dans $\mathbb{K}(n)\langle S_n\rangle$. De plus il n'existe pas d'opérateur de récurrence R tel que :

$$R\cdot(\psi_n(x)) = 0.$$

Le terme presque-orthogonale est utilisé car l'équation (mult_x) est une généralisation de la relation de récurrence à trois termes vérifiée par les polynômes orthogonaux et l'équation (diff_x) est aussi vérifiée par une large classe de polynômes orthogonaux en particulier par l'ensemble des polynômes orthogonaux classiques. Des exemples de familles presque-orthogonales sont donnés dans le tableau 8.1.

Définition 8.5. Pour une famille de fonctions presque-orthogonales, On se fixe le 3-uplet $[\psi,\mathcal{F}_\psi,\mathcal{U}_\psi]$, où $\mathcal{F}_\psi$ est un espace métrique contenant la famille de fonctions (ψ) et $\mathcal{U}_\psi$ un espace vectoriel de suites, tel que pour tout $f\in\mathcal{F}_\psi$, il existe un unique $u\in\mathcal{U}_\psi$, tel que

$$\lim_{n\to\infty}\sum_{i=-n}^{n} u_i\psi_i = f.$$

Nom	$\psi_n(x)$	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{D}_1$	$\mathcal{D}_2$
base monomiale	x^n	S_n	1	$n+1$	S_n
polynômes de Tchebychev	$T_n(x)$	S_n^2+1	$2S_n$	$2S_n$	$\left(S_n^2-1\right)\frac{1}{n}$
polynômes de Gegenbauer	$C_n^{(\lambda)}$	$n+2\alpha+(n+2)S_n^2$	$(2n+2+2\alpha)S_n$	$(-2\alpha-2-2n)S_n$	$1-S_n^2$
polynômes de Jacobi	$P_n^{(\alpha,\beta)}$	$\frac{2(n+\alpha+1)(n+\beta+1)}{(2n+\alpha+\beta+3)_1}+\frac{(2n+\alpha+\beta+3)(\beta^2-\alpha^2)}{(2n+\alpha+\beta+4)_2}S_n+\frac{2(n+2)(n+\alpha+\beta+2)}{(2n+\alpha+\beta+4)_1}S_n^2$	S_n	$\frac{1}{2}(n+\alpha+\beta)(2n+\alpha+\beta+1)_3S_n$	$-(n+\alpha+1)(n+\beta+1)(2n+\alpha+\beta+3)+(\alpha-\beta)(n+\alpha+\beta+1)(2n+\alpha+\beta+3)S_n+(n+\alpha+\beta+1)_2(2n+\alpha+\beta+2)S_n^2$
fonctions de Bessel	$\mathrm{J}_n(x)$	$2(n+1)S_n$	S_n^2+1	$1-S_n^2$	$2S_n$
polynômes de Hermite	$H_n(x)$	$S_n^2+2(n+1)$	$2S_n$	$2(n+1)$	S_n
polynômes de Laguerre	$L_n^{(\alpha)}(x)$	$(n+1+\alpha)+(n+2)S_n^2-(2n+3+\alpha)S_n$	$-S_n$	1	$1-S_n$
fonctions de Legendre	$P_n^{\mu}(x)$	$(n-\mu+2)S_n^2+(n+\mu+1)$	$(2n+3)S_n$	$(n+2+\mu)(n+1)S_n+(n+4)(n+3-\mu)S_n^3$	$\frac{(n+2+\mu)(n+1+\mu)}{(2n+3)}-\frac{(n+4-\mu)(n+3-\mu)}{(2n+7)}S_n^4+\frac{2(2n+5)(n^2+5n+\mu^2+5)S_n^2}{(2n+3)(2n+7)}S_n^2$
fonctions hypergéométriques	${}_2F_1\left(\begin{smallmatrix}n,2\\1\end{smallmatrix}\middle\vert x\right)$	$-n+(2n+1)S_n-(n+1)S_n^2$	$(n-1)S_n-(n+1)S_n^2$	$(n-1)S_n-(n+1)S_n^2$	$1-S_n$

FIGURE 8.1 – Exemples de familles de fonctions presque-orthogonales

La somme

$$\sum_{i=-\infty}^{+\infty} u_i \psi_i,$$

est appelée développement en série de Fourier généralisée de f relativement à (ψ).

Exemple 8.6. Les séries formelles de Laurent sont des séries de Fourier généralisées. La famille de fonctions est $\psi_n(x) = x^n$. L'espace $\mathcal{F}_\psi$ est l'espace des séries formelles muni de la métrique $d(f,g) = 2^{-\operatorname{val}(f-g)}$, où $\operatorname{val}(f)$ est la valuation de la série formelle f. L'espace $\mathcal{U}_\psi$ est l'ensemble des suites indexées sur $\mathbb{Z}$.

Exemple 8.7. Les séries formelles sont des séries de Fourier généralisées. La famille de fonctions est $\psi_n(x) = x^n$. L'espace des fonctions est l'espace des séries formelles muni de la métrique $d(f,g) = 2^{-\operatorname{val}(f-g)}$. L'espace des suites est l'ensemble des suites (u) telles que $u_n = 0$ si $n < 0$.

Exemple 8.8. Les séries entières sont des séries de Fourier généralisées. La famille de fonctions est $\psi_n(x) = x^n$. L'espace des fonctions est l'ensemble des fonctions analytiques en 0. L'espace $\mathcal{U}_\psi$ est l'ensemble des suites (u) telles que $\limsup_{n\to\infty} \left(|u_n|^{1/n}\right)$ est finie et $u_n = 0$ si $n < 0$.

Exemple 8.9. Les séries de Tchebychev sont des séries de Fourier généralisées. La famille de fonctions est $\psi_n(x) = \mathrm{T}_n(x)$. L'espace $\mathcal{F}_\psi$ est l'ensemble des fonctions analytiques sur le segment $[-1,1]$ muni de la norme uniforme sur ce segment. L'espace $\mathcal{U}_\psi$ est l'ensemble des suites (u) symétriques et convergeant exponentiellement vers 0.

Exemple 8.10. Les séries de Neumann sont des séries de Fourier généralisées. La famille de fonctions est $\psi_n(x) = \mathrm{J}_n(x)$. L'espace $\mathcal{F}_\psi$ est l'ensemble des séries formelles muni de la métrique $d(f,g) = 2^{-\operatorname{val}(f-g)}$. L'espace $\mathcal{U}_\psi$ est l'ensemble des suites (u) symétriques.

En effet, J_n étant une série entière de valuation n et $\mathrm{J}_n = \mathrm{J}_{-n}$, pour toute série formelle $f = \sum_{n\in\mathbb{N}} u_n x^n$, on peut construire récursivement la série de Bessel $f = \sum_{n\in\mathbb{Z}} c_n \mathrm{J}_n(x)$, en remarquant que :

$$2c_k \mathrm{J}_k(x) \mod x^{k+1} = \sum_{n=0}^{k} u_n x^n - \sum_{n=-k+1}^{k-1} c_n \mathrm{J}_n(x) \mod x^{k+1}.$$

On a aussi l'égalité

$$u_k x^k = \sum_{n=0}^{k} c_n \mathrm{J}_n(x) - \sum_{n=-k+1}^{k-1} u_n x^n \mod x^{k+1},$$

qui montre l'unicité de la suite (c_n).

Certaines séries associées aux familles données dans le tableau 8.1 sont classiques comme les séries de Taylor ou de Tchebychev. D'autres sont moins classiques mais étudiées, par

exemple les séries de Laguerre et de Hermite sont utilisées pour développer des fonctions sur des droites (Laguerre sur les réels positifs et Hermite sur les réels [MD73, Boy01]). Une des qualités de ces développements est qu'ils ne sont pas trop perturbés par les singularités non réelles des fonctions développées.

L'égalité entre une fonction et son développement de Taylor en 0 n'est valable que dans un disque centré en 0 et ne contenant aucune singularité. Ce résultat est spécifique aux séries de Taylor ; les autres développements ont aussi des domaines de convergence ne contenant pas les singularités. Par exemple l'égalité entre une fonction et son développement de Tchebychev est valable dans une ellipse de foyers -1 et 1 et ne contenant pas les singularités de la fonction (voir chapitre 2). En particulier le segment $[-1, 1]$ est inclus dans cette ellipse. Le domaine de convergence des séries de Laguerre est une bande contenant la demi-droite $[0, \infty]$ et aucune singularité [Boy01, page 353]. Le domaine de convergence des séries d'Hermite est une bande contenant l'axe réel et aucune singularité [Boy01, page 346]. L'exemple suivant illustre ces domaines de convergence.

Exemple 8.11. La figure 8.11 donne un exemple de l'utilisation de ces séries avec la fonction $\exp(1/(1+2x^2))$ qui admet deux singularités en $i/2$ et en $-i/2$.

Dans ce graphe, l'approximation de la fonction $f = \exp\left(\frac{1}{1+2x^2}\right)$ par une série de Taylor tronquée est mauvaise en dehors du segment $[-1/2, 1/2]$. La fonction f admet deux singularités en $i/2$ et $-i/2$, le rayon de convergence de la série de Taylor est donc $1/2$. Les singularités étant imaginaires pures, la série de Tchebychev tronquée approxime correctement la fonction f sur une ellipse de foyer $[-1, 1]$ et ne contenant pas les points $i/2$ et $-i/2$. On voit qu'en dehors de cette ellipse, l'approximation devient mauvaise. Sur ce graphe encore les séries de Laguerre et Hermite qui approximent f, comme prévu, sur l'axe réel positif pour Laguerre et l'axe réel pour Hermite.

Ces exemples montrent aussi qu'en fonction du besoin, il est utile d'avoir le choix entre différentes approximations.

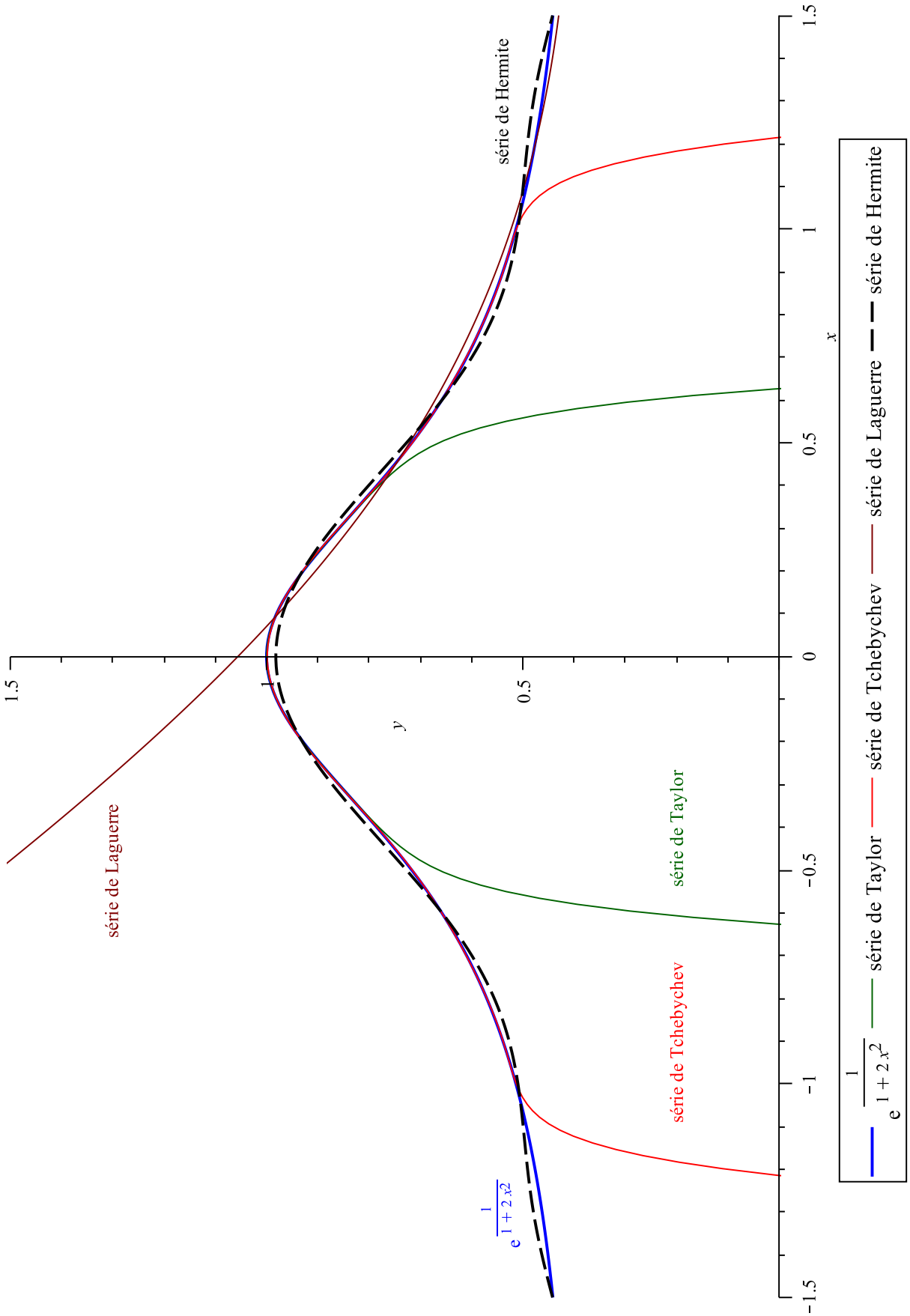
Une propriété fondamentale des séries de Fourier généralisées qui sera démontrée dans la section 3 est que si elles sont solutions d'équations différentielles linéaires à coefficients polynomiaux, alors leurs coefficients sont solutions d'une récurrence linéaire à coefficients polynomiaux.

On peut remarquer que les séries de Fourier n'ont jamais été évoquées jusqu'ici. En fait ces séries ne satisfont pas les bonnes propriétés pour vérifier la propriété précédente.

3 Récurrences des coefficients de séries généralisées

Le chapitre 5 exhibe un morphisme d'anneaux entre l'anneaux des opérateurs différentiels et le corps des fractions d'opérateurs de récurrence. Ce morphisme établit le lien entre l'opérateur différentiel annulant une série de Tchebychev et l'opérateur de récurrence qui annule ses coefficients. Cette section présente une généralisation de cette construction à toutes les séries de Fourier généralisées. L'image par le morphisme d'un opérateur différentiel

FIGURE 8.2 – Approximation de la fonction $\exp(\frac{1}{1+2x^2})$



L est une paire d'opérateurs de récurrence permettant de relier les coefficients d'une série de Fourier généralisée f aux coefficients de la série $L \cdot f$.

Pour simplifier la lecture, la notation suivante est utilisée :

Notation 8.12. Soient ψ une famille de fonctions presque-orthogonales, $\mathcal{F}_\psi$ un espace métrique de fonctions et $\mathcal{U}_\psi$ un espace de suites associés à $\mathcal{F}_\psi$.

Pour tout opérateur différentiel L et toute suite $u \in \mathcal{U}_\psi$ telle que $\sum u_n \psi_n(x) \in \mathcal{F}_\psi$ et $L \cdot \sum u_n \psi_n(x) \in \mathcal{F}_\psi$, on note (u^L) la suite dans $\mathcal{U}_\psi$ associée qui vérifie :

$$L \cdot \sum_{n\in\mathbb{Z}} u_n \psi_n(x) = \sum_{n\in\mathbb{Z}} u_n^L \psi_n(x).$$

3.1 Paires d'opérateurs de récurrence associées à une famille de fonctions presque-orthogonales

La propriété fondamentale satisfaite par les familles de fonctions presque-orthogonales est résumée dans le théorème suivant. Cette propriété découle naturellement des équations (mult_x) et (diff_x).

Théorème 8.13. *Soit $(\psi_n)_{n\in\mathbb{Z}}$ une famille de fonctions presque-orthogonales. Pour tout opérateur différentiel linéaire à coefficients polynomiaux L, il existe une paire d'opérateurs de récurrence $(\mathcal{A}_1, \mathcal{A}_2)$, telle que :*

$$\mathcal{A}_2 \cdot (L \cdot \psi_n(x)) = \mathcal{A}_1 \cdot \psi_n(x). \tag{8.9}$$

Démonstration. Si L est la multiplication par x ou la dérivation, les paires données par les égalités (mult_x) et (diff_x) satisfont l'équation (8.9). Si L est une constante λ, la paire d'opérateurs $(\lambda, 1)$ satisfait aussi l'équation.

Pour montrer ce théorème pour un opérateur différentiel quelconque, il suffit de montrer que si la propriété est vraie pour les opérateurs L_1 et L_2, alors elle l'est aussi pour les opérateurs $L_1 + L_2$ et $L_2 L_1$. Soient les paires $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ telles que :

$$\begin{aligned} \mathcal{A}_2 \cdot (L_1 \cdot \psi_n(x)) &= \mathcal{A}_1 \cdot \psi_n(x), \\ \mathcal{B}_2 \cdot (L_2 \cdot \psi_n(x)) &= \mathcal{B}_1 \cdot \psi_n(x). \end{aligned}$$

En utilisant la formule d'addition de paires (4.14) page 70 et la proposition 4.29 qui en découle, on a :

$$\operatorname{lclm}(\mathcal{A}_1, \mathcal{B}_1) \cdot (L_1 + L_2) \cdot \psi_n(x) = \left((\mathcal{B}_1)^{\mathcal{A}_1} \mathcal{A}_2 + (\mathcal{A}_1)^{\mathcal{B}_1} \mathcal{B}_2 \right) \cdot \psi_n(x),$$

donc la paire $(\mathcal{A}_1, \mathcal{A}_2) + (\mathcal{B}_1, \mathcal{B}_2)$ est associée à l'opérateur $L_1 + L_2$. La propriété est donc vraie pour l'addition de deux opérateurs différentiels. De la même façon, la définition de la multiplication entre opérateurs (4.15) page 71 et la proposition 4.32 qui en découle montrent que la paire $((\mathcal{A}_1)^{\mathcal{B}_2} \mathcal{B}_1, (\mathcal{B}_2)^{\mathcal{A}_1} \mathcal{A}_2)$ est associée à l'opérateur $L_2 L_1$, ce qui conclut la preuve. □

La proposition suivante nous donne la relation entre deux paires associées à un même opérateur différentiel.

Proposition 8.14. *Soient $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ deux paires d'opérateurs de récurrence, L un opérateur différentiel et $(\psi_n)_{n\in\mathbb{N}}$ une famille de fonctions presque-orthogonales. Si*

$$\begin{aligned} \mathcal{A}_2 \cdot (L \cdot \psi_n(x)) &= \mathcal{A}_1 \cdot \psi_n(x) \quad \text{et} \\ \mathcal{B}_2 \cdot (L \cdot \psi_n(x)) &= \mathcal{B}_1 \cdot \psi_n(x), \end{aligned}$$

alors les paires $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ sont équivalentes (noté aussi $(A_1, A_2) \equiv (B_1, B_2)$) au sens de la définition 4.37 page 76.

