Marc MEZZAROBBA
sous la direction de Bruno SALVY

Génération automatique de procédures numériques pour les fonctions D-finies

Rapport de stage de Master 2
Master parisien de recherche en informatique

---

février-juillet 2007
version 1.1, octobre 2007

http://www.marc.mezzarobba.net/info/m2/

TABLE DES MATIÈRES

INTRODUCTION

On attend d'un logiciel de calcul formel généraliste qu'il permette d'évaluer à grande précision les expressions mathématiques qu'il manipule. Cela comprend en particulier les fonctions élémentaires sin, cos, exp, log, arctan, etc., les fonctions spéciales usuelles, notamment en physique mathématique, comme la fonction Γ, les fonctions de Bessel et de Hankel, d'Airy, de Whittaker, la fonction W de Lambert, les intégrales de Fresnel... Actuellement, des logiciels comme Maple, Mathematica ou Maxima disposent de ces fonctionnalités d'évaluation numérique, mais utilisent du code spécialisé, écrit à la main, pour « chaque » fonction. Ainsi, Maple 10 reconnaît plus de 120 fonctions transcendantes, et possède autant de sous-routines de `evalf`, d'efficacité variable, pour en calculer des approximations.

Or des algorithmes relativement simples permettent en principe de traiter de façon unifiée de larges classes de fonctions spéciales, sans perdre grand-chose du point de vue de la complexité algorithmique par rapport à des méthodes plus *ad hoc*. Cependant, en pratique, ces algorithmes ont principalement été considérés comme des « recettes » pour fabriquer des routines numériques spécialisées. Il est naturel de chercher à faire produire ces routines par un programme, de façon entièrement automatisée, à partir d'une spécification la plus légère possible de la fonction à évaluer.

Ce document est une version modifiée (qui vise à corriger un certain nombre d'erreurs et à clarifier quelques passages par rapport à la « version 1.0 » soumise au jury) du rapport du travail effectué sous la direction de Bruno Salvy durant mon stage de deuxième année de master, au sein de l'équipe Algo de l'Inria Rocquencourt. Le sujet s'inscrivait dans cette problématique. Plus précisément, je me suis concentré sur l'évaluation *efficace* à *grande précision* des *fonctions holonomes* par *scindage binaire*.

Efficace s'entend ici aussi bien asymptotiquement qu'en pratique, et par rapport à la taille du résultat, c'est-à-dire au nombre de chiffres demandés. De ce point de vue, les méthodes itératives classiques, à convergence « polynomiale », de l'analyse numérique (par exemple la méthode de Runge-Kutta d'ordre 4 pour l'intégration des équations différentielles) ont une complexité exponentielle. Les algorithmes auxquels nous allons nous intéresser sont de nature radicalement différente : on vise des algorithmes *quasi-optimaux*, c'est-à-dire de complexité linéaire, à des facteurs logarithmiques près, en la taille cumulée des données fournies en entrée et du résultat.

En accord avec cela, les *grandes précisions* ciblées vont de quelques centaines à quelques centaines de milliers de chiffres. Il s'agit en priorité d'obtenir du code efficace en pratique pour ce genre de précisions, non de battre des records — de l'ordre quant à eux d'une centaine de millions à quelques milliards de chiffres décimaux pour les « constantes remarquables » comme e, $\sqrt{2}$, π, $\zeta(3)$... S'il serait gênant, en interaction avec des manipulations formelles,

d'introduire une limite arbitraire à la précision que l'on peut atteindre, une estimation numérique à 10^{-20} près suffit très largement dans beaucoup d'applications. Un exemple de domaine où plusieurs dizaines de milliers de chiffres sont nécessaires est la théorie des nombres, où des expériences statistiques sur les développements décimaux de certaines constantes sont utilisées pour tester des conjectures de transcendence. Plus modestement, une centaine de chiffres sont utiles pour « reconnaître » une constante à partir d'une approximation numérique à l'aide de l'algorithme LLL.

Les *fonctions holonomes* [**Lip89, BCS07**] sont les fonctions analytiques complexes solutions d'équations différentielles linéaires à coefficients polynomiaux. Cette classe recouvre un grand nombre de fonctions usuelles : par exemple, les fonctions cosinus hyperbolique et arctangente, la fonction d'erreur erf, les fonctions d'Airy Ai et Bi, les fonctions de Bessel I_ν satisfont toutes des équations du premier ou du second ordre, respectivement

$$\cosh''(z)-\cosh(z)=0, \qquad (1+z^2)\arctan'(z)=1, \qquad \operatorname{erf}''(z)+2z\operatorname{erf}(z)=0,$$
$$\operatorname{Ai}''(z)-z\operatorname{Ai}(z)=0, \qquad \operatorname{Bi}''(z)-z\operatorname{Bi}(z)=0, \qquad z^2 I_\nu''(z)+zI_\nu'(z)+\nu^2 I_\nu(z)=z^2.$$

Plus généralement, des fonctions holonomes arbitraires apparaissent naturellement en combinatoire comme séries génératrices [**FS07, Odl95**], ou encore en théorie des nombres [**CC87, CC90**].

La méthode d'évaluation numérique étudiée ici s'inscrit dans une algorithmique générale sur les fonctions holonomes *représentées implicitement*, par une équation différentielle accompagnée de conditions initiales. Cet usage d'une équation comme structure de données est analogue à la représentation usuelle des nombres algébriques par leur polynôme minimal sur $\mathbb{Q}$. Il est adéquat pour de nombreuses opérations, qui vont jusqu'à la vérification, voire le calcul automatique d'identités non triviales entre expressions faisant intervenir des fonctions holonomes [1]. Cela est remarquable au vu des résultats d'indécidabilité qui parsèment le calcul formel, comme le théorème classique de Richardson sur le test d'équivalence à zéro des expressions formelles, ou, plus près des objets que nous manipulerons, celui de Denef et Lipshitz sur la convergence des séries solutions d'équations différentielles.

THÉORÈME 0.1 (Richardson [**Ric68**]). — Soit $f(x)$ une expression construite par composition à partir des constantes π et $\ln 2$ et des fonctions $+$, $-$, $\times$, exp, sin, $|\cdot|$. Il n'existe pas d'algorithme qui décide si $(\exists x\in\mathbb{R})(f(x)\geqslant 0)$, ni si $(\forall x\in\mathbb{R})(f(x)=0)$.

COROLLAIRE 0.2 (Denef et Lipshitz [**DL89**]). — Soit $f\in\mathbb{Q}[[z]]$ une série formelle donnée comme l'unique solution d'une équation différentielle algébrique $P(z,f,f',\dots,f^{(r)})=0$ vérifiant des conditions initiales spécifiées. Le problème de déterminer si le rayon de convergence de f est ou non $\geqslant 1$ est indécidable.

Utiliser les équations différentielles comme structure de données centrale représente un point de vue intermédiaire entre celui des principaux logiciels généralistes mentionnés plus haut, qui manipulent des expressions formelles (à la sémantique parfois peu claire) construites à partir des noms des fonctions usuelles, modulo certaines règles de simplification ; et celui de l'analyse effective [**vdH05, vdH06**], qui vise à traiter une classe de fonctions la plus large possible, avec pour seule limite la (semi-)décidabilité des problèmes.

Le *scindage binaire*, enfin, consiste à calculer un long produit d'entiers, ou d'objets (matrices, polynômes...) à coefficients entiers, en le découpant en sous-produits équilibrés de façon à contrôler la taille des résultats intermédiaires. Sommairement, l'algorithme qui va

1. Comme les systèmes polynomiaux, et les algorithmes (d'élimination notamment) qui vont avec, étendent les nombres algébriques et leur algorithmique, l'intérêt de la représentation de fonctions par équations fonctionnelles linéaires s'étend à des « systèmes holonomes » de plusieurs variables, qui peuvent mêler des équations de plusieurs natures, différentielles et de récurrence par exemple [**CS98, Chy98, BCS07**]. C'est dans ce cadre que les résultats les plus impressionnants apparaissent. Mais ces notions sortent du cadre de ce rapport.

nous occuper dans la suite fonctionne de la façon suivante : soit à évaluer une fonction holonome $f(z)$ (donnée par une équation différentielle et des conditions initiales en 0) en un point $z_1 \in \mathbb{C}$ situé dans le disque de convergence de sa série de Taylor en 0. On commence par calculer un entier n tel que n termes de ladite série prise en $z = z_1$ fournissant le résultat à la précision requise. On déduit de l'équation différentielle une relation de récurrence vérifiée par les sommes partielles $S_k(z_1)$ de cette série. La récurrence fournit une expression de $S_n(z_1)$ recherchée comme (un coefficient d') un produit de matrices appliqué à un vecteur de conditions initiales :

$$\begin{bmatrix} S_n(z_1) \\ \vdots \\ S_{n+r-1}(z_1) \end{bmatrix} = A(n-1)\cdots A(1)\cdot A(0)\cdot \begin{bmatrix} S_0(z_1) \\ \vdots \\ S_{r-1}(z_1) \end{bmatrix},$$

produit que l'on calcule par scindage binaire. On peut ensuite appliquer le même procédé aux dérivées successives de f, de façon à obtenir une nouvelle famille de « conditions initiales » définissant f, exprimées cette fois en z_1. Ces dernières permettent à leur tour de calculer les valeurs de $f, f', f'', \ldots$ en un point z_2, éventuellement situé en-dehors du disque de convergence du développement en série au voisinage de 0 utilisé auparavant. Et ainsi de suite : on obtient un algorithme de *prolongement analytique numérique* des solutions d'une équation différentielle linéaire à coefficients polynomiaux, qui permet d'évaluer une fonction holonome en un point quelconque de sa surface de Riemann.

L'idée du scindage binaire est classique (voir par exemple [**KS73**]), et sans doute quasiment aussi vieille que la multiplication rapide (Karatsuba, 1963). Son application à l'évaluation des fonctions holonomes a été redécouverte à de nombreuses reprises, avec divers degrés de généralité [**Bre76a, Ber87, CC90, Kar91, vdH97, HP97**]. Le rapport de recherche *HAKMEM* [**BGS72**, §178] cite déjà, sans preuve ni algorithme, un résultat de complexité voisin de celui obtenu par l'algorithme esquissé ci-dessus, et l'attribue à Schroeppel et Salamin. Le cas général (avec étude explicite du prolongement analytique et de la complexité en fonction du point d'évaluation) a été traité en détail par Chudnovsky et Chudnovsky [**CC90**] puis indépendamment par van der Hoeven [**vdH99**]. Ces deux articles ont servi de point de départ à mon stage. Le second a l'avantage, important dans l'optique d'une automatisation complète, de donner une version « complètement effective » de l'algorithme, qui comprend le calcul des bornes nécessaires au contrôle de la précision [(2)]. L'un comme l'autre mentionnent que l'algorithme se généralise à « l'évaluation » en une singularité, généralisation étudiée ensuite plus soigneusement par van der Hoeven [**vdH01, vdH07**]. Van der Hoeven a également donné dans [**vdH01**] une seconde méthode, plus simple, pour le calcul des bornes, méthode qui est raffinée au chapitre 5 du présent rapport.