Démonstration. En multipliant la première égalité par $(\mathcal{B}_2)^{\mathcal{A}_2}$ et la seconde par $(\mathcal{A}_2)^{\mathcal{B}_2}$, puis en les soustrayant, on obtient :

$$\Big((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1 - (\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1\Big) \cdot \psi_n(x) = 0.$$

La définition 8.4 page 173 entraîne $(\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1 = (\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1$, les paires (A_1, A_2) et (B_1, B_2) sont donc bien équivalentes. □

De ce théorème se déduit le corollaire fondamental suivant, qui exprime la règle de Leibniz pour les paires d'opérateurs de récurrence.

Corollaire 8.15. *Les opérateurs $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{D}_1$ et $\mathcal{D}_2$ associés à une famille de fonctions presque-orthogonales par les formules* (mult_x) *et* (diff_x) *vérifient*

$$(\mathcal{X}_1, \mathcal{X}_2)(\mathcal{D}_1, \mathcal{D}_2) \equiv (\mathcal{D}_1, \mathcal{D}_2)(\mathcal{X}_1, \mathcal{X}_2) + (1, 1).$$

Démonstration. La proposition 8.14 nous indique qu'il suffit de trouver un opérateur différentiel L associé aux deux fractions de l'énoncé du corollaire pour montrer leur équivalence.

Soient $(\mathcal{A}_1, \mathcal{A}_2) = (\mathcal{D}_1, \mathcal{D}_2)(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2) = (\mathcal{X}_1, \mathcal{X}_2)(\mathcal{D}_1, \mathcal{D}_2)$. La proposition 4.32 page 71 implique les égalités suivantes :

$$\begin{aligned} & \mathcal{A}_2 \cdot ((x\partial_x) \cdot \psi_n(x)) = \mathcal{A}_1 \cdot \psi_n(x) \\ \text{et} \quad & \mathcal{B}_2 \cdot ((\partial_x x) \cdot \psi_n(x)) = \mathcal{B}_1 \cdot \psi_n(x). \end{aligned}$$

La proposition 4.29 page 70 implique que la paire $(\mathcal{C}_1, \mathcal{C}_2) = (\mathcal{B}_1, \mathcal{B}_2) + (1, 1)$ vérifie la propriété :

$$\mathcal{C}_2 \cdot ((\partial_x x + 1) \cdot \psi_n(x)) = \mathcal{C}_1 \cdot \psi_n(x).$$

La règle de Leibniz implique que $\partial_x x = \partial_x x + 1$ ce qui prouve l'équivalence entre $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{C}_1, \mathcal{C}_2)$. □

3.2 Paires adjointes d'opérateurs de récurrence

Les opérateurs de récurrence adjoints sont des outils importants dans la théorie des opérateurs de récurrence. Une des applications classiques est la factorisation d'un opérateur de récurrence. En effet l'algorithme de Petkovšek [Pet92] permet trouver un facteur droit d'ordre 1. La division à droite entre opérateurs devient une division à gauche entre les adjoints des mêmes opérateurs. Pour chercher un facteur gauche d'ordre 1 d'un opérateur, il suffit donc de calculer son adjoint et d'appliquer l'algorithme de Petkovšek comme suggéré dans [PWZ96, p. 163]. De cette façon, on sait par exemple trouver un facteur droit irréductible d'ordre 2 d'un opérateur d'ordre 3 en cherchant un facteur gauche d'ordre 1. Cette section introduit la notion de paire adjointe d'opérateurs de récurrence. Celle-ci est essentielle pour faire le lien entre les paires appliquées à une famille de fonctions presque-orthogonales et les coefficients des séries de Fourier généralisées développées dans cette famille. On rappelle d'abord la définition d'un opérateur de récurrence adjoint.

Définition 8.16. L'opérateur de récurrence

$$\mathcal{R} = \sum_{k=0}^{s} l_k(n) S_n^k,$$

a pour *opérateur de récurrence adjoint* l'opérateur noté $\mathcal{R}^*$ défini par :

$$\mathcal{R}^* = \sum_{k=0}^{s} l_k(n-k) S_n^{-k}.$$

Les lemmes suivants découlent naturellement de cette définition.

Lemme 8.17 ([RZ07])**.** *Pour tout opérateur $\mathcal{R}$, l'adjoint $\mathcal{R}^*$ vérifie l'égalité suivante :*

$$\sum_{n\in\mathbb{Z}} u_n \mathcal{R}\cdot v_n = \sum_{n\in\mathbb{Z}} (\mathcal{R}^*\cdot u_n) v_n,$$

où (u) et (v) sont des suites définies sur $\mathbb{Z}$.

Démonstration. Si $\mathcal{R} = \sum_{k=0}^{s} l_k(n) S_n^k$ on a l'égalité suivante :

$$\sum_{n\in\mathbb{Z}} u_n \mathcal{R}\cdot v_n = \sum_{n\in\mathbb{Z}} \sum_{k=0}^{s} u_n l_k(n) v_{n+k} = \sum_{k=0}^{s} \sum_{n\in\mathbb{Z}} u_n l_k(n) v_{n+k}.$$

En effectuant le changement de variable $n \mapsto n-k$, on a :

$$\sum_{n\in\mathbb{Z}} u_n l_k(n) v_{n+k} = \sum_{n\in\mathbb{Z}} u_{n-k} l_k(n-k) v_n.$$

On a donc

$$\sum_{n\in\mathbb{Z}} u_n \mathcal{R}\cdot v_n = \sum_{n\in\mathbb{Z}} \sum_{k=0}^{s} u_{n-k} l_k(n-k) v_n.$$

ce qui est bien égal à $\sum(\mathcal{R}^*\cdot u_n)v_n$. □

Lemme 8.18 ([PWZ96])**.** *Les opérateurs de récurrence $\mathcal{R}_1$ et $\mathcal{R}_2$, vérifient l'égalité :*

$$\mathcal{R}_1^*\mathcal{R}_2^* = (\mathcal{R}_2\mathcal{R}_1)^*. \tag{8.10}$$

Démonstration. Par linéarité $\mathcal{R}_1^* + \mathcal{R}_2^* = (\mathcal{R}_2 + \mathcal{R}_1)^*$, il suffit donc de prouver ce lemme pour le produit de deux monômes.

Le produit de

$$\mathcal{R}_1 = l_1(n)S_n^i \quad \text{et} \quad \mathcal{R}_2 = l_2(n)S_n^j.$$

donne l'égalité :

$$\mathcal{R}_2\mathcal{R}_1 = l_2(n)l_1(n+j)S_n^{i+j},$$

L'adjoint de ce produit est :

$$(\mathcal{R}_2\mathcal{R}_1)^* = l_2(n-i-j)l_1(n-i)S_n^{-i-j}.$$

On retrouve bien le produit des monômes $l_1(n-i)S_n^{-i}$ et $l_2(n-j)S_n^{-j}$, c'est-à-dire le produit $\mathcal{R}_1^*\mathcal{R}_2^*$. □

L'objectif de la construction de paires adjointes d'opérateurs de récurrence est l'obtention d'un résultat similaire au lemme 8.17. C'est-à-dire, si pour une famille de fonctions presque-orthogonales $\psi_n(x)$, et pour un opérateur différentiel L, il existe une paire qui lie $\psi_n(x)$ et $L \cdot \psi_n(x)$ alors on peut construire automatiquement une paire qui lie (u) et (u^L).

Définition 8.19. Pour une paire d'opérateurs de récurrence $(\mathcal{A}_1, \mathcal{A}_2)$, la paire (A_1, A_2) appelée *paire d'opérateurs de récurrence adjointe* à la paire $(\mathcal{A}_1, \mathcal{A}_2)$ est définie par :

$$(A_1, A_2) = (\mathcal{A}_1, \mathcal{A}_2)^* = \left((\mathcal{A}_1^*)^{\mathcal{A}_2^*}, (\mathcal{A}_2^*)^{\mathcal{A}_1^*}\right).$$

On rappelle ici que les cofacteurs du lclm entre deux polynômes de Laurent A et B ont été définis dans le chapitre 4 par :

$$(A)^B B = (B)^A A = \mathrm{lclm}(S_n^{d_A+d_B}A, S_n^{d_A+d_B}B).$$

Propriété 8.20. *Pour toute paire, on a la relation :*

$$(\mathcal{A}_1, \mathcal{A}_2)^* \equiv (\mathcal{A}_1^*, 1)\,(1, \mathcal{A}_2^*)\,.$$

Démonstration. L'application de la règle de multiplication entre paires pour multiplier les deux paires du membre gauche de l'équation entraîne la validité du résultat. □

Remarque 8.21. Si le second membre de la paire est 1, on a $A_1 = S_n^{d_A}\mathcal{A}_1^*$ et $A_2 = S_n^{d_A}$ où d_A est l'ordre de l'opérateur $\mathcal{A}_1$, donc l'adjoint de la paire $(\mathcal{A}_1, 1)$ est $(S_n^{d_A}\mathcal{A}_1^*, S_n^{d_A})$ qui est équivalent à la paire $(\mathcal{A}_1^*, 1)$. Il s'agit donc bien d'une généralisation des opérateurs adjoints de récurrence. Le lemme 8.17 est généralisé par la proposition 8.23 ci-desous.

Remarque 8.22. Pour toute paire $(\mathcal{A}_1, \mathcal{A}_2)$, les opérateurs de la paire adjointe vérifient $\mathrm{gcld}(A_1, A_2) = 1$. En effet par définition du lclm, les opérateurs $(\mathcal{A}_1^*)^{\mathcal{A}_2^*}$ et $(\mathcal{A}_2^*)^{\mathcal{A}_1^*}$ sont premiers entre eux.

Proposition 8.23. *Soient L un opérateur différentiel et un 3-uplet $[\psi, \mathcal{F}_\psi, \mathcal{U}_\psi]$ (voir définition 8.5 page 173). Si la paire $(\mathcal{A}_1, \mathcal{A}_2)$ est telle que :*

$$\mathcal{A}_2 \cdot (L \cdot \psi_n(x)) = \mathcal{A}_1 \cdot \psi_n(x),$$

et vérifie en plus la propriété que pour tout $u \in \mathcal{U}_\psi$ il existe une suite (v) telle que $\mathcal{A}_2^* \cdot v = u$ et $\mathcal{A}_1^* \cdot v \in \mathcal{U}_\psi$, alors toute suite $u \in \mathcal{U}_\psi$ est liée à (u^L) par :

$$A_1 \cdot u = A_2 \cdot u^L, \tag{8.11}$$

où la paire (A_1, A_2) est la paire adjointe de $(\mathcal{A}_1, \mathcal{A}_2)$.

Démonstration. Soit (v) une suite telle que $\mathcal{A}_2^* \cdot v = u$ et $\mathcal{A}_1^* \cdot v \in \mathcal{U}_\psi$. On a alors la suite d'égalités suivantes :

$$L \cdot \sum_{n\in\mathbb{Z}} u_n \psi_n(x) = \sum_{n\in\mathbb{Z}} (\mathcal{A}_2^* \cdot v_n)(L \cdot \psi_n(x)) = \sum_{n\in\mathbb{Z}} v_n \mathcal{A}_2 \cdot (L \cdot \psi_n(x)).$$

On a par hypothèse :

$$\mathcal{A}_1 \cdot \psi_n(x) = \mathcal{A}_2 \cdot (L \cdot \psi_n(x)),$$

donc

$$L \cdot \sum_{n\in\mathbb{Z}} u_n \psi_n(x) = \sum_{n\in\mathbb{Z}} v_n (\mathcal{A}_1 \cdot \psi_n(x)) = \sum_{n\in\mathbb{Z}} (\mathcal{A}_1^* \cdot v_n) \psi_n(x).$$

Les suites $(\mathcal{A}_1^* v)$ et (u^L) sont dans U_ψ, par unicité du développement en série de Fourier généralisée des fonctions dans $\mathcal{F}_\psi$, on a donc $u^L = \mathcal{A}_1^* \cdot v$. Si on utilise la définition des cofacteurs de lclm, on obtient la suite d'égalités suivantes :

$$A_1 \cdot u = (\mathcal{A}_1^*)^{\mathcal{A}_2^*} \cdot u = (\mathcal{A}_1^*)^{\mathcal{A}_2^*} \mathcal{A}_2^* \cdot v = (\mathcal{A}_2^*)^{\mathcal{A}_1^*} \mathcal{A}_1^* \cdot v = (\mathcal{A}_2^*)^{\mathcal{A}_1^*} \cdot u^L = A_2 \cdot u^L. \tag{8.12}$$

□

Les paires adjointes vérifient des propriétés pour l'addition et la multiplication similaires aux propriétés d'addition et de multiplication d'opérateurs adjoints. Ces propriétés sont résumées par les propositions suivantes.

Proposition 8.24. *Si les trois paires $(\mathcal{A}_1, \mathcal{A}_2)$, $(\mathcal{B}_1, \mathcal{B}_2)$ et $(\mathcal{C}_1, \mathcal{C}_2)$ sont reliées par*

$$(\mathcal{A}_1, \mathcal{A}_2) + (\mathcal{B}_1, \mathcal{B}_2) = (\mathcal{C}_1, \mathcal{C}_2),$$

alors

$$(A_1, A_2) + (B_1, B_2) \equiv (C_1, C_2).$$

Démonstration. On souhaite montrer que :

$$(\mathcal{A}_1, \mathcal{A}_2)^* + (\mathcal{B}_1, \mathcal{B}_2)^* \equiv \left(\left((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1\right) + \left((\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1\right), \mathrm{lclm}\,(\mathcal{A}_2, \mathcal{B}_2)\right)^*.$$

Le terme de gauche est équivalent à

$$\left((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1, \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)\right)^* + \left((\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1, \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)\right)^*.$$

Partant de la propriété 8.20, cette expression se réécrit comme

$$\left(\left((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1\right)^*, 1\right)\left(1, \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)^*\right) + \left(\left((\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1\right)^*, 1\right)\left(1, \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)^*\right).$$

En factorisant cette expression par le terme commun à gauche, on obtient la paire équivalente :

$$\left(\left((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1\right)^* + \left((\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1\right)^*, 1\right)\left(1, \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)^*\right),$$

qui est bien, par définition d'une paire adjointe, la paire recherchée :

$$\left(\left((\mathcal{B}_2)^{\mathcal{A}_2}\mathcal{A}_1\right) + \left((\mathcal{A}_2)^{\mathcal{B}_2}\mathcal{B}_1\right), \mathrm{lclm}\,(\mathcal{B}_2, \mathcal{A}_2)\right)^*.$$

□

Proposition 8.25. *Si les trois paires* $(\mathcal{A}_1, \mathcal{A}_2)$, $(\mathcal{B}_1, \mathcal{B}_2)$ *et* $(\mathcal{C}_1, \mathcal{C}_2)$ *sont reliées par*

$$(\mathcal{B}_1, \mathcal{B}_2)(\mathcal{A}_1, \mathcal{A}_2) = (\mathcal{C}_1, \mathcal{C}_2),$$

alors :

$$(A_1, A_2)(B_1, B_2) \equiv (C_1, C_2).$$

Démonstration. On souhaite montrer que :

$$(\mathcal{A}_1, \mathcal{A}_2)^*\,(\mathcal{B}_1, \mathcal{B}_2)^* \equiv \left((\mathcal{A}_1)^{\mathcal{B}_2}\mathcal{B}_1, (\mathcal{B}_2, \mathcal{A}_1)_2^{\mathcal{A}}\right)^*.$$

Par la propriété 8.20, le membre gauche de cette équation est équivalent à

$$(\mathcal{A}_1^*, 1)\,(1, \mathcal{A}_2^*)\,(\mathcal{B}_1^*, 1)\,(1, \mathcal{B}_2^*). \tag{8.13}$$

Afin d'exprimer ce terme comme une paire adjointe, on réexprime le produit de paires du milieu (en utilisant la définition de multiplication de paires (4.15) 71), comme :

$$(1, \mathcal{A}_2^*)\,(\mathcal{B}_1^*, 1) \equiv (\mathcal{A}_1^*, \mathcal{B}_2^*) \equiv \left(\left((\mathcal{A}_1)^{\mathcal{B}_2}\right)^*, 1\right)\left(1, \left((\mathcal{B}_1)^{\mathcal{A}_1}\right)^*\right).$$

On peut alors simplifier l'équation (8.13) par le terme équivalent :

$$(\mathcal{A}_1^*, 1)\left(\left((\mathcal{A}_1)^{\mathcal{B}_2}\right)^*, 1\right)\left(1, \left((\mathcal{B}_1)^{\mathcal{A}_1}\right)^*\right)(1, \mathcal{B}_2^*) \equiv \left(\left((\mathcal{A}_1)^{\mathcal{B}_2}\mathcal{A}_1\right)^*, 1\right)\left(1, \left(\mathcal{B}_2(\mathcal{B}_1)^{\mathcal{A}_1}\right)^*\right),$$

qui est équivalent au terme droit de la relation recherchée par la propriété 8.20 . □

Entrée: $L := \sum_{i=0}^{k} p_i(x)\partial^i$ et ψ muni de 4 opérateurs de récurrence $\mathcal{X}_1,\ \mathcal{X}_2,\ \mathcal{D}_1,\ \mathcal{D}_2$ tels que $\mathcal{X}_1 \cdot \psi = \mathcal{X}_2 \cdot x\psi$ et $\mathcal{D}_1 \cdot \psi = \mathcal{D}_2 \cdot \psi'$
Sortie: P_1 tel que $\varphi_\psi(L) = P_2^{-1}P_1$

Algorithme 8.2: Morphisme pour le calcul des récurrences

3.3 Morphisme dans le corps des fractions d'opérateurs de récurrence

Cette section explique comment à l'aide des paires adjointes d'opérateurs de récurrence, on peut définir un morphisme d'anneaux φ_ψ de l'anneau des opérateurs différentiels dans le corps des fractions d'opérateurs de récurrence. L'image d'un opérateur différentiel L par ce morphisme, qui dépend d'une famille de fonctions presque-orthogonales ψ, est une fraction d'opérateurs dont le numérateur est la récurrence qui annule les coefficients des séries de Fourier généralisées solutions de L. Ainsi, ce morphisme répond au besoin de l'algorithme, qui permet de rendre équivalents les algorithmes 8.1 page 170 et 8.2.

Avant de définir le morphisme d'anneaux, on va définir le morphisme d'anneaux $\bar{\varphi}_\psi$ qui va de l'algèbre libre des opérateurs différentiels $\mathbb{K}\langle x, \partial_x\rangle$ (les variables ne commutent pas), dans le corps des fractions d'opérateurs de récurrence. On déduira φ_ψ par quotient.

Définition 8.26. Pour une famille de fonctions presque-orthogonales ψ, le morphisme d'algèbres $\bar{\varphi}_\psi$ de l'algèbre libre $\mathbb{K}\langle x, \partial_x\rangle$ dans le corps des fractions d'opérateurs de récurrence est défini par :

$$\bar{\varphi}_\psi(x) = X_2^{-1}X_1 \quad \text{et} \quad \bar{\varphi}_\psi(\partial_x) = D_2^{-1}D_1,$$

où (X_1, X_2) et (D_1, D_2) sont respectivement les paires adjointes de $(\mathcal{X}_1, \mathcal{X}_2)$ (mult_x) et $(\mathcal{D}_1, \mathcal{D}_2)$ (diff_x).

En quotientant l'algèbre libre $\mathbb{K}\langle x, \partial_x\rangle$ par l'idéal bilatère engendré par le polynôme $x\partial_x - \partial_x x - 1$, on obtient l'anneau des opérateurs différentiels $\mathbb{K}[x]\langle\partial_x\rangle$ (4.3 page 56). La proposition suivante montre que ce quotient conduit naturellement à définir φ_ψ à partir de $\bar{\varphi}_\psi$.

Proposition 8.27. *Pour toute famille de fonctions presque-orthogonales ψ, on a l'égalité suivante :*

$$\bar{\varphi}_\psi(x\partial_x - \partial_x x - 1) = 0.$$

Démonstration. Cette proposition revient à montrer l'égalité

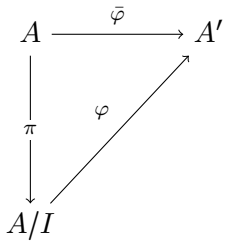
$$X_2^{-1}X_1D_2^{-1}D_1 - D_2^{-1}D_1X_2^{-1}X_1 = 1. \tag{8.14}$$

Le corollaire 8.15 page 179 montre que les paires d'opérateurs $(\mathcal{D}_1, \mathcal{D}_2)(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{X}_1, \mathcal{X}_2)(\mathcal{D}_1, \mathcal{D}_2)+(1, 1)$ sont équivalentes.

Par les propositions 8.24 et 8.25, on en déduit que les paires adjointes$(X_1, X_2)(D_1, D_2)$ et $(D_1, D_2)(X_1, X_2) - 1$ sont équivalentes. On a donc bien l'égalité (8.14). □

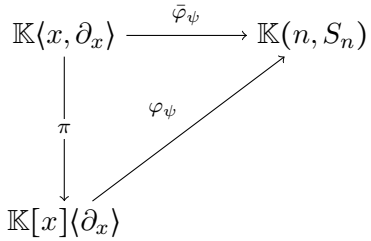
Cette proposition combinée au théorème classique suivant, permet de définir le morphisme φ_ψ.

Théorème 8.28. *[Lan02, page 89] Si $\bar{\varphi} : A \to A'$ est un morphisme d'anneaux dont le noyau contient l'idéal I, alors il existe un unique morphisme d'anneaux $\varphi : A/I \to A'$ qui rend le diagramme suivant commutatif :*



Ici π est l'application canonique : $\pi : A \to A/I$.

On applique ce théorème avec $A = \mathbb{K}\langle x, \partial_x\rangle$ et I l'idéal engendré par l'opérateur $x\partial_x - \partial_x x - 1$, on a alors $A/I = \mathbb{K}[x]\langle\partial_x\rangle$ comme on a vu dans la remarque 4.3 page 56. On identifie aussi A' avec le corps des fractions d'opérateurs de récurrence $\mathbb{K}(n, S_n)$. Selon la proposition 8.2, le noyau de $\bar{\varphi}_\psi$ contient I, le théorème précédent nous dit donc que le diagramme suivant est commutatif :



Notation 8.29. Dans la suite de ce chapitre φ_ψ sera toujours le morphisme défini par le diagramme précédent avec $\bar{\varphi}_\psi$ défini par 8.26

On peut à ce stade annoncer le théorème principal de ce chapitre. Ce théorème prouve que la sortie de l'algorithme 8.2 est bien la même que celle annoncée par l'algorithme 8.1 page 170.

Théorème 8.30. *Soient $\psi_n(x)$ une famille de fonctions presque-orthogonales et L un opérateur différentiel. Soit $A_2^{-1}A_1$ l'image de L par le morphisme φ_ψ (en utilisant la notation 8.29). Soient les suites (u) et (u^L) de coefficients de séries de Fourier généralisées telles que :*

$$L \cdot \sum_{n\in\mathbb{Z}} u_n\psi_n(x) = \sum_{n\in\mathbb{Z}} u_n^L\psi_n(x).$$

Ces coefficients vérifient alors l'égalité suivante :

$$A_1 \cdot u = A_2 \cdot u^L.$$

En particulier si $\sum u_n \psi_n(x)$ *est annulée par* L, (u) *est annulée par* A_1.

Ce théorème est un corollaire immédiat de la proposition 8.23 et du lemme suivant :

Lemme 8.31. *Soient* L *un opérateur différentiel et* $(\mathcal{A}_1, \mathcal{A}_2)$ *une paire d'opérateurs tels que :*

$$\mathcal{A}_2 \cdot (L \cdot \psi_n(x)) = \mathcal{A}_1 \cdot \psi_n(x). \tag{8.15}$$

Alors $\varphi_\psi(L)$ *est la paire adjointe de* $(\mathcal{A}_1, \mathcal{A}_2)$.

Démonstration. Si L est l'opérateur différentiel de multiplication par x ou de dérivation, la propriété est vraie par définition des fractions $X_2^{-1}X_1$ et $D_2^{-1}D_1$. Si L est l'opérateur de multiplication par une constante λ, alors la proposition est vraie pour la fraction λ (définie par la paire $(\lambda, 1)$).

En montrant que si la propriété est vraie pour deux opérateurs différentiels $L_\mathcal{A}$ et $L_\mathcal{B}$, alors elle est vraie pour les opérateurs différentiels $L_\mathcal{A} + L_\mathcal{B}$ et $L_\mathcal{A}L_\mathcal{B}$, on peut conclure que la propriété est vraie pour n'importe quel opérateur différentiel.

On suppose donc que la proposition est vraie pour deux opérateurs différentiels $L_\mathcal{A}$ et $L_\mathcal{B}$. C'est-à-dire que pour toutes les paires d'opérateurs $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ associées respectivement à $L_\mathcal{A}$ et $L_\mathcal{B}$ (le théorème 8.13 nous indique qu'il en existe toujours), les paires adjointes de $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ sont respectivement les images de $\varphi_\psi(L_\mathcal{A})$ et $\varphi_\psi(L_\mathcal{B})$.

Soient $(\mathcal{A}_1, \mathcal{A}_2)$ et $(\mathcal{B}_1, \mathcal{B}_2)$ deux paires associées respectivement à $L_\mathcal{A}$ et $L_\mathcal{B}$, c'est-à-dire :

$$\mathcal{A}_2 \cdot (L_\mathcal{A} \cdot \varphi_n(x)) = \mathcal{A}_1 \cdot \varphi_n(x) \quad \text{et} \quad \mathcal{B}_2 \cdot (L_\mathcal{B} \cdot \varphi_n(x)) = \mathcal{B}_1 \cdot \varphi_n(x)$$

Selon la proposition 4.29 page 70, la paire $(\mathcal{C}_1, \mathcal{C}_2) = (\mathcal{A}_1, \mathcal{A}_2) + (\mathcal{B}_1, \mathcal{B}_2)$ vérifie l'égalité

$$\mathcal{C}_1 \cdot \varphi_n(x) = \mathcal{C}_2 \cdot (L_\mathcal{A} \cdot \varphi_n(x) + L_\mathcal{B} \cdot \varphi_n(x)) = \mathcal{C}_2 \cdot (L_{\mathcal{A}+\mathcal{B}} \cdot \varphi_n(x)).$$

La paire $(\mathcal{B}_1, \mathcal{B}_2)$ vérifie l'égalité

$$\mathcal{A}_2 \cdot L_\mathcal{B} \cdot (L_\mathcal{A}\varphi_n(x)) = \mathcal{A}_1 \cdot L_\mathcal{A}\varphi_n(x),$$

donc selon la proposition 4.32 page 71, la paire $(\mathcal{B}_1, \mathcal{B}_2)(\mathcal{A}_1, \mathcal{A}_2)$ est associée à l'opérateur $L_\mathcal{A} + L_\mathcal{B}$.

Comme $\varphi_\psi(L_\mathcal{A}) = A_2^{-1}A_1$ et $\varphi_\psi(L_\mathcal{B}) = B_2^{-1}B_1$, on a aussi $\varphi_\psi(L_\mathcal{A}+L_\mathcal{B}) = A_2^{-1}A_1 + B_2^{-1}B_1$ et $\varphi_\psi(L_\mathcal{B}L_\mathcal{A}) = B_2^{-1}B_1A_2^{-1}A_1$. Selon la proposition 8.24, la paire adjointe de $(\mathcal{A}_1, \mathcal{A}_2) + (\mathcal{B}_1, \mathcal{B}_2)$ est équivalente à la paire $(A_1, A_2) + (B_1, B_2)$ et selon la proposition 8.25, la paire adjointe de $(\mathcal{A}_1, \mathcal{A}_2)(\mathcal{B}_1, \mathcal{B}_2)$ est équivalente à $(B_1, B_2)(A_1, A_2)$. □

Démonstration du théorème 8.30. Selon le théorème 8.13, si $\psi_n(x)$ est une famille de fonctions presque-orthogonales, pour tout opérateur différentiel L, il existe une paire d'opérateurs $(\mathcal{A}_1, \mathcal{A}_2)$ tels que

$$\mathcal{A}_1 \cdot \psi_n(x) = \mathcal{A}_2 \cdot (L \cdot \psi_n(x)).$$

Selon le lemme 8.31, $\varphi_\psi(L)$ est la paire adjointe à $(\mathcal{A}_1, \mathcal{A}_2)$. Par la proposition 8.23 page 182, cette paire adjointe (A_1, A_2) vérifie

$$A_1 \cdot u = A_2 \cdot u^L,$$

ce qui conclut la preuve. □

4 Algorithmes

Dans cette section, plusieurs algorithmes sont décrits pour calculer l'image d'un opérateur différentiel par φ_ψ, chacun présentant des stratégies différentes. Dans un premier temps, un algorithme général est présenté pour calculer ce morphisme. Cet algorithme fonctionne avec toutes les fractions et les familles de fonctions presque-orthogonales. Pour une famille de fonctions donnée, on verra que des algorithmes plus performants peuvent être choisis. Ici la performance se juge sur deux critères qui sont d'ailleurs souvent incompatibles :

- La complexité de l'algorithme. Même si ici aucune complexité n'est estimée, en sachant que l'algorithme pour le calcul du lclm est beaucoup plus coûteux que la multiplication d'opérateurs, on va montrer comment on peut éviter de calculer des lclm et effectuer seulement des multiplications dans de nombreux cas.
- L'ordre de la récurrence calculée. On verra dans la section 5 comment réduire l'ordre des récurrences calculées en normalisant la fraction d'opérateurs de récurrence (sans pour autant garantir la minimalité du résultat).

4.1 Méthode de Horner pour un algorithme général

On déduit du théorème 8.30 page 185, un algorithme général pour calculer une paire de récurrence avec un opérateur différentiel en entrée. De cet algorithme se déduit le théorème central de cette section.

Théorème 8.32. *Soit L un opérateur différentiel avec des coefficients polynomiaux. Pour toute famille de fonctions presque-orthogonales $\psi_n(x)$, l'algorithme 8.4 page 189 calcule la fraction $\varphi_\psi(L)$.*

Avant d'énoncer cet algorithme, on reprend l'exemple du début de chapitre sur le développement de la fonction $-\frac{x}{2}\,\mathrm{J}_1(x)$ en série de Bessel. En utilisant le module Maple OreField défini dans le chapitre 4. On initialise d'abord les fractions $\mathcal{X}_2^{-1}\mathcal{X}_1$ et $\mathcal{D}_2^{-1}\mathcal{D}_1$ à l'aide des formules (8.1) et (8.2) :

```
> with(OreField);
> initOreField(n);
```

$$[\text{`*`}, \text{`+`}, \text{`-`}, \text{`.`}, \text{`^`}, adjointOfFrac, initOreField, normal]$$

$$UnivariateOreRing\,(n, shift)$$

```
> X := OreFrac([0,2*(n+1)], [1,0,1]);
> Dx :=OreFrac([1, 0, -1], [0,2]);
```

$$X := [(2n+2)\,Sn, 1 + Sn^2]$$

$$Dx := [1 - Sn^2, 2Sn]$$

On calcule ensuite les opérateurs adjoints des paires $(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{D}_1, \mathcal{D}_2)$.

```
> X := adjointOfFrac(X);
> Dx := adjointOfFrac(Dx);
```

$$X := [2n\,(n+2)\,Sn, n + 2 + nSn^2]$$

$$Dx := [1 - Sn^2, -2Sn]$$

On applique alors la méthode d'Horner à l'équation différentielle vérifiée par la fonction $-\frac{x}{2}\,\mathrm{J}_1(x)$. C'est-à-dire, on réécrit l'équation (8.5) page 169 comme :

$$\left(x\frac{d}{dx} - 1\right)\frac{d}{dx} + x.$$

La méthode d'Horner donne donc :

```
> Res := X;
> Res := Res.Dx-1;
> Res := Res.Dx+X;
```

$$[2n\,(n+2)\,Sn, n + 2 + nSn^2]$$

$$[n^2 + 3n + 2 - \left(n + n^2\right)Sn^2, -n - 2 - nSn^2]$$

$$[6 + n^2 + 5n + \left(2n^2 + 8n + 4\right)Sn^2 + (n+1)\,(n+2)\,Sn^4, (2n+6)\,Sn + (2n+2)\,Sn^3]$$

Le théorème 8.30 nous dit que le premier élément de cette dernière est un opérateur de récurrence annulant les coefficients de la série de Neumann de $x\,\mathrm{J}_1(x)$. En voici une forme lisible en équation de récurrence.

```
> ord := nops(op(1, Res))-1:
> add(op(i+1, op(1,Res))*u(n+i),i=0..ord):
> collect(eval(%, n=n-2), u, expand)=0;
```

$$\left(2n^2 - 4\right)u\,(n) + \left(n^2 + n\right)u\,(n-2) + \left(n^2 - n\right)u\,(n+2) = 0$$

On retrouve bien la récurrence d'ordre 4 qui annule les coefficients calculée au début du chapitre.

La méthode illustrée par cet exemple se généralise avec les algorithmes 8.3 et 8.4. L'algorithme 8.3 permet de calculer l'image d'un polynôme en remplaçant la variable x par la fraction $X_2^{-1}X_1$ et l'algorithme 8.4 permet de calculer le numérateur souhaité avec une équation différentielle en entrée.

Lemme 8.33. *Soit p un polynôme. Pour toute famille de fonctions presque-orthogonales $\psi_n(x)$, l'algorithme 8.3 calcule la fraction $\varphi_\psi(p)$.*

Entrée: $L := a_0 + a_1 x + \cdots + a_k x^k$ et une paire $\varphi_\psi(x) = X = (X_1, X_2)$.
Sortie: Une paire $(P_1, P_2) = \varphi_\psi(L)$
$P_1 := a_k$
$P_2 := 1$
pour tout i de $k-1$ à 1 **faire**
Calculer $(P_1, P_2) := (P_1, P_2)X + (a_i, 1)$
fin pour
renvoyer (P_1, P_2)

Algorithme 8.3: Calcul de l'image d'un polynôme par la méthode d'Horner.

Entrée: $L := \sum_{i=0}^{k} p_i(x)\partial_x^i$, une famille de fonctions presque-orthogonales $\psi_n(x)$ et les paires $(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{D}_1, \mathcal{D}_2)$ tels que $\mathcal{X}_1 \cdot \psi(x) = \mathcal{X}_2 \cdot x\psi(x)$ et $\mathcal{D}_1 \cdot \psi = \mathcal{D}_2 \cdot \psi'$
Sortie: Une paire $(P_1, P_2) = \varphi_\psi(L)$
Calculer $X = (X_1, X_2) := \left((\mathcal{X}_1^*)^{\mathcal{X}_2^*}, (\mathcal{X}_2^*)^{\mathcal{X}_1^*}\right)$ et $D = (D_1, D_2) = \left((\mathcal{D}_1^*)^{\mathcal{D}_2^*}, (\mathcal{D}_2^*)^{\mathcal{D}_1^*}\right)$
$(P_1, P_2) := \varphi_\psi(p_k(x))$ {Calcul de $\varphi(p_k(x))$ par l'algorithme 8.3}
pour tout i de $k-1$ à 0 **faire**
$(P_1, P_2) = (P_1, P_2)D + \varphi_\psi(p_i(x))$
{Calcul de $\varphi(p_i(x))$ par l'algorithme 8.3}
fin pour
renvoyer (P_1, P_2) ou si l'utilisateur le souhaite normal(P_1, P_2)

Algorithme 8.4: Calcul de l'image d'un opérateur différentiel par la méthode d'Horner.

Les preuves de ce lemme ainsi que du théorème 8.32, qui nous certifient la validité des algorithmes suivants, se déduisent immédiatement du théorème 8.30 page 8.30.

4.2 Algorithme sans lclm

L'algorithme 8.4 calcule de nombreux lclm. En effet, lors de chaque multiplication et addition de paires, un lclm est effectué. Dans de nombreux cas, on peut éviter de calculer ces lclm, ce qui rend l'algorithme plus efficace. Le but de cette section est de montrer comment les idées de Paszkowski [Pas75] se généralisent et permettent d'éviter ces lclm pour la plupart des familles de fonctions présentent dans le tableau 8.1.

Théorème 8.34. *Soit L un opérateur différentiel avec des coefficients polynomiaux. Pour toute famille de fonctions presque-orthogonales $\psi_n(x)$, l'algorithme 8.6 calcule la fraction $\varphi_\psi(L)$. Si de plus $\mathcal{X}_2$ et $\mathcal{D}_2$ sont des monômes, ou $\mathcal{X}_2$ et $\mathcal{D}_1$ sont des monômes ou $\mathcal{X}_1$ et $\mathcal{D}_2$ sont des monômes, alors l'algorithme n'effectue pas de* lclm.