Cet algorithme a en revanche peu été étudié en pratique, au moins dans le cas général. Sa complexité vis-à-vis des autres paramètres que la précision n'est pas non plus parfaitement claire. Il s'agissait donc pendant ce stage de rassembler un certain nombre d'idées bien connues en théorie, de les implémenter et de confronter le résultat à la pratique, avec comme objectif à long terme la génération complètement automatisée de procédures numériques efficaces couvrant une gamme de points d'évaluation et de précisions la plus vaste possible.

Le titre de ce document, repris du sujet du stage, est un peu trompeur : la production de code n'aura finalement été qu'un aspect mineur de mon travail, faute (en l'état actuel des algorithmes) d'informations pertinentes, indépendantes du point d'évaluation, à précalculer. Si la fonction `diffeqtoproc` du module Maple que j'ai développé répond formellement au sujet en renvoyant des procédures Maple pour l'évaluation numérique de fonctions holonomes,

2. Notre perspective ici est donc assez différente de celle de [**CC90**], qui donne, plus qu'un algorithme général, une méthode à instancier dans des cas particuliers, et s'écarte un peu de celle de [**vdH99**], dans la mesure où van der Hoeven inscrit l'évaluation numérique des fonctions holonomes dans un cadre plus général d'analyse effective.

il s'agit principalement d'une commodité pour l'utilisateur, sans gain sur la complexité (mis à part dans le cas particulier de l'évaluation dans le disque de convergence, où le gain pratique reste minime).

Par ailleurs, certaines parties du travail réalisé s'écartent un peu du fil conducteur de l'évaluation numérique. Elles partagent avec lui les deux idées centrales que sont le scindage binaire et le développement d'une algorithmique uniforme pour les fonctions holonomes.

Contributions. — La principale réalisation de ce stage est le module Maple `NumGfun` (annexe A), qui implémente une bonne partie des algorithmes décrits ici. En particulier, la fonction `ac_eval` semble être la première implémentation générale de l'algorithme de prolongement analytique numérique par scindage binaire [3]. Sur le plan théorique, les majorants « fins » du chapitre 5 (sans prétendre à la moindre originalité du point de vue mathématique !) améliorent leurs analogues issus de [**vdH01, vdH03**].

Pour le reste, la présentation de la complexité du scindage binaire (§1.4 et chapitre 2) est personnelle [4]. Le recensement des algorithmes de produit de petites matrices sur un anneau commutatif (§2.2.4) et l'étude expérimentale de l'algorithme de Waksman (§2.2.2) sont simples et sans surprise, mais n'ont apparemment pas d'équivalent. Enfin, l'heuristique concernant les chemins de prolongement analytique (§4.3) constitue un début de généralisation de résultats de [**CC87, vdH99**], entrepris dans le but d'automatiser le choix de ces chemins.

Exemples. — Pour terminer sur une note concrète cette (longue) présentation, voici quelques exemples de calculs que permettent les algorithmes présentés ici. Tous ont été effectués à l'aide du logiciel développé dans le cadre de ce stage.

1. La fonction `nth_term` de `NumGfun`, fondée sur les algorithmes classiques présentés dans la première partie, calcule en moins de six secondes le 100 000e nombre de Catalan. C'est cinq fois moins de temps qu'il n'en faut à Maple pour évaluer la formule close
$$C_{100\,000} = \frac{1}{100\,001}\binom{200\,000}{100\,000}.$$
2. À l'aide de la formule des frères Chudnovsky
$$\frac{1}{\pi} = 12\sum_{k=0}^{\infty}\frac{(-1)^k(6k)!(13591409+545140134k)}{(3k)!(k!)^3 640320^{3k+3/2}},$$
sa variante `fnth_term` calcule un million de chiffres décimaux de π en moins de trente secondes. En comparaison, Maple, qui utilise la même formule, met près de sept minutes à évaluer `evalf[1000000](Pi)`.
3. La méthode d'évaluation numérique des fonctions holonomes du chapitre 4 calcule à 10 000 chiffres décimaux après la virgule les valeurs de e, $\arctan(1/2)$ ou encore $\operatorname{erf}(1/2)$ en une poignée de secondes.
4. L'algorithme de calcul de bornes du chapitre 5 permet de déterminer, à partir de l'équation différentielle ci-dessus, que la série de Taylor en 0 de $\arctan z$ est bornée terme à terme par celle de $1/(1-z)$. On en déduit, toujours automatiquement, que 808 termes suffisent à

3. Il semble que les frères Chudnovsky n'aient implémenté qu'un analogue en précision fixe de leur algorithme de prolongement analytique numérique, et leur code n'est apparemment pas disponible publiquement. Par ailleurs, Joris van der Hoeven a annoncé récemment avoir commencé à travailler sur le support des fonctions holonomes dans le logiciel Mathemagix, cependant le prolongement analytique rapide est absent des versions diffusées pour l'instant.

4. À ce sujet, si la traduction matricielle « économique » des récurrences (§2.3) est utilisée à bon escient dans des cas particuliers (par exemple dans [**CC90**]), je n'ai pas pu trouver de mention explicite de sa forme générale, ni des avantages de celle-ci. À l'inverse, plusieurs travaux en utilisent une variante moins efficace que la traduction naïve.

obtenir la valeur de $\arctan \frac{3}{4}$ à 10^{-100} près. Le véritable nombre de chiffres corrects donnés par cette approximation est 104.

5. À partir de la définition

$$\{(n+1)u(n+1) = (2n+3)u(n), u(0) = 1\},$$

le même algorithme détermine que la suite $u(n)$ est bornée par $(n+1)2^n$.

6. L'équation différentielle (tirée au hasard)

$$\{(z+1)(3z^2-z+2)\,y''' + (5z^3+4z^2+2z+4)y'' + (z+1)(4z^2+z+2)y' \\ + (4z^3+2z^2+5)y = 0, \quad y(0) = 1, y'(0) = i, y''(0) = 0\}$$

admet au voisinage de zéro une unique solution analytique, qui a une rayon de convergence voisin de 0,8. La fonction `dsolve()` de Maple n'en trouve pas d'expression plus simple que l'équation elle-même. Au moyen de l'algorithme de prolongement analytique numérique du chapitre 4, on calcule que le prolongement analytique le long du segment $[0,-1+i]$ de cette solution prend à l'extrémité la valeur

$$y(-1+i) \simeq -1.42105 - 1.28693i.$$

Il est possible de déterminer cette valeur à une précision de 10^{-500} en quelques dizaines de secondes.

D'autres exemples, plus détaillés, sont donnés en introduction des chapitres 1 et 4, ainsi que §5.6.

Plan. — Le reste de ce rapport s'organise comme suit. La première partie est consacrée au scindage binaire, qui, en plus d'être un outil essentiel pour la suite, présente un intérêt propre. Elle peut bien sûr se lire indépendamment du reste. Dans cette partie, le chapitre 1 présente l'algorithme de scindage binaire et son application au calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux, et analyse sa complexité. Le chapitre 2 décrit quelques algorithmes qui peuvent être couplés avec le scindage binaire pour améliorer la « constante » de sa complexité dans des cas particuliers.

La seconde partie se concentre sur l'évaluation numérique des fonctions holonomes. Elle débute par le chapitre 3, consacré à des rappels mathématiques sur celles-ci et plus généralement sur les solutions d'équations différentielles linéaires dans le plan complexe. Le chapitre 4 contient la description détaillée de l'algorithme de prolongement analytique esquissé ci-dessus. Enfin, au chapitre 5, nous nous attaquerons au problème de la précision des calculs, et plus largement du calcul automatique de bornes sur les fonctions holonomes, leurs coefficients et les restes de leurs développements en série.

Deux annexes décrivent l'utilisation de `NumGfun` et reprennent des comptes-rendus de séminaires écrits pendant ces quelques mois, qui complètent le texte du rapport en ouvrant sur des sujets connexes.

Notations

J'emploie dans la suite quelques notations personnelles, rares, ou usuelles mais avec des conventions fluctuantes. Elles sont regroupées dans le tableau 1. J'utilise $z!$, ainsi que $x^{\uparrow z}$, $x^{\downarrow z}$ et les coefficients binomiaux pour des arguments non entiers, en convenant que $z! = \Gamma(z+1)$ pour tout $z \in \mathbb{C} \setminus \mathbb{Z}^*_-$. Concernant la terminologie, sauf précision contraire, équation différentielle sous-entend linéaire à coefficients polynomiaux (eux-mêmes à coefficients complexes).