Remarque 8.35. $\mathcal{X}_2$ et $\mathcal{D}_2$ sont des monômes pour les polynômes x^n et les polynômes de Hermite. $\mathcal{X}_2$ et $\mathcal{D}_1$ sont des monômes pour les polynômes de Jacobi (donc aussi pour les polynômes de Gegenbauer et Tchebychev) et les polynômes de Laguerre. $\mathcal{X}_1$ et $\mathcal{D}_2$ sont des

Entrée: $L := \sum_{i=0}^{k} \partial_x^i q_i(x)$, et les paires $X = (X_1, 1)$ et $D = (1, D_2)$
Sortie: Une paire $(P_1, P_2) = \sum_{i=0}^{k} D^i q_i(X)$
$P_1 := q_0(X)$ {Calcul de $\varphi(p_k(x))$ par l'algorithme 8.3 et X}
pour tout i de 1 à k **faire**
$P_1 := D_2 P_1 + q_i(X)$
fin pour
$P_2 := D_2^k$
renvoyer (P_1, P_2) ou si l'utilisateur le souhaite normal(P_1, P_2)

Algorithme 8.5: Méthode de Horner sans lclm.

monômes pour les fonctions de Bessel. Pour toutes ces familles de fonctions, on peut donc éviter de calculer des lclm.

Démonstration. La première remarque est que pour toute paire $(\mathcal{A}_1, \mathcal{A}_2)$, si $\mathcal{A}_i$ est un monôme alors A_i en est un aussi. En effet si $\mathcal{A}_i$ en est un $\mathcal{A}_i^*$ en est un aussi et donc pour tout opérateur de récurrence B, $(\mathcal{A}_i^*)^B$ est un monôme.

Si $\mathcal{X}_2$ et $\mathcal{D}_2$ sont des monômes, alors on utilise l'algorithme 8.4. Dans ce cas, lors des additions ou des multiplications, les dénominateurs sont des monômes. Les lclm effectués à ce moment sont donc triviaux. Si $\mathcal{D}_1$ ou $\mathcal{X}_2$ sont des monômes, on appelle alors l'algorithme 8.5 qui n'effectue aucun lclm. La correction de cet algorithme se voit en remarquant que la sortie de cet algorithme est la fraction

$$P_2^{-1} P_1 = D^k \left(\sum_{i=0}^{k} D^{-i} q_{k-i}(X) \right) = \sum_{i=0}^{k} D^{k-i} q_{k-i}(X), \tag{8.16}$$

qui est bien la fraction que l'on souhaite calculer. Si $\mathcal{X}_1$ et $\mathcal{D}_2$ sont des monômes, le changement entre les variables x et ∂_x permet de se ramener au cas où $\mathcal{X}_2$ et $\mathcal{D}_1$ sont des monômes. Pour les mêmes raisons que précédemment, le théorème est vérifié pour ce cas.

Le théorème est vrai pour les autres cas, car on utilise à nouveau l'algorithme 8.4. □

Exemple 8.36. On reprend l'exemple de la fonction $-\frac{x}{1} \mathrm{J}_1(x)$ que l'on veut développer en série de Bessel. On rentre alors dans le second cas de l'algorithme 8.5. Celui-ci nous indique donc d'inverser l'ordre entre l'opérateur différentiel et le polynôme. On obtient alors l'opérateur différentiel :

$$\partial_x(x^2 + 1) - x,$$

qui est déjà sous forme d'Horner. On définit les paires `X1` et `Dx1` comme les paires $(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{D}_1, \mathcal{D}_2)$:

```
> X1 := OreFrac([0,2*(n+1)], [1,0,1]);
> Dx1 :=OreFrac([1, 0, -1], [0,2]);
```

$$X1 := [(2n + 2)\, Sn, 1 + Sn^2]$$

$$Dx1 := [1 - Sn^2, 2Sn]$$

Entrée: $L := \sum_{i=0}^{k} p_i(x)\partial_x^i$, une famille de fonctions presque-orthogonales $\psi_n(x)$ et les paires $(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{D}_1, \mathcal{D}_2)$ tels que $\mathcal{X}_1 \cdot \psi(x) = \mathcal{X}_2 \cdot x\psi(x)$ et $\mathcal{D}_1 \cdot \psi = \mathcal{D}_2 \cdot \psi'$
Sortie: Une paire $(P_1, P_2) = \varphi_\psi(L)$
1: **si** $\mathcal{X}_2$ **et** $\mathcal{D}_1$ sont des monômes **alors**
2: $(X_1, X_2) := \left((X_1^*)^{X_2^*}, (X_2^*)^{X_1^*}\right)$ et $(D_1, D_2) = \left((D_1^*)^{D_2^*}, (D_2^*)^{D_1^*}\right)$
3: $X = (X_2^{-1}X_1, 1)$ et $D = (1, D_1^{-1}D_2)$ {on a donc ici des opérateurs de Laurent}
4: Calculer $q_i(x)$ tel que $L := \sum_{i=0}^{k} \partial_x^i q_i(x)$
5: {en utilisant l'algorithme 5.5 page 5.5}
6: Calculer (P_1, P_2) avec l'algorithme 8.5 avec en entrées $\sum_{i=0}^{k} \partial_x^i q_i(x)$, X, D
7: **sinon si** $\mathcal{X}_1$ **et** $\mathcal{D}_2$ sont des monômes **alors**
8: $(X_1, X_2) := \left((X_1^*)^{X_2^*}, (X_2^*)^{X_1^*}\right)$ et $(D_1, D_2) = \left((D_1^*)^{D_2^*}, (D_2^*)^{D_1^*}\right)$
9: Calculer $q_i(x)$ tel que $L := \sum_{i=0}^{k_2} x^i q_i(\partial_x)$ {k_2 est le maximum des degrés des p_i}
10: $X = (D_2^{-1}D_1, 1)$ et $D = (1, X_1^{-1}X_2)$ {on a donc ici des opérateurs de Laurent}
11: Calculer (P_1, P_2) avec l'algorithme 8.5 avec en entrées $\sum_{i=0}^{k_2} \partial_x^i q_i(x)$, X, D
12: **sinon**
13: Calculer $(P_1, P_2) = \varphi_\psi(L)$ par l'algorithme 8.4
14: **finsi**
15: **renvoyer** (P_1, P_2)

Algorithme 8.6: Calcul efficace d'une récurrence vérifiée par les coefficients d'une série de Fourier généralisée.

On définit ensuite les paires `X` et `Dx` comme paires adjointes de $(\mathcal{X}_1, \mathcal{X}_2)$ et $(\mathcal{D}_1, \mathcal{D}_2)$.

```
> Dx := adjointOfFrac(X1);
> X := adjointOfFrac(Dx1);
```

$$Dx := [2n\,(n+2)\,Sn, n+2+nSn^2]$$

$$X := [1-Sn^2, -2Sn]$$

On applique ensuite la méthode de Horner comme dans l'algorithme 8.5, c'est-à-dire on multiplie par Dx^{-1} plutôt que par Dx, on remarque que de cette façon les dénominateurs sont des monômes et que donc les additions et multiplications sont triviales.

```
> Dx := adjointOfFrac(X1);
> X := adjointOfFrac(Dx1);
```

$$Dx := [2n\,(n+2)\,Sn, n+2+nSn^2]$$

$$X := [1-Sn^2, -2Sn]$$

```
> Res := X^2+1;
> Dx^(-1).X;
> Res := Res-%;
```

$$Res := [-1-2Sn^2-Sn^4, -4Sn^2]$$

$$[-n-3+2Sn^2+(n+1)\,Sn^4, \left(4n^2+16n+12\right)Sn^2]$$

$$Res := [n^2+5n+6+\left(2n^2+8n+4\right)Sn^2+\left(n^2+3n+2\right)Sn^4, \left(4n^2+16n+12\right)Sn^2]$$

Le numérateur obtenu est l'opérateur de récurrence d'ordre 4 dont on déduit la récurrence

$$(n^2+5n+5)u_n+(2n^2+8n+4)u_{n+2}+(n^2+3n+2)u_{n+4}=0.$$

On obtient bien de cette façon une récurrence qui annule les coefficients. On remarque que cette récurrence est différente de celle obtenue par l'autre algorithme. L'algorithme ne nous renvoie pas non plus le dénominateur de $\varphi_\psi(L)$. Mais on connaît ce dénominateur puisqu'il s'agit de Dx, on peut ainsi montrer l'équivalence entre les fractions calculées ici et dans la section 4.1.

5 Abaissement de l'ordre

5.1 Par le gcld

On a proposé plusieurs algorithmes pour calculer la récurrence. Dans ma présentation, ce qui les différentiait était la complexité des calculs effectués. Je n'ai pas comparé les résultats retournés. Un des critères de comparaison les plus simples est l'ordre de la récurrence. Pour l'utilisation d'une récurrence, il est souvent plus pratique d'avoir un ordre petit. Par exemple, pour dérouler une récurrence, on a de cette façon moins de conditions initiales à déterminer, et moins de calculs à effectuer (les degrés des coefficients polynomiaux rentrent aussi en compte).

Un premier calcul simple permet de minimaliser l'ordre de la récurrence obtenue par les algorithmes 8.4 et 8.6. Ces algorithmes calculent un numérateur de la fraction $\varphi_\psi(L)$, pour diminuer l'ordre de ce numérateur, on peut normaliser la fraction par l'algorithme 4.7 page 82. Ceci revient à calculer un gcld entre le numérateur et le dénominateur.

Exemple 8.37. Si on veut retrouver la récurrence d'ordre 2 dans l'exemple de la section 4.1, il suffit de normaliser cette fraction avec la procédure normal de OreField.

```
> ord := nops(op(1, normal(Res)))-1:
> add(op(i+1, op(1, Res))*u(n+i), i=0..ord):
> collect(eval(%, n=n-ord/2), u, expand);
```

$$(n+1)\,u\,(n-1)+(n-1)\,u\,(n+1)$$

On retrouve alors la récurrence d'ordre 2 déjà présentée dans l'introduction.

Exemple 8.38. On reprend maintenant le résultat obtenu dans l'exemple 8.36. On a vu que le dénominateur de la fraction $\varphi_\psi(L)$ est Dx, on peut alors normaliser le résutat pour obtenir :

```
> normal(Dx.Res);
```

$$[n+2+nSn^2, 2Sn]$$

On obtient ainsi à nouveau la récurrence d'ordre 2.

Malheureusement cette baisse de l'ordre par le calcul de gcld ne nous garantit pas la minimalité de l'opérateur de récurrence. Dans la suite, on verra d'autres exemples qui montrent que la quête de cette minimialité est un problème difficile.

5.2 Par la méthode Rebillard-Zakrajšek

Rebillard et Zakrajšek [RZ06] ont présenté une méthode pour calculer des opérateurs de récurrence. Comme déjà expliqué dans l'introduction, leur méthode est moins générale que la nôtre car elle ne concerne que les séries dont la base est une famille de polynômes hypergéométriques. Cependant, ils donnent une méthode pour abaisser l'ordre de la récurrence. Dans certains cas leur méthode permet d'avoir des récurrences d'ordre plus faible que celles calculées par la méthode proposée dans la section 5.

Avant d'expliquer leur méthode, regardons d'abord leur contexte et les opérateurs de récurrence qu'ils calculent.

Les polynômes hypergéométriques sont une généralisation des polynômes orthogonaux classiques vérifiant beaucoup de propriétés. On peut définir cette famille à partir de la proposition suivante.

Proposition 8.39. *Soient σ et τ deux polynômes de degré respectivement plus petit que 2 et 1 tels que la fonction qui à tout entier positif n associe*

$$\lambda_n = -n\left(\tau' + (n-1)\frac{\sigma''}{2}\right)$$

est injective.

L'équation différentielle hypergéométrique

$$\sigma(x)y''(x) + \tau(x)y'(x) + \lambda_n y(x) = 0, \tag{8.17}$$

a comme solution un polynôme P_n de degré n. La famille de polynômes P_n est appelée famille de polynômes hypergéométriques.

Les propriétés vérifiées par ces familles de polynômes sont traitées dans [NU88]. Pour notre propos deux propriétés majeures vérifiées par ces polynômes nous intéressent. Ils vérifient les équations (mult_x) et (diff_x) page 173 mais dans ces équations $\mathcal{X}_2$ et $\mathcal{D}_1$ sont des monômes de la forme

$$\begin{aligned}\mathcal{X}_2 &= x_1(n)S_n + x_0(n) + x{-}1(n)S_n^{-1},\\ \mathcal{D}_1 &= \rho_1(n)S_n + \rho_1(n) + \rho_{-1}(n)S_n^{-1},\end{aligned}$$

où les coefficients x_i et ρ_i sont des fractions rationnelles déterminées par les valeurs de σ, τ et des conditions initiales de l'équation différentielle. On a déjà vu avec l'algorithme 8.5 que dans ce cas, on sait calculer un opérateur de récurrence sans effectuer de lclm. Comme dans cet algorithme, on définit les polynômes de Laurent (après calcul des paires adjointes)

$$X = X_2^{-1}X_1 \quad \text{et} \quad I = D_1^{-1}D_2.$$

Pour un opérateur différentiel écrit comme $L = \sum_{i=0}^{k} \partial_x^i p_i(x)$ et une famille de polynômes ψ_n hypergéométriques, on a, comme dans l'équation (8.16) page 190,

$$\varphi_\psi(L) = I^{-k} \sum_{i=0}^{k} I^{k-i} p_i(X).$$

Le numérateur, donc la récurrence que l'on souhaite calculer, est alors $\sum_{i=0}^{k} I^{k-i} p_i(X)$.

La remarque de Rebillard et Zakrajšek est que l'on peut dans certains cas encore abaisser l'ordre par rapport au résultat obtenu dans la section 5.1 en normalisant l'opérateur

$$I^{-k}\varphi_\psi(L) = \varphi_\psi(\partial_x^k L),$$

en choisissant un bon k. Le numérateur de la fraction calculée est une récurrence satisfaisante car cette fraction est l'image de l'opérateur $\partial_x^k L$ qui annule les solutions de L.

Un des objectifs de leur article est d'obtenir l'entier k optimal. Optimal veut dire ici que c'est le plus petit k tel que pour tout $j > k$, le numérateur de la fraction normalisée $I^{-j}\varphi_\psi(L)$ est d'ordre supérieur ou égal à l'ordre du numérateur de $I^{-k}\varphi_\psi(L)$. (Comme ils n'utilisent pas les fractions d'opérateurs, la minimalité de cet entier est primordiale.)

Rebillard et Zakrajšek proposent une méthode pour calculer cet entier. Cette méthode est très technique, et dans notre contexte, on peut se contenter d'une borne supérieure. En effet comme I est l'inverse d'un opérateur, on n'augmente pas l'ordre du numérateur d'une fraction en multipliant celle-ci par I à gauche. Le calcul de cette borne est résumé dans la proposition suivante.

Proposition 8.40 ([RZ06])**.** *Soit ψ_n une famille de polynômes hypergéométriques solutions de l'équation (8.17) et un opérateur différentiel $L = \sum_{i=0}^{r} p_i(x)\partial_x^i$. Soit $\sigma(x-\xi_1)(x-\xi_2)$ le polynôme de degré 2 défini comme dans l'équation (8.17). Soit ℓ le plus grand entier tel que $(x-\xi_1)^\ell$ ou $(x-\xi_2)^\ell$ divise p_r, et soit $k := \ell$ si $\ell > r$ et $2\ell - r$ sinon.*

Alors pour tout i, l'ordre du numérateur de $\varphi_\psi(\partial^i L)$ est supérieur ou égal à l'ordre du numérateur de $\varphi_\psi(\partial^k L)$.

On reprend un exemple de [RZ06] pour illustrer cette méthode.

Exemple 8.41. Soit $\psi_n = T_n$, la famille des polynômes de Tchebychev. Le polynôme σ est donc $(x-1)(x+1)$. Soit l'équation différentielle

$$L = (x+1)^2 y'' - (x+1)y' + (x+7/4)y = 0. \tag{8.18}$$

Par les algorithmes de ce chapitre, la récurrence obtenue à partir de la fraction normalisée de $\varphi_\psi(L)$ est

```
> L =(x+1)^2*diff(y(x),x,x)-(x+1)*diff(y(x), x)+(x+7/4)*y(x);
> diffeqToGFSRec(L, y(x), u(n), functions=ChebyshevT(n,x),  normalize=true);
```

$$(4n+10)\,u(n) + \left(8n^3 + 20n^2 - 2n - 1\right)u(n+1) + 4\,(n+2)\left(4n^2 + 16n + 5\right)u(n+2)$$
$$+ \left(185 + 222n + 76n^2 + 8n^3\right)u(n+3) + (4n+6)\,u(n+4)$$

qui est une récurrence d'ordre 4.

Ici $(x+1)^2$ divise le coefficient de tête. Selon la proposition 8.40 on obtient l'ordre optimal en utilisant l'équation différentielle $\partial_x^2 L$. C'est ce qu'effectue l'instruction suivante.

```
> diffeqToGFSRec(diff(L,x, x), y(x), u(n),
> functions=ChebyshevT(n,x),  normalize=true);
```

$$
\begin{aligned}
&2\,(n+1)\,(n+2)\left(2\,n^3+21\,n^2+73\,n+84\right)u\,(n+1)\\
&+(n+1)\,(n+2)\left(8\,n^5+100\,n^4+462\,n^3+941\,n^2+745\,n+84\right)u\,(n+2)\\
&+(2\,n+3)\,(n+1)\,(n+2)\left(4\,n^3+48\,n^2+189\,n+244\right)\,(n+3)\,u\,(n+3)\\
&+2\,(2\,n+3)\,(n+1)\,(n+2)\left(n^2+7\,n+12\right)u\,(n+4)
\end{aligned}
$$

On obtient alors une récurrence d'ordre 3 (il n'y a pas de terme en $u(n)$).

5.3 Un contre-exemple dû à Lewanowicz

Lewanowicz a beaucoup travaillé [Lew76, Lew79, Lew83, Lew85, Lew91, Lew92] sur le calcul de récurrence lorsque les familles de fonctions sont des familles de polynômes de Jacobi. Dans ces différents articles il s'intéresse à minimaliser l'ordre de l'opérateur renvoyé. Une de ses contributions est d'avoir donné des algorithmes, lorsque ψ_n est une famille de polynômes de Jacobi, qui renvoient le même opérateur que celui renvoyé par les algorithmes de ce chapitre après normalisation de la fraction. Cet opérateur est calculé sans fractions d'opérateurs de récurrence et lclm avec des méthodes *ad-hoc*, que l'on peut voir comme une généralisation de celle présentée dans l'algorithme 5.1 page 89.

Dans un de ses articles [Lew91], Lewanowicz donne l'exemple d'une équation différentielle avec laquelle les techniques, décrites ci-dessus, pour abaisser l'ordre échouent.

Exemple 8.42. On souhaite développer en série de Tchebychev la fonction $x\exp(x)$. Par les résultats du chapitre 6, on sait que cette fonction se développe comme

$$x\exp(x) = {\sum_{n\leqslant 0}}' \left(\mathrm{I}_{n+1}(1) + \mathrm{I}_{n-1}(1)\right) T_n(x).$$

Par les techniques utilisées jusqu'à présent, on calcule la récurrence vérifiée par les coefficients de Tchebychev à partir de l'équation différentielle

$$deq := xy(x)' - (x+1)y = 0.$$

```
> diffeqToGFSRec(deq, y(x), u(n), functions=ChebyshevT(n,x), normalize=true);
```

$$-u\,(n) + 2\,nu\,(n+1) + (2\,n+8)\,u\,(n+3) + u\,(n+4)$$

Cette récurrence est d'ordre 4, alors que l'on sait par la formule des formes closes des coefficients qu'une récurrence d'ordre 2 existe. L'astuce de Rebillard et Zakrajšek ne

s'applique pas ici puisque le terme dominant est x. Lewanowicz propose d'utiliser l'équation différentielle

$$deq2 = (-1+x^2)y''' - (x^2-3x-1)y'' - (4x-1)y' - 3y(x) = 0,$$

qui annule aussi $x\exp(x)$. On obtient alors la récurrence

```
> diffeqToGFSRec(deq2, y(x), u(n), functions=ChebyshevT(n,x), normalize=true);
```

$$\left(-n^2-3\,n-3\right)u\left(n\right)+\left(2\,n^3+6\,n^2+6\,n+2\right)u\left(n+1\right)+\left(n^2+n+1\right)u\left(n+2\right)$$

Paszkowski a travaillé aussi sur cet exemple et pu obtenir la récurrence d'ordre 2 en divisant par x^2 l'équation différentielle.

Lewanowicz n'a proposé aucune méthode *automatique* pour obtenir une équation différentielle plus adaptée pour minimaliser l'ordre. Obtenir l'ordre minimal est donc encore aujourd'hui un problème ouvert.

6 Conclusion

Les résultats de ce chapitre montrent comment calculer les coefficients d'une large classe de séries de Fourier généralisées en utilisant un outil original : les fractions d'opérateurs de récurrence.

On a vu que ces fractions permettaient aussi d'abaisser l'ordre des récurrences par normalisation. La dernière section montre que les fractions d'opérateurs n'apportent cependant pas la solution au problème du calcul de la récurrence d'ordre minimal.

Dans la littérature, certaines séries de Fourier généralisées sont développées dans une base de fonctions qui n'est pas presque-orthogonale. On peut penser notamment au séries de Fourier ou encore à la série de l'exemple suivant. Les coefficients de ces séries peuvent dans certains cas être quand même solutions de récurrence. Dans ce cas, l'utilisation des fraction d'opérateurs de récurrence utilisées autrement permet parfois encore de retrouver les récurrences.

Exemple 8.43. Le développement en série suivant [Nat10, http://dlmf.nist.gov/7.6.E9]

$$\operatorname{erf}(ax) = 2e^{(\frac{1}{2}-a^2)x^2}\sum_{n=0}^{\infty} T_n(a) I_{n+\frac{1}{2}}\left(\frac{x^2}{2}\right), \tag{8.19}$$

ne rentre pas dans notre cadre, alors que les coefficient $T_n(a)$ vérifient une récurrence.

Appendix A

The Dynamic Dictionary of Mathematical Functions (DDMF)

Abstract

We describe the main features of the Dynamic Dictionary of Mathematical Functions (version 1.5). It is a website consisting of interactive tables of mathematical formulas on elementary and special functions. The formulas are automatically generated by computer algebra routines. The user can ask for more terms of the expansions, more digits of the numerical values, or proofs of some of the formulas.

This work is joint with Frédéric Chyzak, Alexis Darrasse, Stefan Gerhold, Marc Mezzarobba and Bruno Salvy.

Sommaire

1 Motivation

Dictionaries of mathematical functions are commonly used by scientists and engineers. Some of the most famous ones are Abramowitz & Stegun's *Handbook of Mathematical Functions* [AS64]; the Bateman project *Higher Transcendental Functions* [Erd81]; Gradshtein & Ryzhik's *Table of Integrals, Series, and Products* [GR96]; and the multivolume *Integrals and Series* by Prudnikov, Brychkov, and Marichev [PBM86]. These dictionaries gather formulas such as differential equations, definite and indefinite integrals, inequalities, recurrence relations, power series, asymptotic expansions, approximations, and sometimes graphs and numerical tables, for a large set of functions. They have been prepared by specialists of these functions and carefully checked and proofread. Their success is attested to by the hundreds of thousands of citations they have received [BL01].

The first editions of those books were published between 60 and 30 years ago.

Since then, the advent of the World Wide Web has changed the way people now look for information. Aware of this change, the NIST has published a new version of [AS64] in 2010, called the *NIST Handbook of Mathematical Functions* [OLBC10] together with a web site, the NIST Digital Library of Mathematical Functions. This site offers navigation in the formulas, active links, export to various formats, and a search engine.

In parallel, computer algebra systems have grown into huge libraries of mathematical algorithms. While the implementation of mathematical functions in these systems is often basically a coding of formulas from the dictionaries mentioned above, the algorithms have matured to a level where many of those formulas can actually be computed automatically.

The aim of the DDMF is to combine recent algorithms in computer algebra together with web interaction into a dictionary of mathematical functions that is automatically generated, easily navigable with export to various formats, and interactive [1]. Interactivity means that formulas or graphics can be adapted to the user's needs; that arbitrary precision can be given on demand; and that proofs can be displayed if desired.

At this stage, the reader is encouraged to have a look at the DDMF at the following url

http://ddmf.msr-inria.inria.fr

A typical page is presented in Figure A.1.

The rest of this article presents the ideas underlying our current version (1.5), first from the point of view of the document system and then from the computer algebra viewpoint.

2 Dynamic Mathematics on the Web

The language we use to produce the DDMF is called DynaMoW for *Dynamic Mathematics on the Web*. The main principle on which it is based is captured by the following statement:

> *The document being generated by the symbolic computation engine is an object of the language.*

1. An ancestor of these ideas without interactivity was presented in [MS03].

url

The Special Function $\mathrm{erf}(x)$

change rendering

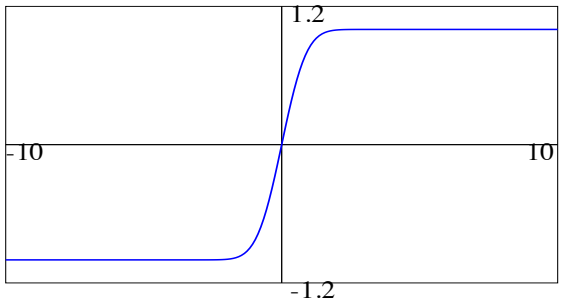
1. Differential equation

The function $\mathrm{erf}(x)$ satisfies

$$2\left(\frac{d}{dx}y(x)\right)x+\frac{d^2}{dx^2}y(x)=0$$

with initial values $y(0)=0$, $(y')(0)=2\frac{1}{\sqrt{\pi}}$.

2. Plot of $\mathrm{erf}(x)$



min = -10 max = 10 Submit

3. Numerical Evaluation

$$\mathrm{erf}(1/4+1/4\,i)\approx 0.29339518+0.26991350\,i$$

(Below, `path` may be either a point z or a broken-line path $[z_1,z_2,\ldots,z_n]$ along which to perform analytic continuation of the solution of the defining differential equation. Each z_k should be of the form `x + y*i` .)

path = 1/4+1/4 precision = 8 Submit

4. Symmetry

The function $\mathrm{erf}(x)$ is odd:

$$\mathrm{erf}(x)=-\mathrm{erf}(-x)$$

for all complex numbers x .

See the Proof That the Function $\mathrm{erf}(x)$ is Odd.

5. Taylor expansion of $\mathrm{erf}(x)$ at 0

- Expansion of erf at 0 :

$$\mathrm{erf}(x)=\sum_{n=0}^{\infty}2\,\frac{(-1)^n x^{2n+1}}{\sqrt{\pi}(2n+1)\,n!}$$

- See the recurrence relation for the coefficients of the Taylor expansion.

Figure A.1: The beginning of the page of the DDMF on the error function

Thus, instead of using a fixed template whose fields are filled in during a computation, the structure of the document itself depends on the results of intermediate computations. For instance, the number of subsections on asymptotic expansions is a result of computing the singularities of the function; the section on symmetries only occurs if the function has been proved even or odd.

DynaMoW is a layer between a symbolic computation engine [2] and a web server. It lets one mix symbolic code together with pieces of documents in a single source code in a natural way. This provides an easy way to showcase computer algebra algorithms to users who do not know the syntax of a computer algebra system: all they need is a web browser; DynaMoW has been designed to be produce pages compatible with the most popular ones.

Moreover, once the document becomes part of the computation, new possibilities arise. For instance, being able to glue together pieces of documents during the computation lets us turn a trace of the computation into a detailed mathematical proof of its result. (See, for instance, the proof of the recurrence formula for the coefficients of the Taylor series of the Airy Ai function.) This answers a frequent request of users of computer algebra systems, who want to be able to understand where the results come from and how they can check or trust them. Traces are not the only type of proof that can be generated. For instance, we may present a simple proof for the solution of a recurrence once the solution has been found, instead of retracing its computation.

The implementation of the DynaMoW language itself is work in progress and a stable version will be described in due course. We believe that this language will be of interest outside of the DDMF. For instance, we have also used it with success for an encyclopedia of combinatorial structures.

3 Computer Algebra Algorithms

From the computer algebra point of view, what is a good definition of a mathematical function such that all the desired formulas can be computed algorithmically? Our choice is to concentrate on

Functions given as solutions of linear differential equations or linear recurrences.

Our basic data-structure consists of these equations and their initial conditions. In the example of Fig. A.1, this is the content of Section 1.

This data-structure has become common in computer algebra, starting with works of Stanley [Sta80], Lipschitz [Lip89], Zeilberger [Zei90] and more recently Chyzak, Salvy *et alii* [Chy98, Chy00, CKS09]. In particular, we rely on the Maple `gfun` package [SZ94] for many of our computations.

Given this data structure, we have used or developed algorithms to compute many relevant properties of mathematical functions. For instance, Section 3 of our example offers numerical approximations of guaranteed quality in good complexity (see [Mez10] for the algorithm). Such approximations can be used to produce graphs as in Section 2 of the

2. Currently we use Maple, but DynaMoW is designed so that other systems can be used as well.

example. Section 5 is based on recurrences for the Taylor coefficients that are obtained from the differential equations. When they exist, closed-form hypergeometric solutions of these recurrences can be computed [Pet92]. In all cases, the rest of that Section 5 (beyond the part that is visible in Fig. 1) gives the first terms of these expansions and bounds on tails of power series [MS10]. The same computations are performed at each singularity including infinity. Further results include Chebyshev expansions [BS09] and differential equations for the Laplace transform (Sections 7 and 8).

Future Work

Some of our next steps include these tasks: automatic handling of families of functions or functions with parameters, like the Bessel functions, either by letting the user choose values for the parameters, or by performing an automatic discussion according to the possible range of values; automatic generation of good numerical code at fixed precision; more integral transforms; expansions on other bases; information on the zeros of the functions; handling of branch-cuts; and support for user-defined functions.

Appendix B

Quasi-optimal multiplication of linear differential operators

Abstract

We show that linear differential operators with polynomial coefficients can be multiplied in quasi-optimal time. This answers an open question raised by van der Hoeven.

This work is joint with Alin Bostan and Joris van der Hoeven.

Sommaire

1 Introduction

The product of *polynomials* and the product of *matrices* are two of the most basic operations in mathematics; the study of their computational complexity is central in computer science. In this paper, we will be interested in the computational complexity of multiplying two *linear differential operators*. These algebraic objects encode linear differential equations, and form a non-commutative ring that shares many properties with the commutative ring of usual polynomials [Ore32, Ore33]. The structural analogy between polynomials and linear differential equations was discovered long ago by Libri and Brassinne [Lib33, Bra64, Dem83]. Yet, the algorithmic study of linear differential operators is currently much less advanced than in the polynomial case: the complexity of multiplication has been addressed only recently [vdH02, BCLR08], but not completely solved. The aim of the present work is to make a step towards filling this gap, and to solve an open question raised in [vdH02].

Let $\mathbb{K}$ be an effective field. That is, we assume data structures for representing the elements of $\mathbb{K}$ and algorithms for performing the field operations. The aim of *algebraic complexity theory* is to study the cost of basic or more complex algebraic operations over $\mathbb{K}$ (such as the cost of computing the greatest common divisor of two polynomials of degrees less than d in $\mathbb{K}[x]$, or the cost of Gaussian elimination on an $r \times r$ matrix in $\mathbb{K}^{r\times r}$) in terms of the number of operations in $\mathbb{K}$. The *algebraic complexity* usually does not coincide with the *bit complexity*, which also takes into account the potential growth of the actual coefficients in $\mathbb{K}$. Nevertheless, understanding the algebraic complexity usually constitutes a first useful step towards understanding the bit complexity. Of course, in the special, very important, case when the field $\mathbb{K}$ is finite, both complexities coincide up to a constant factor.

The complexities of operations in the rings $\mathbb{K}[x]$ and $\mathbb{K}^{r\times r}$ have been intensively studied during the last decades. It is well established that polynomial multiplication is a *commutative complexity yardstick*, while matrix multiplication is a *non-commutative complexity yardstick*, in the sense that the complexity of operations in $\mathbb{K}[x]$ (resp. in $\mathbb{K}^{r\times r}$) can generally be expressed in terms of the cost of multiplication in $\mathbb{K}[x]$ (resp. in $\mathbb{K}^{r\times r}$), and for most of them, in a quasi-linear way [AHU74, BP94a, BCS97, Pan01, vzGG03].

Therefore, understanding the algebraic complexity of multiplication in $\mathbb{K}[x]$ and $\mathbb{K}^{r\times r}$ is a fundamental question. It is well known that two polynomials of degrees $< d$ can be multiplied in time $\mathsf{M}(d) = \mathcal{O}(d \log d \log\log d)$ using algorithms based on the Fast Fourier Transform (FFT) [CT65, SS71, CK91], and two $r \times r$ matrices in $\mathbb{K}^{r\times r}$ can be multiplied in time $\mathcal{O}(r^\omega)$, with $2 \leqslant \omega \leqslant 3$ [Str69, Pan84, CW90]. The current tightest upper bound, due to Vassilevska Williams [Vas12], is $\omega < 2.3727$, following work of Coppersmith and Winograd [CW90] and Stothers [Sto10]. Finding the best upper bound on ω is one of the most important open problems in algebraic complexity theory.

In a similar vein, our thesis is that understanding the algebraic complexity of multiplication of linear differential operators is a very important question, since the complexity of more involved, higher-level, operations on linear differential operators can be reduced to

that of multiplication [vdH11].

Let $\mathbb{K}[x,\partial]$ denote the associative algebra $\mathbb{K}\langle x,\partial;\partial x = x\partial + 1\rangle$ of linear differential operators in $\partial = \frac{d}{dx}$ with polynomial coefficients in x. Any element L of $\mathbb{K}[x,\partial]$ can be written as a finite sum $\sum_i L_i(x)\partial^i$ for uniquely determined polynomials L_i in $\mathbb{K}[x]$. We say that L has bidegree less than (d,r) in (x,∂) if L has degree less than r in ∂, and if all L_i's have degrees less than d in x. The degree in ∂ of L is usually called the *order* of L.

The main difference with the commutative ring $\mathbb{K}[x,y]$ of bivariate polynomials is the commutation rule $\partial x = x\partial + 1$ that simply encodes, in operator notation, Leibniz's differentiation rule $\frac{d}{dx}(xf) = x\frac{d}{dx}(f) + f$. This slight difference between $\mathbb{K}[x,\partial]$ and $\mathbb{K}[x,y]$ has a considerable impact on the complexity level. On the one hand, it is classical that multiplication in $\mathbb{K}[x,y]$ can be reduced to that of polynomials in $\mathbb{K}[x]$, due to a technique commonly called *Kronecker's trick* [Moe76, vzGG03]. As a consequence, any two polynomials of degrees less than d in x, and less than r in y, can be multiplied in *quasi-optimal time* $\mathcal{O}(\mathsf{M}(dr))$. On the other hand, it was shown by van der Hoeven [vdH02] that, if the base field $\mathbb{K}$ has characteristic zero, then the product of two elements from $\mathbb{K}[x,\partial]$ of bidegree less than (n,n) can be computed in time $\mathcal{O}(n^\omega)$. Moreover, it has been proved in [BCLR08] that conversely, multiplication in $\mathbb{K}^{n\times n}$ can be reduced to a constant number of multiplications in $\mathbb{K}[x,\partial]$, in bidegree less than (n,n). In other words, multiplying operators of *well-balanced bidegree* is computationally equivalent to matrix multiplication.

However, contrary to the commutative case, higher-level operations in $\mathbb{K}[x,\partial]$, such as *left common least multiple* (LCLM) and *greatest common right divisor* (GCRD), do not preserve well-balanced bidegrees [Gri90, BCLS12]. For instance, the LCLM of two operators of bidegrees less than (n,n) is of bidegree less than $(2n(n+1),2n) = \mathcal{O}(n^2,n)$, and this bound is generically reached. This is a typical phenomenon: operators obtained from computations in $\mathbb{K}[x,\partial]$ tend to have much larger degrees in x than in ∂.

In the general case of operators with possibly unbalanced degrees d in x and r in ∂, the naive algorithm has cost $\mathcal{O}(d^2r^2\min(d,r))$; a better algorithm, commonly attributed to Takayama, has complexity $\tilde{\mathcal{O}}(dr\min(d,r))$. We refer to [BCLR08, §2] for a review of these algorithms. When $r \leqslant d \leqslant r^{4-\omega}$ in ∂, the best current upper bound for multiplication is $\mathcal{O}(r^{\omega-2}d^2)$ [vdH02, vdH11]. It was asked by van der Hoeven [vdH02, §6] whether this complexity could be lowered to $\tilde{\mathcal{O}}(r^{\omega-1}d)$. Here, and hereafter, the soft-O notation $\tilde{\mathcal{O}}(\,)$ indicates that polylogarithmic factors in d and in r are neglected. The purpose of the present work is to provide a positive answer to this open question. Our main result is encapsulated in the following theorem.

Theorem 1. *Let $\mathbb{K}$ be an effective field of characteristic zero. Operators in $\mathbb{K}[x,\partial]$ of bidegree less than (d,r) in (x,∂) can be multiplied using*

$$\tilde{\mathcal{O}}(dr\min(d,r)^{\omega-2})$$

operations in $\mathbb{K}$.