Notations générales :

$\tilde{O}(\cdot)$	$g(n)=\tilde{O}(f(n))$ s'il existe $k\in\mathbb{N}$ tel que $g(n)=O(f(n)\log^k(f(n))$
$\Omega(\cdot)$	$g(n)=\Omega(f(n))$ s'il existe $a\geqslant 0$ tel que $\forall n, \lvert g(n)\rvert \geqslant af(n)$
$\Theta(\cdot)$	$g(n)=\Theta(f(n))$ si $f(n)=O(g(n))$ et $f(n)=\Omega(g(n))$
log	logarithme en base 2, logarithme complexe
$x^{\uparrow n}$	factorielle montante (symbole de Pochhammer) $x^{\uparrow n}=\dfrac{(x+n-1)!}{(x-1)!}=x(x+1)\dots(x+n-1)$
$x^{\downarrow n}$	factorielle descendante $x^{\downarrow n}=\dfrac{x!}{(x-n)!}=x(x-1)\dots(x-n+1)$
$[\![i,j]\!]$	intervalle entier $\{i,\dots,j\}$
$\mathbf{1}[\cdot]$	$\mathbf{1}[P]=1$ si la proposition P est vraie, 0 sinon
$\mathbb{A}[[z]]$	séries formelles à une indéterminée sur un anneau $\mathbb{A}$
$\mathbb{A}((z))$	séries de Laurent formelles à une indéterminée $\sum_{n=-N}^{\infty} f_n z^n$ sur un anneau $\mathbb{A}$
∂	opérateur de dérivation des séries formelles
$\mathrm{ord}_P\,\mu$	ordre de μ comme racine du polynôme P
$\mathrm{M}(\cdot)$	fonction de multiplication (définition 1.3 page 15)
ω	exposant de l'algèbre linéaire (définition 1.5 page 17)
$\preccurlyeq$	relation de majoration sur les séries (définition 5.1 page 57)

Pour $f\in\mathbb{A}((z))$:

$[z^n]f(z)$	coefficient de z_n dans f
f_n	coefficient de z_n dans f (on croisera occasionnellement des familles $f_1,f_2,\dots$ de séries, mais le contexte et l'utilisation dans ce cas de la notation $[z^n]f$ devraient éviter toute ambiguïté)
$f_{i;j}$	tranche $f_{i;j}(z)=\sum_{k=i}^{j-1} f_k z^k$
$f_{;n}$	somme partielle $f_{;n}(z)=f_{0;n}(z)=\sum_{k=0}^{n-1} f_k z^k$
$f_{n;}$	reste $f_{n;}(z)=f_{n;\infty}(z)=\sum_{k=n}^{\infty} f_k z^k$
$\mathrm{val}\,f$	valuation $\mathrm{val}\,f=\min\{n\in\mathbb{Z}\,\vert\, f_n\neq 0\}$

TAB. 1. Notations

Configuration de test

La machine utilisée (sauf mention contraire) pour toutes les expériences de ce rapport est un ordinateur de bureau équipé d'un microprocesseur Intel Pentium 4 "Northwood" cadencé à 2.8 GHz, avec 512 Ko de mémoire cache (L2), et de 1 Go de mémoire vive. Le noyau est Linux 2.6.11. La version de Maple utilisée est Maple 10 ("IBM INTEL LINUX"). Les programmes C sont compilés avec gcc 3.4.3, et précisément

```
gcc -O3 -march=pentium4 -fomit-frame-pointer -lgmp -lm -std=c99
```

La bibliothèque d'arithmétique multiprécision utilisée, tant par Maple que dans les programmes C, est GMP 4.1.4, et plus précisément le binaire fourni avec Maple pour l'architecture "IBM_INTEL_LINUX/P4SSE2". Les fonctions les « moins expérimentales » écrites pendant ce stage sont (provisoirement) regroupées dans le module Maple `NumGfun` (voir annexe A) ; les exemples utilisent la version 0.1 de ce module. Celui-ci, de même que divers exemples donnés dans ce rapport, fait appel à la bibliothèque `algolib` pour Maple 10 (07-05-11), qui contient en particulier `gfun` 3.21 (mais la version 3.05 fournie avec Maple 10 devrait généralement donner les mêmes résultats). La ligne

```
> with(gfun): with(NumGfun):
```

qui importe les commandes de ces deux modules est sous-entendue au début de tous les exemples de sessions Maple.

I

AUTOUR DU SCINDAGE BINAIRE

CHAPITRE 1

LE SCINDAGE BINAIRE

Le scindage binaire est une technique pour effectuer rapidement de longs produits d'entiers, de rationnels, de nombres algébriques, ou encore de matrices. Après quelques préliminaires sur la complexité du produit de grands entiers et de matrices, ce chapitre présente l'idée générale du scindage binaire, puis sa principale application, à savoir « dérouler » une récurrences linéaire pour obtenir un terme lointain en complexité quasi-optimale, essentiellement proportionnelle à la taille du résultat.

Appliquées à des suites d'entiers issues de la combinatoire, ces méthodes permettent de « compter » de grosses structures. Par exemple, les nombres de Motzkin, qui dénombrent les arbres unaires-binaires de taille donnée, satisfont

$$(n+3)\,u(n+2) = 3n\,u(n) + (2n+3)\,u(n+1), \quad u(0)=0, u(1)=1, u(2)=1.$$

La session Maple suivante construit une procédure qui calcule les nombres de Motzkin, et l'utilise pour calculer les quelques premiers termes de la suite, puis le 100 000e.

```
> m := rectoproc_binsplit({(n+3)*u(n+2)=3*n*u(n)+(2*n+3)*u(n+1),
  u(0)=0,u(1)=1,u(2)=1},u(n)):
> seq(m(n),n=0..10);
```

$$0, 1, 1, 2, 4, 9, 21, 51, 127, 323, 835$$

```
> time(m(100000));
```

$$9.489$$

(On a $u(100\,000) = 20626\ldots81045$.) On peut calculer de même, toujours en une poignée de secondes, le nombre de permutations sans point fixe de 100 000 éléments ($10389\ldots0001$).

Plus près de ce qui nous intéresse ici, les mêmes idées fonctionnent sur des suites convergentes, permettant d'évaluer leurs limites. Ainsi, dès la précision de 300 chiffres décimaux, un petit programme utilisant la formule [**HP97, Yac07**]

$$\Gamma(z) = k^z e^{-k} \sum_{n=0}^{\infty} \frac{1}{z^{\uparrow(n+1)}} k^n + \int_{t=k}^{\infty} e^{-t} t^{z-1} \mathrm{d}t$$

évalue $\Gamma(5/3)$ (valeur qui intervient, par exemple, dans une écriture simple de la valeur en zéro des fonctions d'Airy) plus rapidement que la fonction intégrée de Maple :

```
> g := proc(s, prec)
     k := ceil(evalf((prec)*ln(10) + ln((prec)*ln(10)))):
     rec := {u(n+1) = k/(n+s+1)*u(n), u(0) = 1/s}:
     a := fnth_term(rec,u(n),6*k,prec+1,ndseries):
     evalf[prec+1](k^s*exp(-k)*a):
  end proc:
```

```
> p := 300: time(g(5/3,p)), time(evalf[p](GAMMA(5/3)));
```

$$0.159, 0.191$$

L'écart se creuse quand on augmente la précision :

```
> p := 3000: time(g(5/3,p)), time(evalf[p](GAMMA(5/3)));
```

$$1.656, 35.299$$

Bien souvent pourtant, le scindage binaire n'atteint pas la meilleure complexité asymptotique connue pour l'évaluation à grande précision de constantes. L'exemple le plus classique est celui du calcul de π : le calcul de n décimales par scindage binaire à l'aide de la formule de Machin (ou d'une de ses variantes, ou de la formule des frères Chudnovsky — seule la constante change) nécessite $\Theta(n\log^3 n\log\log n)$ opérations, tandis que l'algorithme de Brent-Salamin, fondé sur l'itération de la moyenne arithmético-géométrique, n'en demande que $\Theta(n\log^2 n\log\log n)$. Cependant, même à des précisions record, des variantes du scindage binaire peuvent s'avérer meilleures en pratiques, témoin les performances du programme `gmp-chudnovsky.c` de Hanhong Xue fourni sur le site web de GMP [**Gra07**]. Le problème de l'évaluation de constantes à plusieurs millions de chiffres, et les différents algorithmes applicables, sont décrits en détail dans le livre [**BB87**].

1.1. Préliminaires

1.1.1. Mesures de complexité. — On peut distinguer deux grands types d'estimations de complexité pour les algorithmes de calcul formel. La *complexité binaire* d'un algorithme ou d'un calcul est le nombre d'opérations logiques élémentaires qu'il effectue, exprimé généralement en fonction de la taille en bits des données d'entrée. La *complexité arithmétique* est le nombre d'opérations sur des objets d'une structure algébrique abstraite, par exemple d'additions et multiplications dans un anneau non spécifié, ou souvent tout simplement dans $\mathbb{Z}$. Elle se ramène (en ordre de grandeur) à la complexité binaire si les opérations arithmétiques que l'on compte sont des opérations en temps constant, sur des objets de taille bornée. Ainsi, la complexité arithmétique et la complexité binaire du produit de polynômes sur un corps fini sont du même ordre.

On peut donner une version plus formelle des définitions précédentes dans le modèle RAM (voir par exemple [**AHU74**]). Dans ce modèle, on considère une machine abstraite disposant d'une bande d'entrée, d'une bande de sortie, et d'une infinité de registres sur lesquels elle peut effectuer des opérations en temps constant. Complexité binaire et complexité arithmétique se distinguent alors essentiellement par la nature des registres : la complexité arithmétique est le temps de calcul d'une machine RAM dont les registres peuvent contenir des entiers quelconques [1] (ou des éléments d'une structure algébrique plus générale), pour la complexité binaire, les registres sont astreints à ne stocker que des bits. Pour certaines analyses fines de complexité, on ne peut plus se permettre l'approximation que l'accès à la mémoire est uniforme. Le modèle des machines de Turing à plusieurs bandes tel que défini dans [**SGV94**] (voir aussi [**Knu97a**, §2.6, p. 463-464]) peut alors donner une meilleure notion de complexité binaire.

Sauf mention contraire, les estimations de complexité données dans ce qui suit s'entendent en opérations binaires, et dans le pire des cas. Mais pour le type d'algorithme présentés ici, et sauf distribution très particulière des entrées, le cas le pire est le cas générique. Par ailleurs,

1. Cette variante de complexité arithmétique est souvent utilisée implicitement dans les domaines de l'informatique où, pour des tailles de problèmes raisonnables, les opérations arithmétiques concernent des entiers ou des flottants machine. Ce n'est plus un modèle adéquat de la complexité sur une machine réelle lorsqu'on s'intéresse, comme ici, à des algorithmes où les opérations sur les entiers représentent l'essentiel du coût du calcul.

dans certaines parties, nous nous attacherons à donner des complexités « sans $O(\cdot)$ », c'est-à-dire à expliciter (une borne sur) la constante précédant le terme dominant de l'expression de la complexité.

1.1.2. Grands entiers. — Par les algorithmes de l'école primaire, l'addition de deux entiers de taille n prend un temps $\Theta(n)$, leur multiplication, un temps $\Theta(n^2)$. Pour l'addition (hors parallélisme ou représentation redondante), c'est évidemment optimal. Concernant la multiplication, il est aujourd'hui classique que les algorithmes de multiplication à base de transformée de Fourier rapide permettent de multiplier les grands entiers en temps *essentiellement linéaire* — mais beaucoup plus élevé que celui nécessaire pour faire une addition.

Algorithmes. — Passons brièvement en revue les principaux algorithmes de multiplication rapide. Pour plus de détails, on pourra consulter les classiques [**Knu97b**, §4.3] et [**Ber01**].

Le plus simple est celui de Karatsuba : on remarque que le produit en base B

$$(aB+b)(cB+d) = ac\,B^2 + (ad+bc)\,B + bd$$

peut se calculer en trois multiplications de chiffres au lieu de quatre. Pour cela, on calcule $(a+b)(c+d) = ac+bd+(ad+bc)$ en « oubliant » la base, puis on en soustrait ac et bd, calculés séparément, pour obtenir le coefficient $ad+bc$. En appliquant récursivement cette méthode, on obtient une multiplication de complexité $O(n^{\log_2 3})$.

L'algorithme de Karatsuba s'applique encore mieux au produit de polynômes. Une interprétation moins naïve est alors de le voir comme un algorithme d'*évaluation-interpolation* : pour multiplier deux polynômes de degré 1, on les évalue en les trois points $0, 1, \infty$, on multiplie les valeurs obtenues, et on récupère par interpolation les coefficients du produit. L'idée se généralise à des polynômes de degré quelconque : à d fixé, on peut multiplier deux polynômes de degré d en $2d-1$ multiplications de scalaires non constants, et en faisant cela récursivement, on obtient un algorithme de complexité $O(n^{\log_d(2d-1)})$ pour le produit de polynômes de degré n. Ces algorithmes s'appellent des schémas de Toom-Cook. Ils s'adaptent à leur tour aux entiers.

Enfin, les algorithmes les plus rapides connus pour la multiplication de grands entiers sont ceux à base de transformée de Fourier rapide. Ils reposent eux aussi sur l'évaluation-interpolation : le point de départ est de remarquer que, si A est un polynôme de degré $n-1$ et ω une racine n-ième de l'unité, l'évaluation de A en les ω^j peut se voir comme la transformée de Fourier discrète $\hat{a}_k = \sum_{j=0}^{n-1} a_j \omega^{kj}$ du n-uplet $(a_j)_{j\in[\![0,n-1]\!]}$ des coefficients de A, que l'on sait calculer en $O(n\log n)$ opérations dans l'anneau de base. Une fois résolus les problèmes de trouver des anneaux disposant des racines de l'unité nécessaires, de contrôler la complexité binaire des opérations dans ces anneaux, et de gérer les retenues dans le cas des entiers, on peut aboutir aux résultats théoriques [2] suivants.

THÉORÈME 1.1 (Schönhage-Strassen [**SS71**]). — On peut multiplier deux entiers de taille inférieure ou égale à n en $O(n\log n\log\log n)$ opérations binaires.

Il s'agit là de la borne classique sur la complexité de la multiplication de grands entiers. Mais ce résultat a été amélioré récemment.

THÉORÈME 1.2 (Fürer [**Für07**]). — On peut multiplier deux entiers de taille inférieure ou égale à n en $O(n\log n\, 2^{O(\log^* n)})$ opérations binaires, où $\log^* n = \min\{k \mid \log^{\circ k} n \leqslant 0\}$ désigne la hauteur d'une tour d'exponentielles de valeur de l'ordre de n.

2. Cela ne signifie pas que les algorithmes correspondants sont inutilisables en pratique. Mais il faut garder en tête que les complexités sont exprimés en fonction de la *taille n* de l'entrée, or si n est de l'ordre de la taille cumulée, mesurée en bits, des mémoires vives de tous les ordinateurs personnels en service dans le monde en 2007 (estimation Forrester), $\log\log n \leqslant 6$...

Pour donner un sens précis à ces estimations, il est important de s'entendre sur le modèle de calcul utilisé. Les deux théorèmes précédents sont valables dans le modèle des machines de Turing à plusieurs bandes. Schönhage a étudié la complexité de la multiplication dans d'autres modèles de calcul : dans un modèle RAM (binaire) simple, elle descend à $O(n\log n)$; dans d'autres modèles, elle tombe à $O(n)$ [**Knu97b**, §4.3.3C, p. 311].

Réductions. — La multiplication rapide est une brique de base de bien des algorithmes efficaces — d'où son importance. Ainsi, pour de nombreuses opérations arithmétiques usuelles sur les entiers, par exemple l'inverse, la racine carrée, ou le calcul de pgcd, on dispose d'algorithmes rapides qui ramènent le calcul à un nombre constant ou (poly)logarithmique de multiplications.

Il est d'usage d'exprimer la complexité de ces algorithmes en fonction de la complexité $\mathrm{M}(n)$ de la multiplication d'entiers de taille n. Cela rend les estimations de complexité plus robustes vis-à-vis à la fois du modèle de calcul, des améliorations à venir de la complexité théorique de la multiplication, et du choix de l'algorithme. Cela permet aussi de comparer les constantes dans les estimations de complexité des algorithmes, en mesurant leur temps d'exécution par celui d'un nombre précis de multiplications. Conformément à cette habitude, nous supposerons dans toute la suite que $\mathrm{M}(n)$ est une *fonction de multiplication*, au sens de la définition suivante.

DÉFINITION 1.3. — On appelle fonction de multiplication une fonction croissante $\mathrm{M} : \mathbb{R}_+ \to \mathbb{R}_+$ telle que

(i) il existe un algorithme calculant le produit de deux entiers de taille n en au plus $\mathrm{M}(n)$ opérations binaires ;

(ii) pour tous p et q, $\mathrm{M}(p)+\mathrm{M}(q) \leqslant \mathrm{M}(p+q) \leqslant \mathrm{M}(p)+\mathrm{M}(q)+O(pq)$,

(iii) $n = o(M(n))$.

L'hypothèse $\mathrm{M}(p+q) \geqslant \mathrm{M}(p)+\mathrm{M}(q)$ est standard, elle permet de simplifier des expressions du type de $\sum_{i=0}^{\infty} \mathrm{M}\left(\frac{n}{2^i}\right)$, fréquentes dans l'analyse d'algorithmes basés sur la multiplication rapide. Le reste des points (ii) et (iii) est moins usuel (au moins explicitement), il autorise notamment à remplacer $\mathrm{M}(f(n))+O(f(n))$ par $\mathrm{M}(f(n))\,(1+o(1))$ (si $f(n) \to \infty$) et $\mathrm{M}(n+1)$ par $\mathrm{M}(n)+O(n)$.

Les deux exemples suivants de réductions à la multiplication d'entiers d'autres opérations arithmétiques nous seront utiles dans la suite. L'article de sythèse [**Ber04**] résume toute une gamme de résultats de ce type.

THÉORÈME 1.4. — (i) L'itération de Newton permet de calculer une approximation flottante du quotient, ou encore la division euclidienne (quotient et reste) de deux entiers de taille $\leqslant n$ en $O(\mathrm{M}(n))$ opérations binaires.

(ii) On peut calculer le plus grand commun diviseur de deux entiers de taille $\leqslant n$ en complexité binaire $O(\mathrm{M}(n)\log n)$.

On pourra consulter [**Knu97b**, §4.3.3, p. 312] ou [**AHU74**, §8.2] pour les détails du résultat sur la division. Celui concernant le pgcd est délicat, tant à prouver qu'à implémenter (voir cependant [**vzGG03**, chap. 11], [**Knu97b**, exercice 4.5.3.46] et [**Ber04**, §21–22] pour une discussion plus complète, et les références qui y sont données pour les preuves de différentes variantes de l'algorithme d'Euclide rapide). Il s'agit des meilleures complexités connues. La morale (en ce qui nous concerne !) est que les réductions au même dénominateur sont à éviter lors de calculs répétés avec des rationnels.

REMARQUE. — La discussion précédente se transporte *mutatis mutandis* au cas des opérations sur les polynômes denses, les complexités étant cette fois exprimées en nombre d'opérations dans l'anneau de base. Ainsi, on peut multiplier deux polynômes de degré n en

Chiffres décimaux	10^2	10^3	10^4	10^5	10^6	10^7
Maple	0,02	0,07	0,68	19,18	351,34	
C	$< 0{,}01$	$< 0{,}01$	0,50	15,70	205,30	3 331,30

TAB. 2. Coût du produit d'entiers en millisecondes, avec GMP. *Moyenne sur 100 multiplications d'entiers aléatoires de* 10^j *chiffres décimaux. L'absence de résultat pour Maple en taille* 10^7 *s'explique par la lenteur du générateur pseudo-aléatoire* `rand()`.

$\mathrm{M}_{\mathbb{A}[x]}(n) = O(n \log n \log\log n)$ opérations arithmétiques, et de nombreuses opérations sur les polynômes de degré n ou les séries formelles tronquées à l'ordre n ont une complexité de la forme $\tilde{O}(\mathrm{M}_{\mathbb{A}[x]}(n))$.

Pratique. — Pour importante qu'elle soit, l'implémentation efficace des algorithmes rapides sur les entiers n'est pas chose facile [3]. Les algorithmes quasi-linéaires sont utiles en pratique, et pas uniquement pour des nombres de taille astronomique, mais à la condition expresse de disposer d'une implémentation de bonne qualité : dans une implémentation naïve, les seuils où ces algorithmes deviennent intéressants sont trop élevés. Les bibliothèques d'arithmétique multiprécision les plus rapides disposent d'implémentations en assembleur des routines de base pour leurs différentes architectures cibles, et utilisent des algorithmes hybrides qui choisissent l'une ou l'autre des méthodes présentées plus haut (ou d'autres) en fonction de la taille des entiers à multiplier et de l'architecture.

Comme exemples de bibliothèques relativement connues fournissant une arithmétique entière en précision arbitraire, on peut citer LIP, LiDIA, CLN, NTLn, ou MpNT. J'ai utilisé GMP [**Gra07**], qui est une des meilleures implémentations disponibles de l'arithmétique rapide sur les entiers.

Pour les multiplications de nombres de taille voisine, GMP utilise suivant la taille l'algorithme naïf, l'algorithme de Karatsuba, l'algorithme de Toom-Cook à trois branches ou un algorithme par transformée de Fourier rapide adapté de celui de Schönhage-Strassen. Les multiplications déséquilibrées sont effectuées en découpant la plus grande des opérandes en blocs de la taille de la plus petite. Le tableau 2 donne quelques exemples de temps d'exécution pour le produit d'entiers aléatoires, d'une part avec Maple 10, et d'autre part dans un programme C utilisant « directement » (la couche `mpz` de) GMP. La configuration matérielle et logicielle est celle décrite à la fin de l'introduction [4], page 10. On observe que la complexité « au chronomètre » est nettement sous-quadratique, et qu'un régime proche du linéaire est atteint assez rapidement. On voit aussi que l'utilisation de Maple représente un surcoût tolérable mais qui peut devenir critique sur de gros problèmes.