In the important case $d \geqslant r$, this complexity reads $\tilde{\mathcal{O}}(dr^{\omega-1})$. This is quasi-linear (thus quasi-optimal) with respect to d. Moreover, by the equivalence result from [BCLR08, §3], the exponent of r is also the best possible. Besides, under the (plausible, still conjectural) assumption that $\omega = 2$, the complexity in Theorem 1 is almost linear with respect to the output size. For $r = 1$ we retrieve the fact that multiplication in $\mathbb{K}[x]$ in degree $< d$ can be done in quasi-linear time $\tilde{\mathcal{O}}(d)$; from this perspective, the result of Theorem 1 can be seen as a generalization of the fast multiplication for usual polynomials.

In an expanded version [BBvdH] of this extended abstract, we will show that analogues of Theorem 1 also hold for other types of *skew polynomials*. More precisely, we will deal with the cases when the skew indeterminate $\partial : f(x) \mapsto f'(x)$ is replaced by the Euler derivative $\delta : f(x) \mapsto xf'(x)$, or a shift operator $\sigma^c : f(x) \mapsto f(x+c)$, or a dilatation $\chi_q : f(x) \mapsto f(qx)$. In [BBvdH], we will also prove refined versions of Theorem 1 and complexity bounds for several other interesting operations on skew polynomials.

Main ideas. The fastest known algorithms for multiplication of usual polynomials in $\mathbb{K}[x]$ rely on an evaluation-interpolation strategy at special points in the base field $\mathbb{K}$ [CT65, SS71, CK91]. This reduces polynomial multiplication to the "inner product" in $\mathbb{K}$. We adapt this strategy to the case of linear differential operators in $\mathbb{K}[x, \partial]$: the evaluation "points" are *exponential polynomials* of the form $x^n e^{\alpha x}$ on which differential operators act nicely. With this choice, the evaluation and interpolation of operators is encoded by *Hermite evaluation and interpolation* for usual polynomials (generalizing the classical Lagrange interpolation), for which quasi-optimal algorithms exist. For operators of bidegree less than (d, r) in (x, ∂), with $r \geqslant d$, we use $p = \mathcal{O}(r/d)$ evaluation points, and encode the inner multiplication step by p matrix multiplications in size r. All in all, this gives an FFT-type multiplication algorithm for differential operators of complexity $\tilde{\mathcal{O}}(d^{\omega-1}r)$. Finally, we reduce the case $r \leqslant d$ to the case $r \geqslant d$. To do this efficiently, we design a fast algorithm for the computation of the so-called *reflection* of a differential operator, a useful ring morphism that swaps the indeterminates x and ∂, and whose effect is exchanging orders and degrees.

2 Preliminaries

Throughout the paper, $\mathbb{K}[x]_d$ will denote the set of polynomials of degree less than d with coefficients in the field $\mathbb{K}$, and $\mathbb{K}[x, \partial]_{d,r}$ will denote the set of linear differential operators in $\mathbb{K}[x, \partial]$ with degree less than r in ∂, and polynomial coefficients in $\mathbb{K}[x]_d$.

The cost of our algorithms will be measured by the number of field operations in $\mathbb{K}$ they use. We recall that polynomials in $\mathbb{K}[x]_d$ can be multiplied within $\mathsf{M}(d) = \mathcal{O}(d \log(d) \log\log(d)) = \tilde{\mathcal{O}}(d)$ operations in $\mathbb{K}$, using the FFT-based algorithms in [SS71, CK91], and that ω denotes a feasible exponent for matrix multiplication over $\mathbb{K}$, that is, a real constant $2 \leqslant \omega \leqslant 3$, such that two $r \times r$ matrices with coefficients in $\mathbb{K}$ can be multiplied in time $\mathcal{O}(r^\omega)$.

Most basic polynomial operations in $\mathbb{K}[x]_d$ (division, Taylor shift, extended gcd, mul-

tipoint evaluation, interpolation, etc.) have cost $\tilde{\mathcal{O}}(d)$ [AHU74, BP94a, BCS97, Pan01, vzGG03]. Our algorithms will make a crucial use of the following result due to Chin [Chi76], see also [OS00] for a formulation in terms of structured matrices.

Theorem 2 (Fast Hermite evaluation and interpolation)**.** *Let $c_0, \dots, c_{k-1}$ be k integers, $d = \sum_i c_i$, and let $\mathbb{K}$ be an effective field of characteristic zero. Given k mutually distinct points $\alpha_0, \dots, \alpha_{k-1}$ in $\mathbb{K}$ and a polynomial $P \in \mathbb{K}[x]_d$, one can compute the vector of d values*

$$\begin{aligned}\mathcal{H} \;=\; & (P(\alpha_0), P'(\alpha_0), \dots, P^{(c_0-1)}(\alpha_0), \dots\dots, \\ & P(\alpha_{k-1}), P'(\alpha_{k-1}), \dots, P^{(c_{k-1}-1)}(\alpha_{k-1}))\end{aligned}$$

in $\mathcal{O}(\mathsf{M}(d)\log(k)) = \tilde{\mathcal{O}}(d)$ arithmetic operations in $\mathbb{K}$. Conversely, P is uniquely determined by $\mathcal{H}$, and its coefficients can be recovered from $\mathcal{H}$ in $\mathcal{O}(\mathsf{M}(d)\log(k)) = \tilde{\mathcal{O}}(d)$ arithmetic operations in $\mathbb{K}$.

3 The new algorithm in the case $r \geqslant d$

3.1 Multiplication by evaluation and interpolation

Most fast algorithms for multiplying two polynomials $P, Q \in \mathbb{K}[x]_d$ are based on the evaluation-interpolation strategy. The idea is to pick $2d$ distinct points $\alpha_0, \dots, \alpha_{2d-1}$ in $\mathbb{K}$, and to perform the following three steps:

Evaluation Evaluate P and Q at $\alpha_0, \dots, \alpha_{2d-1}$.

Inner multiplication Compute $(PQ)(\alpha_i) = P(\alpha_i)Q(\alpha_i)$ for $i < 2d$.

Interpolation Recover PQ from $(PQ)(\alpha_0), \dots, (PQ)(\alpha_{2d-1})$.

The inner multiplication step requires only $\mathcal{O}(d)$ operations. Consequently, if both the evaluation and interpolation steps can be performed fast, then we obtain a fast algorithm for multiplying P and Q. For instance, if $\mathbb{K}$ contains a 2^p-th primitive root of unity with $2^p \leqslant 2d \leqslant 2^{p+1}$, then both evaluation and interpolation can be performed in time $\mathcal{O}(d \log d)$ using the Fast Fourier Transform [CT65].

For a linear differential operator $L \in \mathbb{K}[x, \partial]_{d,r}$, it is natural to consider evaluations at powers of x instead of roots of unity. It is also natural to represent the evaluation of L at a suitable number of such powers by a matrix. More precisely, given $k \in \mathbb{N}$, we may regard L as an operator from $\mathbb{K}[x]_k$ into $\mathbb{K}[x]_{k+d}$. We may also regard elements of $\mathbb{K}[x]_k$ and $\mathbb{K}[x]_{k+d}$ as column vectors, written in the canonical bases with powers of x. We will denote by

$$\Phi_L^{k+d,k} \;=\; \begin{pmatrix} L(1)_0 & \cdots & L(x^{k-1})_0 \\ \vdots & & \vdots \\ L(1)_{k+d-1} & \cdots & L(x^{k-1})_{k+d-1} \end{pmatrix} \in \mathbb{K}^{(k+d)\times k}$$

the matrix of the $\mathbb{K}$-linear map $L : \mathbb{K}[x]_k \to \mathbb{K}[x]_{k+d}$ with respect to these bases. Given two operators K, L in $\mathbb{K}[x, \partial]_{d,r}$, we clearly have

$$\Phi_{KL}^{k+2d,k} = \Phi_K^{k+2d,k+d} \Phi_L^{k+d,k}, \quad \text{for all } k \geqslant 0.$$

For $k = 2r$ (or larger), the operator KL can be recovered from the matrix $\Phi_{KL}^{2r+2d,2r}$, whence the formula

$$\Phi_{KL}^{2r+2d,2r} = \Phi_K^{2r+2d,2r+d} \Phi_L^{2r+d,2r} \tag{B.1}$$

yields a way to multiply K and L. For the complexity analysis, we thus have to consider the three steps:

Evaluation Computation of $\Phi_K^{2r+2d,2r+d}$ and $\Phi_L^{2r+d,2r}$ from K and L.

Inner multiplication Computation of the matrix product (B.1).

Interpolation Recovery of KL from $\Phi_{KL}^{2r+2d,2r}$.

In [vdH02, BCLR08], this multiplication method was applied with success to the case when $d = r$. In this "square case", the following result was proved in [BCLR08, §4.2].

Lemma 1. *Let $L \in \mathbb{K}[x, \partial]_{d,d}$. Then*

1. *We may compute $\Phi_L^{2d,d}$ as a function of L in time $\mathcal{O}(d\,\mathsf{M}(d))$.*
2. *We may recover L from $\Phi_L^{2d,d}$ in time $\mathcal{O}(d\,\mathsf{M}(d))$.*

3.2 Evaluation and interpolation at exponential polynomials

Assume now that $r > d$. Then a straightforward application of the above evaluation-interpolation strategy yields an algorithm of suboptimal complexity. Indeed, the matrix $\Phi_{KL}^{2r+2d,2r}$ contains a lot of redundant information and since its mere size exceeds r^2, one cannot expect a direct multiplication algorithm of quasi-optimial complexity $\tilde{\mathcal{O}}(rd^{\omega-1})$.

In order to maintain quasi-optimal complexity in this case as well, the idea is to evaluate at so called *exponential polynomials* instead of ordinary polynomials. More specifically, given $L \in \mathbb{K}[x, \partial]_{d,r}$ and $\alpha \in \mathbb{K}$, we will use the fact that L also operates nicely on the vector space $\mathbb{K}[x]\mathrm{e}^{\alpha x}$. Moreover, for any $P \in \mathbb{K}[x]$, we have

$$L(P\mathrm{e}^{\alpha x}) = L_{\ltimes \alpha}(P)\mathrm{e}^{\alpha x},$$

where

$$L_{\ltimes \alpha} = \sum_i L_i(x)(\partial + \alpha)^i$$

is the operator obtained by substituting $\partial + \alpha$ for ∂ in $L = \sum_i L_i(x)\partial^i$. Indeed, this is a consequence of the fact that, by Leibniz's rule:

$$\partial^i(P\mathrm{e}^{\alpha x}) = \left(\sum_{j \leqslant i} \binom{i}{j} \alpha^j \partial^{i-j} P \right) \mathrm{e}^{\alpha x} = (\partial + \alpha)^i(P)\mathrm{e}^{\alpha x}.$$

Now let $p = \lceil r/d \rceil$ and let $\alpha_0, \ldots, \alpha_{p-1}$ be p pairwise distinct points in $\mathbb{K}$. For each k, we define the vector space

$$\mathbb{V}_k \quad = \quad \mathbb{K}[x]_k \mathrm{e}^{\alpha_0 x} \oplus \cdots \oplus \mathbb{K}[x]_k \mathrm{e}^{\alpha_{p-1} x}$$

with canonical basis

$$(\mathrm{e}^{\alpha_0 x}, \ldots, x^{k-1}\mathrm{e}^{\alpha_0 x}, \ldots\ldots, \mathrm{e}^{\alpha_{p-1} x}, \ldots, x^{k-1}\mathrm{e}^{\alpha_{p-1} x}).$$

Then we may regard L as an operator from $\mathbb{V}_k$ into $\mathbb{V}_{k+d}$ and we will denote by $\Phi_L^{[k+d,k]}$ the matrix of this operator with respect to the canonical bases. By what precedes, this matrix is block diagonal, with p blocks of size d:

$$\Phi_L^{[k+d,k]} \quad = \quad \begin{pmatrix} \Phi_{L \kappa \alpha_0}^{k+d,k} & & \\ & \ddots & \\ & & \Phi_{L \kappa \alpha_{p-1}}^{k+d,k} \end{pmatrix}.$$

Let us now show that the operator L is uniquely determined by the matrix $\Phi_L^{[2d,d]}$, and that this gives rise to an efficient algorithm for multiplying two operators in $\mathbb{K}[x, \partial]_{d,r}$.

Lemma 2. *Let $L \in \mathbb{K}[x, \partial]_{d,r}$ with $r \geqslant d$ and let $p = \lceil r/d \rceil$. Then*

1. *We may compute $\Phi_L^{[2d,d]}$ as a function of L in time $\mathcal{O}(d\,\mathsf{M}(r) \log r)$.*
2. *We may recover L from $\Phi_L^{[2d,d]}$ in time $\mathcal{O}(d\,\mathsf{M}(r) \log r)$.*

Proof. For any operator $L = \sum_{i<d, j<r} L_{i,j} x^i \partial^j$ in $\mathbb{K}[x, \partial]_{d,r}$, we define its truncation L^* at order $\mathcal{O}(\partial^d)$ by

$$L^* \quad = \quad \sum_{i<d, j<d} L_{i,j} x^i \partial^j.$$

Since $L - L^*$ vanishes on $\mathbb{K}[x]_d$, we notice that $\Phi_L^{2d,d} = \Phi_{L^*}^{2d,d}$.

If $L \in \mathbb{K}[\partial]_r$, then L^* can be regarded as the power series expansion of L at $\partial = 0$ and order d. More generally, for any $i \in \{0, \ldots, p-1\}$, the operator $L^*_{\kappa \alpha_i}$ coincides with the Taylor series expansion at $\partial = \alpha_i$ and order d:

$$\begin{aligned} L^*_{\kappa \alpha_i}(\partial) \quad &= \quad L(\partial + \alpha_i)^* \\ &= \quad L(\alpha_i) + L'(\alpha_i)\partial + \cdots + \tfrac{1}{(d-1)!} L^{(d-1)}(\alpha_i) \partial^{d-1}. \end{aligned}$$

In other words, the computation of the truncated operators $L^*_{\kappa \alpha_0}, \ldots, L^*_{\kappa \alpha_{p-1}}$ as a function of L corresponds to a Hermite evaluation at the points α_i, with multiplicity $c_i = d$ at each point α_i. By Theorem 2, this computation can be performed in time $\mathcal{O}(\mathsf{M}(pd) \log(pd)) = \mathcal{O}(\mathsf{M}(r) \log r)$. Furthermore, Hermite interpolation allows us to recover L from $L^*_{\kappa \alpha_0}, \ldots, L^*_{\kappa \alpha_{p-1}}$ with the same time complexity $\mathcal{O}(\mathsf{M}(r) \log r)$.

Now let $L \in \mathbb{K}[x,\partial]_{d,r}$ and consider the expansion of L in x

$$L(x,\partial) \quad = \quad L_0(\partial) + \cdots + x^{d-1} L_{d-1}(\partial).$$

For each i, one Hermite evaluation of L_i allows us to compute the $L^*_{\kappa\alpha_j,i}$ with $j < p$ in time $\mathcal{O}(\mathsf{M}(r)\log r)$. The operators $L^*_{\kappa\alpha_j}$ with $j < p$ can therefore be computed in time $\mathcal{O}(d\,\mathsf{M}(r)\log r)$. By Lemma 1, we need $\mathcal{O}(r\,\mathsf{M}(d)) = \mathcal{O}(d\,\mathsf{M}(r))$ additional operations in order to obtain $\Phi_L^{[2d,d]}$. Similarly, given $\Phi_L^{[2d,d]}$, Lemma 1 allows us to recover the operators $L^*_{\kappa\alpha_j}$ with $j < p$ in time $\mathcal{O}(d\,\mathsf{M}(r))$. Using d Hermite interpolations, we also recover the coefficients L_i of L in time $\mathcal{O}(d\,\mathsf{M}(r)\log r)$. □

Theorem 3. *Let $K, L \in \mathbb{K}[x,\partial]_{d,r}$ with $r \geqslant d$. Then we may compute the product KL in time $\mathcal{O}(d^{\omega-1}r + d\,\mathsf{M}(r)\log r)$.*

Proof. Considering K and L as operators in $\mathbb{K}[x,\partial]_{3d,3r}$, Lemma 2 implies that the computation of $\Phi_K^{[4d,3d]}$ and $\Phi_L^{[3d,2d]}$ as a function of K and L can be done in time $\mathcal{O}(d\,\mathsf{M}(r)\log r)$. The multiplication

$$\Phi_{KL}^{[4d,2d]} \quad = \quad \Phi_K^{[4d,3d]} \Phi_L^{[3d,2d]}$$

can be done in time $\mathcal{O}(pd^\omega) = \mathcal{O}(d^{\omega-1}r)$. Lemma 2 finally implies that we may recover KL from $\Phi_{KL}^{[4d,2d]}$ in time $\mathcal{O}(d\,\mathsf{M}(r)\log r)$. □

4 The new algorithm in the case $d \geqslant r$

Any differential operator $L \in \mathbb{K}[x,\partial]_{d,r}$ can be written in a unique form

$$L = \sum_{i<r,j<d} L_{i,j} x^j \partial^i, \quad \text{for some scalars} \quad L_{i,j} \in \mathbb{K}.$$

This representation, with x on the left and ∂ on the right, is called the *canonical form of* L.

Let $\varphi : \mathbb{K}[x,\partial] \to \mathbb{K}[x,\partial]$ denote the map defined by

$$\varphi\left(\sum_{i<r,j<d} L_{i,j} x^j \partial^i\right) = \sum_{i<r,j<d} L_{i,j} \partial^j (-x)^i.$$

In other words, φ is the unique $\mathbb{K}$-algebra automorphism of $\mathbb{K}[x,\partial]$ that keeps the elements of $\mathbb{K}$ fixed, and is defined on the generators of $\mathbb{K}[x,\partial]$ by $\varphi(x) = \partial$ and $\varphi(\partial) = -x$. We will call φ the *reflection morphism* of $\mathbb{K}[x,\partial]$. The map φ enjoys the nice property that it sends $\mathbb{K}[x,\partial]_{d,r}$ onto $\mathbb{K}[x,\partial]_{r,d}$. In particular, to an operator whose order is higher than its degree, φ associates a "mirror operator" whose degree is higher than its order.

4.1 Main idea of the algorithm in the case $d \geqslant r$

If $d \geqslant r$, then the reflection morphism φ is the key to our fast multiplication algorithm of operators in $\mathbb{K}[x, \partial]_{d,r}$, since it allows us to reduce this case to the previous case when $r \geqslant d$. More precisely, given K, L in $\mathbb{K}[x, \partial]_{d,r}$ with $d \geqslant r$, the main steps of the algorithm are:

(S1) compute the canonical forms of $\varphi(K)$ and $\varphi(L)$,

(S2) compute the product $M = \varphi(K)\varphi(L)$ of operators $\varphi(K) \in \mathbb{K}[x, \partial]_{r,d}$ and $\varphi(L) \in \mathbb{K}[x, \partial]_{r,d}$, using the algorithm described in the previous section, and

(S3) return the (canonical form of the) operator $KL = \varphi^{-1}(M)$.