Pour ce qui est des autres opérations utilisées dans ce travail, GMP dispose, outre plusieurs algorithmes de division « naïfs » pour les cas de base, d'une division rapide de complexité $O(\mathrm{M}(n)\log n)$ et de routines spécifiques pour la division exacte d'entiers et la division de nombres de taille voisine (petit quotient). La division par l'algorithme de Newton n'est pas implémentée dans la version actuelle. GMP n'implémente pas non plus pour l'instant d'algorithme de calcul de pgcd sous-quadratique (mais un algorithme quadratique de très bonne « constante », relativement efficace en pratique). C'est un argument supplémentaire pour fuir les calculs de pgcd de grands entiers dans les programmes.

3. Voir à ce sujet le récent rapport d'implémentation d'une multiplication par FFT améliorée basée sur celle de GMP par Gaudry, Kruppa et Zimmerman [**GKZ07**].

4. Sur une autre machine de test, globalement moins puissante mais équipée d'un processeur AMP Athlon XP (cadencé à 2 GHz, et de 512 Mo de mémoire vive) et de logiciels plus récents (Debian GNU/Linux "lenny", binaire GMP 4.2.1 fourni par la distribution), le même programme C est généralement 15% à 25% plus rapide pour 100 000 chiffres et plus, et jusqu'à 40% en certaines tailles.

1.1.3. Complexité asymptotique du produit matriciel. — Comme le produit d'entiers, le produit matriciel est une opération suffisamment fondamentale (et de complexité imparfaitement connue, tant en théorie qu'en pratique) pour que l'on préfère faire de son coût un paramètre dans les estimations de complexité.

DÉFINITION 1.5. — On appelle exposant de l'algèbre linéaire (sur $\mathbb{K}$), et l'on note ω, la borne inférieure des réels κ tels qu'il existe un algorithme qui effectue le produit de deux matrices carrées de taille n à coefficients dans un corps $\mathbb{K}$ en $O(n^\kappa)$ opérations dans $\mathbb{K}$.

Une fois de plus, cette définition est motivée par le fait que la plupart des opérations de base de l'algèbre linéaire ont essentiellement la même complexité que le produit matriciel [**BCS97**, chap. 16].

Il est clair que $2 \leqslant \omega \leqslant 3$. Historiquement, la première amélioration par rapport à l'algorithme naïf (et la seule, ou presque, représentant un gain en pratique) est la célèbre observation de Strassen que le produit de matrices carrées de taille 2, sur un anneau quelconque, peut être fait en 7 multiplications. L'algorithme de Strassen, après sa simplification par Winograd, peut s'écrire :

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} t_2+t_3-t_6-t_7 & t_4+t_6 \\ t_5+t_7 & t_1-t_3-t_4-t_5 \end{bmatrix} \tag{1}$$

où

$$\begin{aligned} t_1 &= (a_{11}+a_{21})(b_{11}+b_{12}) & t_4 &= a_{11}(b_{12}-b_{22}) \\ t_2 &= (a_{12}+a_{22})(b_{21}+b_{22}) & t_5 &= (a_{21}+a_{22})b_{11} \\ t_3 &= (a_{11}-a_{22})(b_{11}+b_{22}) & t_6 &= (a_{11}+a_{12})b_{22} \\ & & t_7 &= a_{22}(b_{21}-b_{11}). \end{aligned}$$

Appliquée récursivement en faisant des produits par blocs, cette formule montre que $\omega \leqslant \log_2 7 < 2{,}81$.

Il existe un grand nombre d'autres schémas « à la Strassen » pour multiplier des matrices de petite taille, voir par exemple [**Knu97b**, exercice 4.6.4.12]. En taille 2×2, les 7 multiplications et 15 additions de l'algorithme de Strassen-Winograd sont optimales [**HK71, Pro76**]. C'est la seule taille pour laquelle le nombre minimal de multiplications scalaires requises par le produit de matrices carrées est connu. En taille 3×3, la meilleure borne inférieure est actuellement de 19 multiplications [**Blä03**], tandis que le meilleur algorithme en demande 23 (Laderman, généralisé à $n^3-(n-1)^2 = 1, 7, 23, 55, 109\ldots$ multiplications en taille n par Sýkora). Chacun de ces schémas, appliqué récursivement, donne un algorithme de produit matriciel rapide. Le premier à apporter une amélioration — minime — de l'exposant par rapport à Strassen est un algorithme de Laderman pour la taille 20×20. Mais d'autres techniques se sont montrées plus efficaces pour abaisser la complexité théorique du produit matriciel.

THÉORÈME 1.6 (Coppersmith-Winograd [**CW90**]). — L'exposant ω de l'algèbre linéaire sur un corps quelconque vérifie $\omega < 2{,}376$.

Il s'agit là du meilleur exposant connu à ce jour. On peut trouver une preuve d'une version légèrement affaiblie de ce théorème dans le livre [**BCS97**, chap. 15].

Tous ces résultats « passent » en complexité binaire pour les matrices à coefficients entiers (car les algorithmes mentionnés sont *bilinéaires*, voir §1.4.1). Un produit de matrices d'ordre n à coefficients entiers de taille $\leqslant t$ peut donc se faire en $O(n^\omega \mathrm{M}(t))$ opérations.

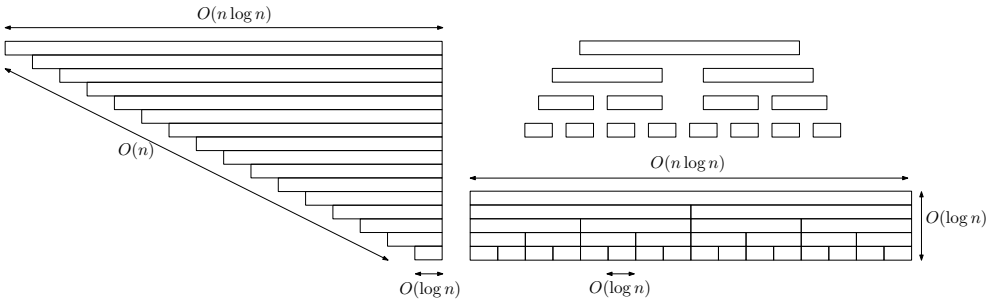


FIG. 1. Structure du calcul d'un produit $a_n \cdots a_1$ où $a_j = O(n)$ et taille des résultats intermédiaires. *À gauche, la méthode naïve, à droite, le scindage binaire.*

1.2. Le scindage binaire

Pour obtenir une bonne complexité binaire dans les algorithmes fondés sur l'arithmétique des grands entiers, il est crucial de *contrôler la croissance des coefficients*, ce qui revient souvent à mettre à profit la multiplication rapide en *faisant des produits équilibrés*, dont les opérandes ont à peu près la même taille. Le scindage binaire (en anglais *binary splitting*) consiste à « diviser pour régner » une suite d'opérations, généralement un long produit $a_n a_{n-1} \cdots a_1$, pour atteindre ces deux objectifs. Diverses variantes de cette idée toute simple permettent, sans changer leur complexité arithmétique, d'abaisser drastiquement la complexité binaire d'un grand nombre de calculs.

Commençons par l'exemple le plus classique, celui du calcul de $n!$. La méthode naïve consiste à multiplier 2 par 3, puis le résultat par 4, et ainsi de suite. Sa complexité arithmétique est linéaire. Mais la croissance des produits intermédiaires est dramatique : à la k-ième étape, on multiplie $k!$, de taille $\lceil \log(k!) \rceil \sim k \log k$, par $k+1$. Ce produit déséquilibré peut se faire en $\Theta(k\,\mathrm{M}(\log k))$ opérations binaires en découpant $k!$ en $\Theta(k)$ blocs de taille $\log k$. Le coût total en opérations binaires de l'algorithme naïf est donc (les constantes cachées par tous les Θ sont les mêmes)

$$\sum_{k=1}^{n-1} \Theta(k\,\mathrm{M}(\log k)) = \Theta(n^2\,\mathrm{M}(\log n)),$$

loin de la taille $\lceil \log(n!) \rceil \sim n \log n$ du résultat. (Sans hypothèse sur le produit déséquilibré, en additionnant simplement les tailles des résultats intermédiaires aux étapes $k \geqslant n/2$, on obtient la borne inférieure $\Omega(n^2 \log n)$.)

Le scindage binaire consiste à découper « au milieu » le produit

$$n! = 1 \times 2 \times \cdots \times \lfloor n/2 \rfloor \times \big(\lfloor n/2 \rfloor + 1\big) \times \cdots \times n$$

et à évaluer récursivement les deux moitiés suivant la même idée. Un peu plus formellement, on calcule $n! = P(1, n+1)$ par l'algorithme

$$P(i,j) := P\Big(\Big\lfloor \frac{i+j}{2} \Big\rfloor, j\Big) \cdot P\Big(i, \Big\lfloor \frac{i+j}{2} \Big\rfloor\Big) \quad \text{si } j > i$$
$$P(i,i) := i.$$

Prenons pour n une puissance de 2. Les 2^k appels récursifs de niveau k effectuent chacun un produit d'entiers de taille $\leqslant 2^{-k-1} n \log n)$ (figure 1), d'où un coût cumulé

$$O\left(\sum_{k=1}^{\log n} 2^k\,\mathrm{M}\left(\frac{n}{2^k}\log n\right)\right) = O\left(\sum_{k=1}^{\log n} \mathrm{M}(n \log n)\right) = O\big(\mathrm{M}(n\log n)\log n\big)$$

d'après les hypothèses sur le coût de la multiplication (définition 1.3). Si $\mathrm{M}(n) = \tilde{O}(n)$, cette complexité est quasi-optimale vis-à-vis de la taille du résultat.

REMARQUE. — À l'inverse, le scindage binaire n'est intéressant que si l'on dispose d'une multiplication rapide. Avec une multiplication quadratique, son coût

$$\simeq \sum_{k=1}^{\log n} 2^{k-1}\,\mathrm{M}\Big(\frac{n}{2^k}\log n\Big) = \sum_{k=1}^{\log n} 2^{k-1}\Big(\frac{n}{2^k}\log n\Big)^2 \sim (n\log n)^2$$

est essentiellement le même que celui de l'algorithme naïf. À l'autre extrême, le scindage binaire ne change rien à la complexité arithmétique du calcul : il n'apporte rien dans un contexte où la multiplication prend un temps constant (entiers ou flottants machine, calculs modulaires). Mais un algorithme pas de bébé-pas de géant dû à Strassen [**Str77**] fait descendre la complexité arithmétique du calcul de $n!$ à $\tilde{O}(\sqrt{n})$ en utilisant la multiplication rapide de polynômes.