Since $d \geqslant r$, step (S2) can be performed in complexity $\tilde{\mathcal{O}}(r^{\omega-1}d)$ using the results of Section 3. In the next subsection, we will prove that both steps (S1) and (S3) can be performed in $\tilde{\mathcal{O}}(rd)$ operations in $\mathbb{K}$. This will enable us to conclude the proof of Theorem 1.

4.2 Quasi-optimal computation of reflections

We now show that the *reflection* and the *inverse reflection* of a differential operator can be computed quasi-optimally. The idea is that performing reflections can be interpreted in terms of Taylor shifts for polynomials, which can be computed in quasi-linear time using the algorithm from [ASU75].

A first observation is that the composition $\varphi \circ \varphi$ is equal to the involution $\psi : \mathbb{K}[x, \partial] \to \mathbb{K}[x, \partial]$ defined by

$$\psi\left(\sum_{i<r, j<d} L_{i,j} x^j \partial^i\right) = \sum_{i<r, j<d} (-1)^{i+j} L_{i,j} x^j \partial^i.$$

As a direct consequence of this fact, it follows that the map φ^{-1} is equal to $\varphi \circ \psi$. Since $\psi(L)$ is already in canonical form, computing $\psi(L)$ only consists of sign changes, which can be done in linear time $\mathcal{O}(dr)$. Therefore, computing the inverse reflection $\varphi^{-1}(L)$ can be performed within the same cost as computing the direct reflection $\varphi(L)$, up to a linear overhead $\mathcal{O}(rd)$.

In the remainder of this section, we focus on the fast computation of direct reflections. The key observation is encapsulated in the next lemma. Here, and in what follows, we use the convention that the entries of a matrix corresponding to indices beyond the matrix sizes are all zero.

Lemma 3. *Assume that $(p_{i,j})$ and $(q_{i,j})$ are two matrices in $\mathbb{K}^{r \times d}$ such that*

$$\sum_{i,j} q_{i,j} x^i \partial^j = \sum_{i,j} p_{i,j} \partial^j x^i.$$

Then

$$i!\, q_{i,j} = \sum_{k \geqslant 0} \binom{j+k}{k} (i+k)!\, p_{i+k, j+k}$$

and where we use the convention that $p_{i,j} = 0$ *as soon as* $i \geqslant r$ *or* $j \geqslant d$.

Proof. Leibniz's differentiation rule implies the commutation rule

$$\partial^j \frac{x^i}{i!} = \sum_{k=0}^{j} \binom{j}{k} \frac{x^{i-k}}{(i-k)!} \partial^{j-k}.$$

Together with the hypothesis, this implies the equality

$$\sum_{i,j} (i!\, q_{i,j}) \frac{x^i}{i!} \partial^j = \sum_{i,j} (i!\, p_{i,j}) \partial^j \frac{x^i}{i!} = \sum_{k \geqslant 0} \left(\sum_{i,j} (i!\, p_{i,j}) \binom{j}{k} \frac{x^{i-k}}{(i-k)!} \partial^{j-k} \right).$$

We conclude by extraction of coefficients. □

Theorem 4. *Let* $L \in \mathbb{K}[x, \partial]_{d,r}$. *Then we may compute* $\varphi(L)$ *and* $\varphi^{-1}(L)$ *using* $\mathcal{O}(\min(d\,\mathsf{M}(r), r\,\mathsf{M}(d))) = \tilde{\mathcal{O}}(rd)$ *operations in* $\mathbb{K}$.

Proof. We first study the case $r \geqslant d$. If $L = \sum_{i<r, j<d} p_{i,j} x^j \partial^i$, then by the first equality of Lemma 3, the reflection $\varphi(L)$ is equal to

$$\varphi(L) = \sum_{i<r, j<d} p_{i,j} \partial^j (-x)^i = \sum_{i<r, j<d} q_{i,j} (-x)^j \partial^i,$$

where

$$i!\, q_{i,j} = \sum_{\ell \geqslant 0} \binom{j+\ell}{j} (i+\ell)!\, p_{i+\ell, j+\ell}. \tag{B.2}$$

For any fixed k with $1 - r \leqslant k \leqslant d - 1$, let us introduce $G_k = \sum_i i! q_{i,i+k} x^{i+k}$ and $F_k = \sum_i i! p_{i,i+k} x^{i+k}$. These polynomials belong to $\mathbb{K}[x]_d$, since $p_{i,j} = q_{i,j} = 0$ for $j \geqslant d$. If $k \leqslant 0$, then Equation (B.2) translates into

$$G_k(x) = F_k(x+1).$$

Indeed, Equation (B.2) with $j = i + k$ implies that $G_k(x)$ is equal to

$$\sum_{i,\ell} \binom{i+k+\ell}{i+k} (i+\ell)!\, p_{i+\ell, i+k+\ell}\, x^{i+k} = \sum_{j,s} j! p_{j,j+k} \binom{j+k}{s} x^s = F_k(x+1).$$

Similarly, if $k > 0$, then the coefficients of x^i in $G_k(x)$ and $F_k(x+1)$ still coincide for all $i \geqslant k$. In particular, we may compute $G_{1-r}, \ldots, G_{d-1}$ from $F_{1-r}, \ldots, F_{d-1}$ by means of $d + r \leqslant 2r$ Taylor shifts of polynomials in $\mathbb{K}[x]_d$. Using the fast algorithm for Taylor shift in [ASU75], this can be done in time $\mathcal{O}(r\,\mathsf{M}(d))$.

Once the coefficients of the G_k's are available, the computation of the coefficients of $\varphi(L)$ requires $\mathcal{O}(dr)$ additional operations.

If $d \geqslant r$, then we notice that the equality (B.2) is equivalent to

$$j!\, q_{i,j} = \sum_{\ell \geqslant 0} \binom{i+\ell}{i} (j+\ell)!\, p_{i+\ell,j+\ell},$$

as can be seen by expanding the binomial coefficients. Redefining $G_k := \sum_i q_{i+k,i} x^{i+k}$ and $F_k := \sum_i i! p_{i+k,i} x^{i+k}$, similar arguments as above show that $\varphi(P)$ can be computed using $\mathcal{O}(d\,\mathsf{M}(r))$ operations in $\mathbb{K}$.

By what has been said at the beginning of this section, we finally conclude that the inverse reflection $\varphi^{-1}(L) = \varphi(\psi(L))$ can be computed for the same cost as the direct reflection $\varphi(L)$. □

4.3 Proof of Theorem 1 in the case $d \geqslant r$

We will prove a slightly better result:

Theorem 5. *Let $K, L \in \mathbb{K}[x,\partial]_{d,r}$ with $d \geqslant r$. Then we may compute the product KL using $\mathcal{O}(r^{\omega-1}d + r\,\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$.*

Proof. Assume that K and L are two operators in $\mathbb{K}[x,\partial]_{d,r}$ with $d \geqslant r$. Then $\varphi(K)$ and $\varphi(L)$ belong to $\mathbb{K}[x,\partial]_{r,d}$, and their canonical forms can be computed in $\mathcal{O}(d\mathsf{M}(r))$ operations by Theorem 4. Using the algorithm from section 3, we may compute $M = \varphi(L)\varphi(K)$ in $\mathcal{O}(r^{\omega-1}d + r\,\mathsf{M}(d)\log d)$ operations. Finally, $LK = \varphi^{-1}(M)$ can be computed in $\mathcal{O}(r\,\mathsf{M}(d))$ operations by Theorem 4. We conclude by adding up the costs of these three steps. □

Liste des algorithmes

Table des figures

Index

Bibliographie

[ADGR04] Iván Area, Dimitar K. Dimitrov, Eduardo Godoy et André Ronveaux : Zeros of Gegenbauer and Hermite polynomials and connection coefficients. *Math. Comp.*, 73(248):1937–1951 (electronic), 2004.

[AHU74] Alfred V. Aho, John E. Hopcroft et Jeffrey D. Ullman : *The design and analysis of computer algorithms*. Addison-Wesley Publishing Co., 1974.

[ALL03] S.A. Abramov, HQ Le et Ziming Li : Oretools : A computer algebra library for univariate ore polynomial rings. *School of Computer Science CS-2003-12, University of Waterloo*, 2003.

[AO05] Simon Altmann et Eduardo L. Ortiz, éditeurs. *Mathematics and social utopias in France*, volume 28 de *History of Mathematics*. American Mathematical Society, Providence, RI, 2005. Olinde Rodrigues and his times.

[App81] P. Appell : Mémoire sur les équations différentielles linéaires. *Ann. Sci. École Norm. Sup. (2)*, 10:391–424, 1881.

[AS64] Milton Abramowitz et Irene A. Stegun : *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55 de *National Bureau of Standards Applied Mathematics Series*. For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C., 1964.

[Ask75] Richard Askey : *Orthogonal polynomials and special functions*. Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1975.

[ASU75] A. V. Aho, K. Steiglitz et J. D. Ullman : Evaluating polynomials at fixed sets of points. *SIAM J. Comput.*, 4(4):533–539, 1975.

[AZ90] Gert Almkvist et Doron Zeilberger : The method of differentiating under the integral sign. *J. Symbolic Comput.*, 10(6):571–591, 1990.

[BBvdH] Alexandre Benoit, Alin Bostan et Joris van der Hoeven : Fast multiplication of skew polynomials. In preparation.

[BBvdH12] Alexandre Benoit, Alin Bostan et Joris van der Hoeven : Quasi-optimal multiplication of linear differential operators. *In FOCS'12 : Proceding of the IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2012, Hyatt Regency, New Brunswick, New Jersey*, 2012. http://arxiv.org/abs/1208.3639.

[BCD+10] Alexandre Benoit, Frédéric Chyzak, Alexis Darrasse, Stefan Gerhold, Marc Mezzarobba et Bruno Salvy : The dynamic dictionary of mathematical functions (DDMF). *In* Komei Fukuda, Joris van der Hoeven, Michael Joswig et Nobuki Takayama, éditeurs : *The Third International Congress on Mathematical Software (ICMS 2010)*, volume 6327 de *Lecture Notes in Computer Science*, pages 35–41, 2010.

[BCLR03] Moulay Barkatou, Frédéric Chyzak et Michèle Loday-Richaud : Remarques algorithmiques liées au rang d'un opérateur différentiel linéaire. *In* F. Fauvet et C. Mitschi, éditeurs : *From Combinatorics to Dynamical Systems (Journées de Calcul Formel, Strasbourg, March 22-23, 2002)*, volume 3 de *IRMA Lectures in Mathematics and Theoretical Physics*, pages 87–129. de Gruyter, 2003. In French. ISBN 3-11-017875-3.

[BCLR08] Alin Bostan, Frédéric Chyzak et Nicolas Le Roux : Products of ordinary differential operators by evaluation and interpolation. *In ISSAC 2008*, pages 23–30. ACM, New York, 2008.

[BCLS12] A. Bostan, F. Chyzak, Z. Li et B. Salvy : Fast computation of common left multiples of linear ordinary differential operators. *In ISSAC'12*, 2012. To appear. Preliminary version available at http://algo.inria.fr/bostan/publications/BoChLiSa12.pdf.

[BCM+52] W. G. Bickley, L. J. Comrie, J. C. P. Miller, D. H. Sadler et A. J. Thompson : *Bessel functions. Part II. Functions of positive integer order*. British Association for the Advancement of Science, Mathematical Tables, vol. X. University Press, Cambridge, 1952.

[BCS97] P. Bürgisser, M. Clausen et M. A. Shokrollahi : *Algebraic complexity theory*. Springer-Verlag, 1997.

[BGY80] Richard P. Brent, Fred G. Gustavson et David Y. Y. Yun : Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1(3):259–295, 1980.

[BH74] James R. Bunch et John E. Hopcroft : Triangular factorization and inversion by fast matrix multiplication. *Math. Comp.*, 28(125):231–236, 1974.

[BJ10] N. Brisebarre et M. Joldeş : Chebyshev interpolation polynomial-based tools for rigorous computing. *In Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 147–154. ACM, 2010.

[BJM] Alexandre Benoit, Mioara Joldeş et Marc Mezzarobba : Rigorous uniform approximation of D-finite functions using Chebyshev expansions. En préparation.

[BL01] R. Boisvert et D.W. Lozier : *Handbook of Mathematical Functions*, chapitre A Century of Excellence in Measurements Standards and Technologys, pages 135–139. CRC Press, 2001.

[BM74] A. Borodin et R. Moenck : Fast modular transforms. *J. Comput. System Sci.*, 8:366–386, 1974. Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972).

[Boy01] John P. Boyd : *Chebyshev and Fourier spectral methods.* Dover Publications Inc., Mineola, NY, second édition, 2001.

[BP94a] Dario Bini et Victor Y. Pan : *Polynomial and matrix computations. Vol. 1.* Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1994. Fundamental algorithms.

[BP94b] Manuel Bronstein et Marko Petkovšek : Ore rings, linear operators and factorization. *Programmirovanie*, (1):27–44, 1994.

[BP96] Manuel Bronstein et Marko Petkovšek : An introduction to pseudo-linear algebra. volume 157, pages 3–33, 1996. Algorithmic complexity of algebraic and geometric models (Creteil, 1994).

[Bra64] E. Brassinne : Analogie des équations différentielles linéaires à coefficients variables, avec les équations algébriques. *In Note III du Tome 2 du Cours d'analyse de Ch. Sturm, École polytechnique, 2ème édition*, pages 331–347, 1864.

[BS93] Manuel Bronstein et Bruno Salvy : Full partial fraction decomposition of rational functions. *In* Manuel Bronstein, éditeur : *ISSAC'93*, pages 157–160. ACM, 1993.

[BS05] A. Bostan et É. Schost : Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, August 2005. Festschrift for the 70th Birthday of Arnold Schönhage.

[BS09] Alexandre Benoit et Bruno Salvy : Chebyshev expansions for solutions of linear differential equations. *In* John May, éditeur : *ISSAC '09 : Proceedings of the twenty-second international symposium on Symbolic and algebraic computation*, pages 23–30, 2009.

[BSS08] Alin Bostan, Bruno Salvy et Éric Schost : Power series composition and change of basis. *In* David J. Jeffrey, éditeur : *ISSAC'08 : Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 269–276. ACM Press, 2008.

[BSS10] Alin Bostan, Bruno Salvy et Éric Schost : Fast conversion algorithms for orthogonal polynomials. *Linear Algebra and its Applications*, 432(1):249–258, January 2010.

[BT97] Günter Baszenski et Manfred Tasche : Fast polynomial multiplication and convolutions related to the discrete cosine transform. *Linear Algebra Appl.*, 252:1–25, 1997.

[BZ10] Richard P. Brent et Paul Zimmermann : *Modern Computer Arithmetic.* Cambridge University Press, nov 2010.

[CD11] Frédéric Chyzak et Alexis Darrasse : Using camlp4 for presenting dynamic mathematics on the web : Dynamow, an ocaml language extension for the run-time generation of mathematical contents and their presentation on the web. *In* Olivier Danvy, éditeur : *ICFP'11 (September 19–21, 2011, Tokyo, Japan)*, pages 259–265. ACM, 2011. (An experience report.).

[Che98] E.W. Cheney : *Introduction to approximation theory*. Amer Mathematical Society, 1998.

[Chi76] Francis Y. Chin : A generalized asymptotic upper bound on fast polynomial evaluation and interpolation. *SIAM J. Comput.*, 5(4):682–690, 1976.

[Chy98] Frédéric Chyzak : *Fonctions holonomes en calcul formel*. Thèse de doctorat, Ecole Polytechnique, 1998.

[Chy00] Frédéric Chyzak : An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Math.*, 217(1-3):115–134, 2000. Formal power series and algebraic combinatorics (Vienna, 1997).

[CJL10] S. Chevillard, M. Joldeş et C. Lauter : Sollya : An environment for the development of numerical codes. *In* K. Fukuda, J. van der Hoeven, M. Joswig et N. Takayama, éditeurs : *Mathematical Software - ICMS 2010*, volume 6327 de *Lecture Notes in Computer Science*, pages 28–31, Heidelberg, Germany, September 2010. Springer.

[CK91] D. G. Cantor et E. Kaltofen : On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28(7):693–701, 1991.

[CKS09] Frédéric Chyzak, Manuel Kauers et Bruno Salvy : A non-holonomic systems approach to special function identities. *In* John May, éditeur : *ISSAC '09 : Proceedings of the twenty-second international symposium on Symbolic and algebraic computation*, pages 111–118, 2009.

[Cle57] C. W. Clenshaw : The numerical solution of linear differential equations in Chebyshev series. *Proc. Cambridge Philos. Soc.*, 53:134–149, 1957.

[Coh63] Paul Moritz Cohn : Rings with a weak algorithm. *Transactions of the American Mathematical Society*, 109(2):332–356, November 1963.

[Cor94] George F. Corliss : Guaranteed error bounds for ordinary differential equations. *In In Theory of Numerics in Ordinary and Partial Differential Equations*, pages 1–75. Oxford University Press, 1994.

[CP70] E. W. Cheney et K. H. Price : Minimal projections. *In Approximation Theory (Proc. Sympos., Lancaster, 1969)*, pages 261–289. Academic Press, London, 1970.

[CS98] Frédéric Chyzak et Bruno Salvy : Non-commutative elimination in Ore algebras proves multivariate identities. *J. Symbolic Comput.*, 26(2):187–227, 1998.

[CT65] James W. Cooley et John W. Tukey : An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[CvHL10] Yongjae Cha, Mark van Hoeij et Giles Levy : Solving recurrence relations using local invariants. *In Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 303–309, New York, NY, USA, 2010. ACM.

[CW90] D. Coppersmith et S. Winograd : Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, mars 1990.

[DBT08] T.A. Driscoll, F. Bornemann et L.N. Trefethen : The chebop system for automatic solution of differential equations. *BIT Numerical Mathematics*, 48(4):701–723, 2008.

[Dem83] S. S. Demidov : On the history of the theory of linear differential equations. *Arch. Hist. Exact Sci.*, 28(4):369–387, 1983.

[EDOS93] M. K. El-Daou, E. L. Ortiz et H. Samara : A unified approach to the tau method and Chebyshev series expansion techniques. *Comput. Math. Appl.*, 25(3):73–82, 1993.