REMARQUE. — Bien entendu, le scindage binaire a son analogue sur les polynômes. En particulier, le calcul par scindage binaire d'un long produit de polynômes de degré 1, où l'on conserve tous les produits intermédiaires, est à la base de l'algorithme usuel d'évaluation-interpolation rapide.

1.3. N-ième terme de suites récurrentes

La méthode utilisée à la section précédente pour le calcul de $n!$ se généralise à toute suite solution d'une récurrence linéaire à coefficients polynomiaux [5]. Considérons la récurrence

$$a_s(n)\,u_{n+s} + a_{s-1}(n)\,u_{n+s-1} + \cdots + a_0(n)\,u_n = 0, \tag{2}$$

où les $a_j \in \mathbb{Q}[n]$. Supposons a_s sans pôle entier, de sorte que s valeurs successives déterminent une solution. Soit (u_n) une suite de rationnels satisfaisant (2), spécifiée par les s conditions initiales $u_0, \ldots, u_{s-1}$. On se propose de calculer un terme u_N, avec $N \gg 1$, de la suite (u_n).

On réécrit (2) sous forme matricielle d'ordre 1, en séparant numérateurs et dénominateur pour éviter les manipulations de rationnels conduisant à des calculs de pgcd [6] :

$$\begin{bmatrix} u_{n+1} \\ \vdots \\ u_{n+s-1} \\ u_{n+s} \end{bmatrix} = \frac{1}{q(n)} \begin{bmatrix} & q(n) & & \\ & & \ddots & \\ & & & q(n) \\ p_0(n) & p_1(n) & \dots & p_{s-1}(n) \end{bmatrix} \begin{bmatrix} u_n \\ \vdots \\ u_{n+s-2} \\ u_{n+s-1} \end{bmatrix} \tag{3}$$

où $p_0, \ldots, p_{r-1}, q \in \mathbb{Z}[n]$ et

$$\forall j \in [\![0, r-1]\!], \quad \frac{p_j(n)}{q(n)} = -\frac{a_j(n)}{a_s(n)}.$$

Notons $A(n)$ la matrice de l'équation (3), et $U_n = (u_n, \ldots, u_{n+s-1})$. On a

$$U_N = \frac{A(N-1)A(N-2)\ldots A(0)}{q(N-1)\,q(N-2)\ldots q(0)} U_0. \tag{4}$$

Les produits au numérateur et au dénominateur de (4) peuvent se calculer séparément par scindage binaire.

Soient d le maximum des degrés des p_j et de q, et h celui des tailles en bits de leurs coefficients. (Si $\hat{h}$ est une borne sur les tailles des numérateurs et dénominateurs des coefficients des a_j, on a $h \leqslant (s+1)(d+1)\hat{h}$.) La taille de l'entier $p_j(n)$ vérifie

$$\log|p_j(n)| \leqslant d\log n + h + \log(d+1)$$

5. L'algorithme pas de bébé-pas de géant mentionné plus haut se généralise de la même façon. Ces deux algorithmes, connus auparavant au moins dans des cas particuliers, ont été explicités en toute généralité par les frères Chudnovsky [**CC90**, §2].

6. Comme nous allons le voir, un seul calcul de pgcd de la taille du résultat a un coût du même ordre de grandeur que l'algorithme complet en séparant numérateurs et dénominateur.

et de même pour $q(n)$. Un produit de matrices carrées d'ordre s fait croître la taille des entrées de $O(\log s)$ par rapport à la somme de celles des opérandes. Par récurrence, la taille des entrées de la matrice $P(N-1)\cdots A(0)$ (et, au terme $\log s$ près, du dénominateur $q(N-1)\cdots q(0)$) est

$$\leqslant N(d\log N + h + \log(d+1) + \log s) = O(N(d\log N + h + \log s)), \tag{5}$$

et cet ordre de grandeur est atteint pour certaines suites (u_n).

L'analyse du coût du scindage binaire se poursuit essentiellement comme dans le cas de la factorielle (voir §1.4), et aboutit à une complexité

$$O\big(s^\omega \mathrm{M}(N(d\log N + h + \log s))\log N\big) \tag{6}$$

pour le numérateur et

$$O\big(\mathrm{M}(N(d\log N + h))\log N\big)$$

pour le dénominateur. Il faut y ajouter l'évaluation des polynômes p_j et q, qui demande $O(d\,\mathrm{M}(d\log n + h))$ opérations par polynôme et par valeur de n, soit

$$O(sdN\,\mathrm{M}(d\log N + h))$$

opérations au total, et la construction de la matrice A, qui en prend $O(sd\,\mathrm{M}(h))$. Si l'on cherche une approximation flottante du résultat, on peut ajouter les divisions finales, qui peuvent être effectuées en $O(s^2\,\mathrm{M}(N(d\log N + h))$ opérations d'après le théorème 1.4. En supposant $d = o(\log N)$ et $\log s = O(\log N)$, l'estimation (6) se simplifie en

$$O\big(s^\omega \mathrm{M}(N(d\log N + h))\log N\big) \tag{7}$$

et représente le terme dominant de la complexité.

Sous les mêmes hypothèses, en comparant avec (5), on voit que cette complexité est quasi-optimale vis-à-vis de N, h et d, mais que la dépendance en s est bien plus importante que celle du résultat. L'écart diminue si l'objet que l'on souhaite calculer est la matrice de l'application $U_0 \mapsto U_N$ et non le seul terme u_N.

Un avantage, en revanche, du fait que l'on calcule toute la matrice est que l'on peut réutiliser le résultat pour calculer un terme plus lointain de la suite. C'est particulièrement appréciable si (u_n) est une suite convergente dont on cherche à approximer la limite. Dans le même ordre d'idée, le scindage binaire se prête bien au calcul parallèle.

En comparaison, l'algorithme naïf, qui déroule la récurrence en exprimant u_{n+s} en fonction de $u_n, \ldots, u_{n+s-1}$, prend un temps $O(sN^2\,\mathrm{M}(d\log N + h))$.

1.4. Algorithme général et complexité

Nous avons vu des exemples d'utilisation du scindage binaire sur des entiers et des matrices d'entiers. On peut calculer de même des produits d'éléments de $\mathbb{Z}[i]$ ou de divers $\mathbb{Z}[\alpha]$, de matrices de tels objets, etc. Plus généralement, l'algorithme de scindage binaire s'applique dans toute $\mathbb{Z}$-algèbre A libre et de type fini comme $\mathbb{Z}$-module : pour calculer un produit (non commutatif) quelconque

$$P(i,j) = a_{j-1}\cdots a_{i+1}a_i, \quad (j > i)$$

d'éléments de A par scindage binaire, on pose $m = \lfloor (i+j)/2 \rfloor$, on calcule récursivement $P(m,j)$ et $P(i,m)$, et on forme leur produit $P(i,j) = P(m,j)\cdot P(i,m)$. En présence d'un produit de rationnels, on sépare numérateurs et dénominateurs et on fait le produit dans $\mathbb{Z}^2$. De même pour une $\mathbb{Q}$-algèbre de dimension finie quelconque, que l'on remplace par une $\mathbb{Z}$-algèbre « de même structure » en ajoutant 1 à son rang.

CONVENTION. — Dans toute cette section, $\mathbb{A}$-module signifie $\mathbb{A}$-module libre de type fini, et $\mathbb{A}$-algèbre, $\mathbb{A}$-algèbre qui, comme $\mathbb{A}$-module, est libre de type fini.

Calculons la complexité du scindage binaire dans ce cadre. Ce point de vue peut sembler démesurément abstrait eu égard à la complexité toute relative du résultat, mais il a le mérite faire le lien avec des concepts classiques de complexité algébrique. Je le trouve par ailleurs pratique pour estimer de façon unifiée les constantes dans les complexités de différentes instances du scindage binaire (cf. chapitre suivant), voire pour chercher la meilleure variante pour des problèmes précis. (Une alternative peut-être plus simple serait de se limiter aux sous-algèbres des matrices d'entiers.)

1.4.1. Complexité quadratique. — La principale caractéristique d'une algèbre que nous allons utiliser pour y estimer le coût du scindage binaire est sa complexité quadratique.

DÉFINITION 1.7 ([**FZ77**, §4], voir aussi [**AL04**, p. 205]). — Soient $\mathbb{A}$ un anneau commutatif, M, N, P des $\mathbb{A}$-modules (libres de type fini). On suppose les éléments de M, N, P représentés par leurs coordonnées dans une base fixée. Soit $\varphi : M \times N \to P$ une application bilinéaire.

(i) La complexité multiplicative de φ est le nombre minimal de multiplications d'éléments non constants de $\mathbb{A}$ dans un calcul d'évaluation (*straight-line program*) qui calcule $\varphi(m,n)$ par une suite d'opérations dans l'anneau $\mathbb{A}$.

(ii) Un tel algorithme est dit quadratique si pour toute multiplication $c \leftarrow a \times b$ d'éléments non constants de $\mathbb{A}$, les opérandes a et b s'expriment comme des formes linéaires en m et n [7]. La complexité quadratique de φ est le nombre minimal de multiplications d'éléments non constants de $\mathbb{A}$ dans un algorithme quadratique qui calcule φ.

La complexité multiplicative, resp. quadratique, d'une $\mathbb{A}$-algèbre est celle de sa multiplication.

Autrement dit, pour calculer $p = \varphi(m,n)$, un algorithme quadratique évalue un certain nombre de formes linéaires de m et n, multiplie certains des résultats obtenus, et en tire les coefficients de p à nouveau en appliquant des formes linéaires. Les évaluations de formes linéaires peuvent faire intervenir des multiplications par des éléments constants de $\mathbb{A}$, qui ne sont pas comptées. La définition se justifie par le fait que la complexité multiplicative et la complexité quadratique sont indépendantes des bases dans lesquelles les arguments sont représentés. La complexité quadratique d'une $\mathbb{A}$-algèbre de rang d est $\leqslant d^2$ (c'est le coût de l'utilisation d'une table de multiplication).

Cependant, ces notions ne disent rien sur la croissance des coefficients du résultat par rapport à ceux des paramètres.

DÉFINITION 1.8. — Soit $M \simeq \mathbb{Z}^r$ un $\mathbb{Z}$-module *muni d'une base*. On appelle hauteur d'un élément de M le maximum des tailles (nombres de chiffres binaires) de ses coefficients.

Un changement de base fait croître la hauteur d'une constante additive (qui dépend des bases, mais pas du vecteur). La définition d'un algorithme quadratique et l'hypothèse $\mathrm{M}(n) = O(n^2)$ (définition 1.3) conduisent au résultat suivant.

PROPOSITION 1.9. — Soit $\varphi : M \times N \to P$ une application bilinéaire de complexité quadratique L entre $\mathbb{Z}$-modules (dont les éléments sont représentés sur des bases fixées). Si $m \in M$ et $n \in N$ sont de hauteur $\leqslant h$, on peut calculer $\varphi(m,n)$ en $L\mathrm{M}(h) + O(h)$ opérations binaires.

Si $\varphi : M \times N \to P$ est une application bilinéaire et si $h(x)$ désigne la hauteur d'un élément,

$$h(\varphi(a,b)) \leqslant h(a) + h(b) + K \tag{8}$$

où K dépend de φ et du choix des bases, mais pas de (a,b).

7. Il n'est pas demandé que a ou b dépende *seulement de m* ou *seulement de n*. Un algorithme qui vérifie cette contrainte supplémentaire est dit bilinéaire [**FZ77**] ou normal [**Knu97b**, §4.6.4]. L'algorithme de Waksman (§2.2.2) est un exemple d'algorithme quadratique non bilinéaire.

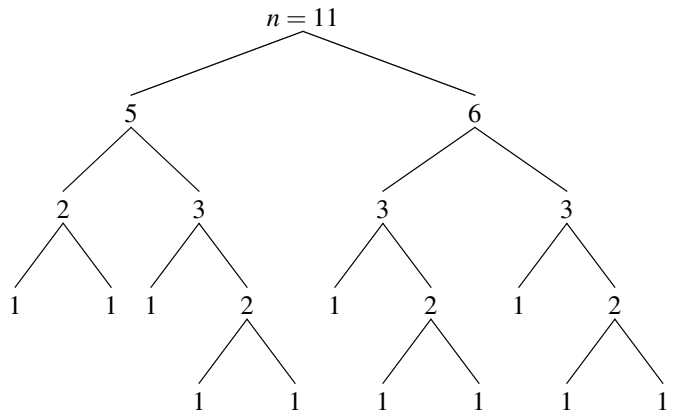


FIG. 2. Diviser-pour-régner déséquilibré. *La profondeur de l'arbre est* $\lceil \log n \rceil$, *le nombre de nœuds est* $n-1$, *et la somme des étiquettes de chaque ligne vaut n. Si le coût de traitement d'un nœud est son étiquette, le coût total est* $\leqslant n\lceil \log n \rceil$. *Si le coût d'un noœud est le maximum des étiquettes de ses fils, le coût total est* $\leqslant n\lceil \log n \rceil + n$

1.4.2. Coût du scindage binaire. — Soit A une $\mathbb{Z}$-algèbre (libre et de type fini comme module) munie d'une base. On suppose les éléments de A représentés dans cette base. Notons L la complexité quadratique de A, et K une constante satisfaisant (8) pour la multiplication de A.

PROPOSITION 1.10. — Soient $a_1, \dots, a_n$ des éléments de A de hauteur bornée par h. Le calcul par scindage binaire du produit

$$a_n a_{n-1} \cdots a_1$$

demande un nombre d'opérations binaires

$$C(n) \leqslant L\mathrm{M}\Big(\frac{h+K}{2} n \log n\Big)\big(1+o(1)\big) \tag{9}$$

quand $n \to \infty$ (A, et donc K et L, étant fixés).

Démonstration. — Le coût $C(n)$ de l'algorithme et la hauteur $H(n)$ du résultat satisfont les récurrences

$$C(n) \leqslant C\Big(\Big\lfloor \frac{n}{2} \Big\rfloor\Big) + C\Big(\Big\lceil \frac{n}{2} \Big\rceil\Big) + L \cdot \mathrm{M}\Big(H\Big(\Big\lceil \frac{n}{2} \Big\rceil\Big)\Big) + O(H(n)) \tag{10}$$

$$H(n) \leqslant H\Big(\Big\lfloor \frac{n}{2} \Big\rfloor\Big) + H\Big(\Big\lceil \frac{n}{2} \Big\rceil\Big) + K \tag{11}$$

avec $C(1) = 0$, $H(1) = h$.

Par récurrence immédiate, $H(n) \leqslant nh + (n-1)K$. Ainsi

$$C(n) \leqslant C\Big(\Big\lfloor \frac{n}{2} \Big\rfloor\Big) + C\Big(\Big\lceil \frac{n}{2} \Big\rceil\Big) + L \cdot \mathrm{M}\Big(\Big\lceil \frac{n}{2} \Big\rceil (h+K)\Big) + \alpha n(h+K) \tag{12}$$

pour une certaine constante α. Soit $T_j(n)$ le multi-ensemble des indices à profondeur j de l'arbre de la récurrence (10) (figure 2) — formellement,

$$T_0(n) = \{n\} \quad \text{et} \quad T_{j+1}(n) = \Big\{ \lceil k/2 \rceil, \lfloor k/2 \rfloor \;\Big|\; k \in T_j(n) \cap [\![1, \infty]\!] \Big\}$$

pour $j \in \mathbb{N}$. On a $T_j(n) = \varnothing$ si $j > \lceil \log n \rceil$, et pour tout j, $\sum_{k \in T_j(n)} k \leqslant n$. La suite (d'indice n)

$$\sum_{j=0}^{\infty} \sum_{k \in T_j(n)} \big(L\mathrm{M}(\lceil k/2 \rceil (h+K)) + \alpha k(h+K) \big)$$

est solution de (12), d'où

$$C(n) \leqslant L\mathrm{M}\left(\sum_{j=0}^{\infty} \frac{n+\left|T_j(n)\right|}{2}(h+K)\right) + \alpha n \lceil \log n \rceil (h+K)$$
$$\leqslant L\mathrm{M}\Big(\frac{1}{2}(h+K)(n\lceil \log n \rceil + n)\Big) + \alpha n \lceil \log n \rceil (h+K)$$

et le résultat. □

Une preuve très similaire utilisant $\mathrm{M}(a+b) \leqslant \mathrm{M}(a)+\mathrm{M}(b)+O(ab)$ donne la variante

$$C(n) \leqslant L\mathrm{M}\Big(\frac{h+K}{2}n\Big)\log n + O(h^2 n \log n), \tag{13}$$

meilleure si $h = o(\log n)$.

Lorsque h tend vers l'infini avec n, le surcoût K devient négligeable. En combinant la proposition 1.10 à l'analyse de la section précédente, on obtient le corollaire suivant, qui couvre le cas des suites récurrentes à coefficients polynomiaux.

COROLLAIRE 1.11. — Soit $a(n) \in A[n]$ de degré $\leqslant d$ et hauteur $\leqslant h$ (sur la base $(n^k e_j)_{j,k}$, où $(e_j)_j$ désigne la base de A). La « factorielle » $a(n)a(n-1)\cdots a(0)$ se calcule par scindage binaire en au plus

$$L\mathrm{M}\Big(\frac{1}{2}n(d\log^2 n + h\log n)\Big)\big(1+o(1)\big) \tag{14}$$

opérations binaires quand $n, h \to \infty$.

REMARQUE (D. Stehlé ? [8]). — Supposons $n = 2^t$,

$$\mathrm{M}(n) \sim \kappa n \log n \log\log n$$

(par exemple : l'important est que $\mathrm{M}(n) = \Omega(n\log n)$) et $\log h = o(\log n)$. Alors

$$C(n) \leqslant \sum_{k=1}^{\log n} 2^k (L+o(1))\,\mathrm{M}\Big(\frac{n}{2^k}h\Big)$$
$$\leqslant (L\kappa + o(1)) \sum_{k=1}^{\log n} 2^k \Big(\frac{nh}{2^k}\log\frac{nh}{2^k}\log\log(nh)\Big)$$
$$\leqslant (L\kappa + o(1)) \sum_{k=1}^{\log n} nh(\log n - k)\log\log n$$
$$\leqslant \frac{L}{4}\mathrm{M}(nh)(\log n)\big(1+o(1)\big).$$

ce qui améliore d'un facteur 2 les estimations (9), (13) et (14). De même, si $\mathrm{M}(n) \sim Cn^\alpha$ où $\alpha > 1$, on obtient $C(n) = O(\mathrm{M}(nh))$.

8. Voir `http://www.loria.fr/~zimmerma/talks/arctan.pdf`.

CHAPITRE 2

SUR LE PRODUIT AUX NŒUDS

Ce chapitre tente de donner quelques pistes pour estimer la « constante » de la complexité du scindage binaire dans un certain nombre de situations usuelles, et si possible pour l'abaisser. Il s'agit de voir dans chaque cas combien de multiplications d'entiers demande le produit élémentaire dont on effectue une longue suite par scindage binaire. L'espoir, dans l'optique de générer automatiquement du code, est de comprendre quelles simplifications de l'algorithme dans des cas particuliers peuvent être détectées automatiquement, pour perdre le moins possible par rapport à du code spécifique.

Les sections suivantes sont donc principalement un catalogue d'algorithmes (tantôt classiques, tantôt moins connus) pour calculer en peu de multiplications d'entiers des produits de « petites » structures de taille fixée. Les situations examinées sont celles qui interviennent pour le calcul de termes de suites récurrentes : produit de petites matrices à coefficients dans $\mathbb{Z}$, cas particuliers des matrices issues de récurrences elles-mêmes obtenues comme sommes partielles d'une autre suite récurrente, produit dans un corps de nombres ou dans son anneau des entiers. La présentation n'est pas aussi formelle, mais la plupart des résultats peuvent se voir comme des estimations de la complexité quadratique de diverses algèbres, et donc de la constante L dans les expressions (9) et (14) de la complexité du scindage binaire.

En combinant les résultats de ce chapitre avec les calculs de complexité du précédent, on aboutit (entre autres) aux estimations suivantes.

PROPOSITION 2.1. — Soit $\mathbb{K} = \mathbb{Q}(\alpha)$ un corps de nombres. Soit $(u_n) \in \mathbb{K}^{\mathbb{N}}$ donnée par une récurrence linéaire d'ordre s, à coefficients dans $\mathbb{Z}[\alpha, n]$ eux-mêmes de degré $\leqslant d$ et de hauteur $\leqslant h$.

(i) On peut calculer un terme u_n de la suite en au plus

$$\big((2[\mathbb{K}:\mathbb{Q}]-1)\,\mathrm{MM}(s)+1\big)\,\mathrm{M}\Big(\frac{1}{2}n(d\log^2 n + h\log n)\Big)(1+o(1))$$

opérations binaires quand $n, h \to \infty$ (tous les autres paramètres étant fixés). Dans cette estimation, on peut prendre $\mathrm{MM}(s)$ égal à

$$n^2\left\lceil\frac{n}{2}\right\rceil + (2n-1)\left\lfloor\frac{n}{2}\right\rfloor \simeq \frac{n^3}{2} + n^2 - \frac{n}{2}$$

ou encore, pour les petites valeurs de s, aux valeurs données dans la dernière ligne du tableau 3.