[EF89] T. H. Einwohner et R. J. Fateman : A MACSYMA package for the generation and manipulation of Chebyshev series. *In Proceedings of the ACM-SIGSAM 1989 international symposium on Symbolic and algebraic computation*, ISSAC '89, pages 180–185, New York, NY, USA, 1989. ACM.

[EMR82a] C. Epstein, W.L. Miranker et T.J. Rivlin : Ultra-arithmetic i : function data types. *Mathematics and Computers in Simulation*, 24(1):1–18, 1982.

[EMR82b] C. Epstein, W.L. Miranker et T.J. Rivlin : Ultra-arithmetic ii : intervals of polynomials. *Mathematics and Computers in Simulation*, 24(1):19–29, 1982.

[Erd81] A. Erdélyi : *Higher Transcendental Functions*, volume 2. R. E. Krieger publishing Company, Inc., Malabar, Florida, second édition, 1981.

[Fav35] Jean Favard : Sur les polynomes de Tchebicheff. *Académie des sciences (France). Comptes rendus hebdomadaires des séances de l'Académie des sciences.*, (200):1952–1953, 1935.

[Fox62] L. Fox : Chebyshev methods for ordinary differential equations. *The Computer Journal*, 4(4):318, 1962.

[FP68] L. Fox et I.B. Parker : *Chebyshev polynomials in numerical analysis*. Oxford University Press, 1968.

[FW61] Jerry L. Fields et Jet Wimp : Expansions of hypergeometric functions in hypergeometric functions. *Math. Comp.*, 15:390–395, 1961.

[vzGG03] Joachim von zur Gathen et Jürgen Gerhard : *Modern computer algebra*. Cambridge University Press, Cambridge, second édition, 2003.

[Ged77a] K. O. Geddes : ALTRAN procedures for the Chebyshev series solution of linear ode's.hebyshev series solution of linear ode's. Rapport technique, University of Waterloo,, 1977.

[Ged77b] K. O. Geddes : Symbolic computation of recurrence equations for the Chebyshev series solution of linear ODE's. *In* Carl M. Andersen, éditeur : *Proceedings of the 1977 MACSYMA User's Conference*, pages 405–423, 1977. NASA CP-2012.

[Geg72] Leopold Gegenbauer : Zur theorie der functionen X_n^m. *Sitzungsberichte der Kaiserlichen Akademie der Wissenschaten mathematisch-naturwissenschaftlichen Classe*, 66(2):55–62, 1872.

[Geg84] Leopold Gegenbauer : Zur theorie der functionen $c_n^\nu(x)$. *Denkschriften der Kaiserlichen Akademie der Wissenschaften*, 48:293–316, 1884.

[GGWY82] William B. Gragg, Fred G. Gustavson, Daniel D. Warner et David Y. Y. Yun : On fast computation of superdiagonal Padé fractions. *Math. Programming Stud.*, (18):39–42, 1982. Algorithms and theory in filtering and control (Lexington, Ky., 1980).

[Gio12] P. Giorgi : On polynomial multiplication in Chebyshev basis. *IEEE Transactions on Computers*, 61(6):780 –789, june 2012.

[GR96] I.S. Gradshteyn et I.M. Ryzhik : *Table of Integrals, Series, and Products*. Academic Press, 1996.

[Gri90] D. Yu. Grigorev : Complexity of factoring and calculating the GCD of linear ordinary differential operators. *J. Symbolic Comput.*, 10(1):7–37, 1990.

[GS96] Xavier Gourdon et Bruno Salvy : Effective asymptotics of linear recurrences with rational coefficients. *Discrete Mathematics*, 153(1-3):145–163, 1996.

[GST07] Amparo Gil, Javier Segura et Nico M. Temme : *Numerical methods for special functions*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2007.

[Gue63] A. O. Guelfond : *Calcul des différences finies*. Collection Universitaire de Mathématiques, XII. Traduit par G. Rideau. Dunod, Paris, 1963.

[GY79] Fred G. Gustavson et David Y. Y. Yun : Fast algorithms for rational Hermite approximation and solution of Toeplitz systems. *IEEE Trans. Circuits and Systems*, 26(9):750–755, 1979.

[Had99] Jacques Hadamard : Théorème sur les séries entières. *Acta Math.*, 22(1):55–63, 1899.

[Hal08] M. Halás : An algebraic framework generalizing the concept of transfer functions to nonlinear systems. *Automatica*, 44(5):1181–1190, 2008.

[Han75] Eldon R. Hansen : A table of series and products. *Prentice Hall Series in Automatic Computation, Englewood Cliffs : Prentice Hall, 1975*, 1, 1975.

[HK07] M. Halás et Ü. Kotta : Pseudo-linear algebra : a powerful tool in unification of the study of nonlinear control systems. *In Proceedings of 7th IFAC Symposium on Nonlinear Control Systems*, 2007.

[vdH02] Joris van der Hoeven : FFT-like multiplication of linear differential operators. *J. Symbolic Comput.*, 33(1):123–127, 2002.

[vdH11] Joris van der Hoeven : On the complexity of skew arithmetic, 2011. Technical Report, HAL 00557750, http://hal.archives-ouvertes.fr/hal-00557750.

[HQZ04] Guillaume Hanrot, Michel Quercia et Paul Zimmermann : The middle product algorithm. I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[Inc] OEIS Foundation Inc. : The On-Line Encyclopedia of Integer Sequences. http://oeis.org.

[Jol11] Mioara Joldeş : *Approximations polynomiales rigoureuses et applications*. Thèse de doctorat, École Normale Supérieure de Lyon, Sept. 2011.

[Kal93] Erich. Kaltofen : Computational differentiation and algebraic complexity theory. *C. H. Bischof, A. Griewank, and P. M. Khademi, editors, Workshop Report on First Theory Institute on Computational Differentiation*, ANL/MCS-TM-183:28–30, Decembrer 1993.

[KM84] E.W. Kaucher et W.L. Miranker : *Self-validating numerics for function space problems*. Academic Press, 1984.

[KM88] Edgar Kaucher et Willard L. Miranker : Validating computation in a function space. *In Reliability in computing*, volume 19 de *Perspect. Comput.*, pages 403–425. Academic Press, Boston, MA, 1988.

[Kou10a] Christoph Koutschan : A fast approach to creative telescoping. *Math. Comput. Sci.*, 4(2-3):259–266, 2010.

[Kou10b] Christoph Koutschan : HolonomicFunctions (User's Guide). Rapport technique 10-01, RISC Report Series, University of Linz, Austria, January 2010.

[Kra06] W. Kramer : intpakx - an interval arithmetic package for maple. *In Scientific Computing, Computer Arithmetic and Validated Numerics, 2006. SCAN 2006. 12th GAMM - IMACS International Symposium on*, page 27, sept. 2006.

[KS98] Roelof Koekoek et René F. Swarttouw : The Askey-scheme of hypergeometric orthogonal polynomials and it's q-analogue. Rapport technique Report 98–17, Delft University of Technology, Faculty of Information Technology and Systems, 1998.

[Lan38] C. Lanczos : Trigonometric interpolation of empirical and analytical functions. *J. Math. Phys*, 17:123–199, 1938.

[Lan56] C. Lanczos : *Applied analysis*. Prentice-Hall, 1956.

[Lan02] Serge Lang : *Algebra*, volume 211 de *Graduate Texts in Mathematics*. Springer-Verlag, New York, third édition, 2002.

[LC61] Yudell L. Luke et Richard L. Coleman : Expansion of hypergeometric functions in series of other hypergeometric functions. *Math. Comp.*, 15:233–237, 1961.

[Lew76] Stanisław Lewanowicz : Construction of a recurrence relation of the lowest order for coefficients of the Gegenbauer series. *Zastos. Mat.*, 15(3):345–396, 1976.

[Lew79] Stanisław Lewanowicz : Construction of a recurrence relation for modified moments. *J. Comput. Appl. Math.*, 5(3):193–206, 1979.

[Lew83] Stanisław Lewanowicz : Construction of the lowest-order recurrence relation for the Jacobi coefficients. *Zastos. Mat.*, 17(4):655–675, 1983.

[Lew85] Stanisław Lewanowicz : Recurrence relations for hypergeometric functions of unit argument. *Math. Comp.*, 45(172):521–535, 1985.

[Lew86] Stanisław Lewanowicz : Recurrence relations for the coefficients in Jacobi series solutions of linear differential equations. *SIAM J. Math. Anal.*, 17(5):1037–1052, 1986.

[Lew91] S. Lewanowicz : A new approach to the problem of constructing recurrence relations for the Jacobi coefficients. *Zastos. Mat.*, 21(2):303–326, 1991.

[Lew92] Stanisław Lewanowicz : Quick construction of recurrence relations for the Jacobi coefficients. *J. Comput. Appl. Math.*, 43(3):355–372, 1992.

[Lib33] G. Libri : Mémoire sur la résolution des équations algébriques dont les racines ont entre elles un rapport donné, et sur l'intégration des équations différentielles linéaires dont les intégrales paticulières peuvent s'exprimer les unes par les autres. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, (10):167–194, 1833.

[Lib36] G. Libri : Note sur les rapports qui existent entre la théorie des équations algébriques et la théorie des équations linéaires aux différentielles et aux différences. *J. math, pures et appt*, 1:10–13, 1836.

[Lip89] L. Lipshitz : D-finite power series. *Journal of Algebra*, 122(2):353–373, 1989.

[Luk59] Yudell L. Luke : Expansion of the confluent hypergeometric function in series of Bessel functions. *Math. Tables Aids Comput.*, 13:261–271, 1959.

[Luk69] Yudell L. Luke : *The special functions and their approximations, Vol. II.* Mathematics in Science and Engineering, Vol. 53. Academic Press, New York, New York, 1969.

[LW04] Stanisław Lewanowicz et Paweł Woźny : Recurrence relations for the coefficients in series expansions with respect to semi-classical orthogonal polynomials. *Numer. Algorithms*, 35(1):61–79, 2004.

[Mat06] Richard J. Mathar : Chebyshev series expansion of inverse polynomials. *J. Comput. Appl. Math.*, 196(2):596–607, 2006.

[MD73] K. O. Mead et L. M. Delves : On the convergence rate of generalized Fourier expansions. *J. Inst. Math. Appl.*, 12:247–259, 1973.

[Mez10] Marc Mezzarobba : NumGfun : a package for numerical and analytic computation with D-finite functions. *In Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation (ISSAC 2010)*, pages 139–145. ACM, 2010.

[Mez11] Marc Mezzarobba : *Autour de l'évaluation numérique des fonctions D-finies*. Thèse de doctorat, École polytechnique, Palaiseau, France, November 2011.

[MH03] J. C. Mason et D. C. Handscomb : *Chebyshev polynomials*. Chapman & Hall/CRC, Boca Raton, FL, 2003.

[Moe76] Robert T. Moenck : Another polynomial homomorphism. *Acta Informat.*, 6(2):153–169, 1976.

[MS03] Ludovic Meunier et Bruno Salvy : ESF : An automatically generated encyclopedia of special functions. *In* J. R. Sendra, éditeur : *Symbolic and Algebraic Computation*, pages 199–205. ACM Press, 2003. Proceedings of ISSAC'03, Philadelphia, August 2003.

[MS10] M. Mezzarobba et B. Salvy : Effective bounds for p-recursive sequences. *Journal of Symbolic Computation*, 45(10):1075–1096, 2010.

[Nat10] National Institute of Standards and Technology : Digital library of mathematical functions. http://dlmf.nist.gov/, May 2010.

[Neu67] C.G. Neumann : *Theorie der Bessel'chen Functionen*. 1867.

[NJC99] N. S. Nedialkov, K. R. Jackson et G. F. Corliss : Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21 – 68, 1999.

[NU88] Arnold F. Nikiforov et Vasilii B. Uvarov : *Special functions of mathematical physics*. Birkhäuser Verlag, Basel, 1988. A unified introduction with applications, Translated from the Russian and with a preface by Ralph P. Boas, With a foreword by A. A. Samarskiĭ.

[OLBC10] F.W.J. Olver, D.W. Lozier, R.F. Boisvert et C.W. Clark, éditeurs. *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.

[Ore31] Oystein Ore : Linear equations in non-commutative fields. *Ann. of Math. (2)*, 32(3):463–477, 1931.

[Ore32] Oystein Ore : Formale Theorie der linearen Differentialgleichungen. *J. Reine Angew. Math.*, 167:221–234, 1932.

[Ore33] Oystein Ore : Theory of non-commutative polynomials. *Ann. of Math. (2)*, 34(3):480–508, 1933.

[OS00] Vadim Olshevsky et Amin Shokrollahi : Matrix-vector product for confluent Cauchy-like matrices with application to confluent rational interpolation. *In Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 573–581 (electronic), New York, 2000. ACM.

[Pan84] Victor Pan : *How to multiply matrices faster*, volume 179 de *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1984.

[Pan96] V. Y. Pan : Optimal and nearly optimal algorithms for approximating polynomial zeros. *Comput. Math. Appl*, 31:97–138, 1996.

[Pan98] V. Y. Pan : New fast algorithms for polynomial interpolation and evaluation on the Chebyshev node set. *Comput. Math. Appl.*, 35(3):125–129, 1998.

[Pan01] Victor Y. Pan : *Structured matrices and polynomials*. Birkhäuser Boston Inc., Boston, MA, 2001. Unified superfast algorithms.

[Pas75] Stefan Paszkowski : *Zastosowania numeryczne wielomianów i szeregów Czebyszewa*. Państwowe Wydawnictwo Naukowe, Warsaw, 1975. Podstawowe Algorytmy Numeryczne. [Fundamental Numerical Algorithms].

[PBM86] A. P. Prudnikov, Yu. A. Brychkov et O. I. Marichev : *Integrals and Series*, volume Volume 1–6. Gordon and Breach, 1986. 798 pages. First edition in Moscow, Nauka, 1981.

[Pec09] Lucien Pech : Algorithmes pour la sommation et l'intégration symboliques. Mémoire de D.E.A., École Normale Supérieure, novembre 2009. 22 pp.

[Pet92] Marko Petkovšek : Hypergeometric solutions of linear recurrences with polynomial coefficients. *J. Symbolic Comput.*, 14(2-3):243–264, 1992.

[PWZ96] Marko Petkovšek, Herbert S. Wilf et Doron Zeilberger : $A = B$. A K Peters Ltd., Wellesley, MA, 1996.

[Reb98] Luc Rebillard : *Etude théorique et algorithmique des series de Chebychev, solutions d'équations différentielles holonomes*. Thèse de doctorat, Institut national polytechnique de Grenoble, 1998.

[Riv90] Theodore J. Rivlin : *Chebyshev polynomials*. Pure and Applied Mathematics (New York). John Wiley & Sons Inc., New York, second édition, 1990. From approximation theory to algebra and number theory.

[Ron79] André Ronveaux : Polynômes orthogonaux dont les polynômes dérivés sont quasi orthogonaux. *C. R. Acad. Sci. Paris Sér. A-B*, 289(7):A433–A436, 1979.

[RZ06] L. Rebillard et H. Zakrajšek : Recurrence relations for the coefficients in hypergeometric series expansions. *In* Ilias Kotsireas et Eugene Zima, éditeurs : *Computer Algebra 2006. Latest Advances in Symbolic Algorithms*, pages 158–180. World Scientific, 2006.

[RZ07] L. Rebillard et H. Zakrajšek : Recurrence relations for the coefficients in hypergeometric series expansions. *In Computer algebra 2006*, pages 158–180. World Sci. Publ., Hackensack, NJ, 2007.

[RZG95] A. Ronveaux, A. Zarzo et E. Godoy : Recurrence relations for connection coefficients between two families of orthogonal polynomials. *J. Comput. Appl. Math.*, 62(1):67–73, 1995.

[Sal05] Bruno Salvy : D-finiteness : Algorithms and applications. *In* Manuel Kauers, éditeur : *ISSAC'05*, pages 2–3. ACM Press, 2005. Abstract for an invited talk.

[Sch80] J. T. Schwartz : Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, 27(4):701–717, 1980.

[Sho94] Victor SHOUP : Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

[Sny66] Martin Avery SNYDER : *Chebyshev methods in numerical approximation*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1966.

[SP95] N. J. A. SLOANE et S. PLOUFFE : *The Encyclopedia of Integer Sequences*. Academic Press, 1995. See also the up to date electronic version at the URL : `http://www.research.att.com/~njas/sequences/`.

[SS71] A. SCHÖNHAGE et V. STRASSEN : Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[Sta80] Richard P. STANLEY : Differentiably finite power series. *European Journal of Combinatorics*, 1(2):175–188, 1980.

[Sto10] A. STOTHERS : *On the Complexity of Matrix Multiplication*. Thèse de doctorat, University of Edinburgh, 2010.

[Str69] Volker STRASSEN : Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[Str72] Volker STRASSEN : Evaluation of rational functions. *In Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 1–10, 187–212. Plenum, New York, 1972.

[SZ94] Bruno SALVY et Paul 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. http://algo.inria.fr/libraries/papers/gfun.html.

[Sze75] Gábor SZEGÖ : *Orthogonal Polynomials*, volume XXIII de *Colloquium Publications*. American Mathematical Society, Providence, R.I., fourth édition, 1975.

[Szw92] Ryszard SZWARC : Linearization and connection coefficients of orthogonal polynomials. *Monatsh. Math.*, 113(4):319–329, 1992.

[Tha64] Henry C. THACHER, Jr. : Conversion of a power to a series of Chebyshev polynomials. *Communications of the ACM*, 7(3):181–182, 1964.

[Tre07] Lloyd N. TREFETHEN : Computing numerically with functions instead of numbers. *Math. Comput. Sci.*, 1(1):9–19, 2007.

[Vas12] V. VASSILEVSKA WILLIAMS : Multiplying matrices faster than Coppersmith-Winograd. *In 44th ACM Symp. on Theory of Computing (STOC'12)*, 2012. To appear. Preliminary version available at http://cs.berkeley.edu/~virgi/matrixmult.pdf.

[Ver66] Arun VERMA : Certain expansions of the basic hypergeometric functions. *Math. Comp.*, 20:151–157, 1966.

[Wat44] G. N. WATSON : *A Treatise on the Theory of Bessel Functions*. Cambridge University Press, Cambridge, England, 1944.

[Wim84] J. Wimp : *Computation with Recurrence Relations.* Pitman, Boston, 1984.

[WL62] Jet Wimp et Yudell L. Luke : Expansion formulas for generalized hypergeometric functions. *Rend. Circ. Mat. Palermo (2)*, 11:351–366, 1962.

[Zah76] Ray V.M. Zahar : A mathematical analysis of Miller's algorithm. *Numerische Mathematik*, 27(4):427–447, 1976.

[Zei90] Doron Zeilberger : A holonomic systems approach to special functions identities. *J. Comput. Appl. Math.*, 32(3):321–368, 1990.