(ii) Avec les mêmes conventions, on peut calculer une somme partielle $\sum_{k=0}^{n-1} u_k$ de la série de terme général (u_n) en la même complexité, où l'on remplace $\mathrm{MM}(s)$ par $\mathrm{MM}(s) + s^2 + s + 1$.

COROLLAIRE 2.2. — Soit $(u_n)_n$ une suite d'éléments de $\mathbb{Q}[i]$ donnée par une récurrence linéaire d'ordre s, à coefficients polynomiaux eux-mêmes de degré $\leqslant d$ et de hauteur h. On peut calculer un terme u_N en au plus

$$(L+o(1))\,\mathrm{M}\left(n(d\log^2 n + h\log n)\right)$$

opérations binaires quand $n,h\to\infty$, où L est donné par le tableau suivant.

s	1	2	3	4	5	6
L	2	11	35	70	140	212

2.1. Produit dans un corps de nombres

Les algorithmes du chapitre précédent s'appliquent dans les extensions finies de $\mathbb{Q}$: on peut calculer par scindage binaire un produit d'éléments d'un corps de nombres, ou encore dérouler par scindage binaire une récurrence à coefficients algébriques. Un exemple d'application est l'évaluation, en un point algébrique donné par son polynôme minimal, d'un polynôme de degré élevé dont les coefficients satisfont une récurrence.

On s'intéresse alors à la complexité du scindage binaire quand le nombre et la hauteur des algébriques à multiplier tendent vers l'infini, l'extension où sont faits les calculs étant fixée. On souhaite cependant améliorer la constante de cette complexité asymptotique, en limitant sa dépendance en le degré de l'extension. L'idée pour cela est de diminuer le nombre de produits d'entiers (de taille $\sim h$) auxquels on ramène un produit d'algébriques (de hauteur h).

Soit donc $\mathbb{K}=\mathbb{Q}(\alpha)\subset\mathbb{C}$ un corps de nombres de degré d. Du point de vue informatique, on suppose $\mathbb{K}$ donné par le polynôme minimal P (unitaire) de α, et ses éléments représentés par leurs coordonnées sur la base $(1,\alpha,\ldots,\alpha^{d-1})$. Effectuer un produit dans $\mathbb{K}$ se ramène ainsi à multiplier deux polynômes de degré $<d$ modulo P.

Comme mentionné §1.1.2, *il existe un algorithme* qui, pour tout d, multiplie deux polynômes de degrés $\leqslant d$ en $O(d\log d\log\log d)$ opérations arithmétiques. La division euclidienne de polynômes se ramène à un nombre constant de produits — dont il resterait à contrôler la complexité binaire dans le cas de $\mathbb{Z}[X]$.

Cependant, *pour tout* d, il existe un algorithme qui multiplie deux polynômes de degré $<d$ à coefficients dans $\mathbb{Z}$ en $2d-1$ multiplications d'entiers non constants : « l'algorithme de Toom-Cook » convient ! Autrement dit, il suffit d'évaluer les polynômes en $2d-1$ points (ce qui est une opération linéaire), de multiplier les valeurs obtenues ($2d-1$ multiplications) et de reconstuire le résultat (linéaire). De même, pour multiplier deux éléments de $\mathbb{Q}[X]/\langle P\rangle$, on en multiplie des représentants dans $\mathbb{Q}[X]$, et l'on réduit le résultat modulo P (c'est encore une opération linéaire). Le cas le plus important pour ce qui nous intéresse est celui de $\mathbb{Z}[i]$, où l'on peut calculer un produit en 3 multiplications entières par l'analogue suivant de l'algorithme de Karatsuba :

$$(x+iy)(x'+iy')=(u-v)+i(w-u-v)\quad\text{où}\quad u=xx',v=yy',w=(x+y)(x'+y').$$

En fait, Fiduccia et Zalcstein [**FZ77**] et Winograd [**Win77**] ont montré que si $\mathbb{K}$ est un corps de caractéristique nulle, la complexité multiplicative (sur $\mathbb{K}$) de l'algèbre $\mathbb{K}[X]/\langle Q\rangle$ est exactement $2\deg Q-k$, où k désigne le nombre de facteurs irréductibles distincts de Q (voir aussi [**Knu97b**, §4.6.4, p. 513–514], qui résume ces résultats et en prouve une version faible). La construction d'algorithmes « à la Toom-Cook » efficaces en pratique a été étudiée récemment par Bodrato et Zanoni [**BZ07**]. Ils expliquent comment construire des séquences d'instructions qui accélèrent la phase d'interpolation. Les représentations alternatives des nombres algébriques et les algorithmes correspondants sont traités dans [**Coh93**, chap. 4].

2.2. Produit matriciel sur un anneau commutatif

Passons au problème analogue pour des matrices d'entiers, ou plus généralement d'éléments d'un anneau commutatif.

Pour pouvoir s'appliquer récursivement, par blocs, et se traduire par un gain sur l'exposant de la complexité asymptotique du produit matriciel, les schémas « à la Strassen » de multiplication de matrices de taille n fixée en moins de n^3 multiplications scalaires mentionnés §1.1.3 supposent que les matrices sont à coefficients dans un anneau non commutatif. En levant cette restriction, on peut trouver des schémas utilisant encore nettement moins de multiplications. Ainsi, les algorithmes de Winograd et de Waksman permettent de multiplier des matrices à coefficients dans un anneau commutatif en $n^3/2$ multiplications environ. Ils sont intéressants quand le produit sur l'anneau de base est coûteux, et notamment pour de petites matrices de grands entiers.

2.2.1. Algorithme de Winograd. — Soit à calculer un produit $(c_{ij}) = (a_{ij}) \times (b_{ij})$ de matrices carrées de taille n. On commence par regrouper les termes consécutifs deux par deux dans l'expression d'un coefficient du résultat en fonction de ceux des opérandes :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = \sum_{t=1}^{\lfloor n/2 \rfloor} (a_{i,2t-1} b_{2t-1,j} + a_{i,2t} b_{2t,j}) + \mathbf{1}[n \bmod 2 = 1]\, a_{in} b_{nj}. \tag{15}$$

À partir de l'identité (qui utilise la commutativité !)

$$(a+b')(a'+b) = aa' + bb' + ab + a'b', \tag{16}$$

l'expression précédente se réécrit

$$c_{ij} = \underbrace{\left(\sum_{t=1}^{\lfloor n/2 \rfloor} (a_{i,2t-1} + b_{2t,j})(a_{i,2t} + b_{2t-1,j}) \right)}_{\text{(S1)}} - \left(\underbrace{\sum_{t=1}^{\lfloor n/2 \rfloor} a_{i,2t-1} a_{i,2t}}_{\text{(S2)}} + \underbrace{\sum_{t=1}^{\lfloor n/2 \rfloor} b_{2t-1,j} b_{2t,j}}_{\text{(S3)}} \right) + \mathbf{1}[n \bmod 2 = 1]\, a_{in} b_{nj}.$$

L'observation cruciale est que les deux sommes (S2) et (S3) sont indépendantes respectivement de j et de i. Pour calculer les valeurs de tous les c_{ij} par la formule précédente, il suffit donc d'évaluer (S1) pour n^2 valeurs de (i, j), et (S2) et (S3) pour n valeurs de i ou j chacune. On peut donc calculer tous les coefficients du résultat en

$$n^2 \left\lfloor \frac{n}{2} \right\rfloor + 2n \left\lfloor \frac{n}{2} \right\rfloor + (n \bmod 2) n^2 = n^2 \left\lceil \frac{n}{2} \right\rceil + 2n \left\lfloor \frac{n}{2} \right\rfloor \simeq \frac{n^3}{2} + n^2$$

multiplications.

2.2.2. Algorithme de Waksman. — Waksman [**Wak70**] a proposé une variante de l'algorithme de Winograd qui économise environ $n/2$ multiplications, au prix de l'introduction de divisions par 2 (qui peuvent être effectuées en temps linéaire, voire en temps constant). Seul change le calcul des sommes (S2) et (S3). Waksman les regroupe, écrit leur terme général sous la forme

$$\frac{1}{2}\big((a_{i,2t-1} + b_{2t,j})(a_{i,2t} + b_{2t-1,j}) + (a_{i,2t-1} - b_{2t,j})(a_{i,2t} - b_{2t-1,j})\big)$$

(le premier terme « est connu » : c'est le terme général de (S1)), et remarque que les relations linéaires entre ces expressions pour les différentes valeurs de i et j (à t fixé) permettent de les calculer toutes en en connaissant $2n-1$.

```
proc(a, b, c)
  local tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8,
    tmp9, tmp10, tmp11, tmp12, tmp13, tmp14, tmp15,
    tmp16, tmp17, tmp18;
  tmp1 := (a[1,1] + b[2,1])*(a[1,2] + b[1,1]);
  tmp3 := (a[1,1] + b[2,2])*(a[1,2] + b[1,2]);
  tmp5 := (a[1,1] + b[2,3])*(a[1,2] + b[1,3]);
  tmp7 := (a[2,1] + b[2,1])*(a[2,2] + b[1,1]);
  tmp9 := (a[2,1] + b[2,2])*(a[2,2] + b[1,2]);
  tmp11 := (a[2,1] + b[2,3])*(a[2,2] + b[1,3]);
  tmp13 := (a[3,1] + b[2,1])*(a[3,2] + b[1,1]);
  tmp15 := (a[3,1] + b[2,2])*(a[3,2] + b[1,2]);
  tmp17 := (a[3,1] + b[2,3])*(a[3,2] + b[1,3]);
  tmp2 := tmp1 + (a[1,1] - b[2,1])*(a[1,2] - b[1,1]);
  tmp4 := tmp3 + (a[1,1] - b[2,2])*(a[1,2] - b[1,2]);
  tmp6 := tmp5 + (a[1,1] - b[2,3])*(a[1,2] - b[1,3]);
  tmp10 := tmp9 + (a[2,1] - b[2,2])*(a[2,2] - b[1,2]);
  tmp18 := tmp17 + (a[3,1] - b[2,3])*(a[3,2] - b[1,3]);
  tmp8 := tmp10 - tmp4 + tmp2;
  tmp12 := tmp10 - tmp4 + tmp6;
  tmp14 := tmp18 - tmp6 + tmp2;
  tmp16 := tmp18 - tmp6 + tmp4;
  c[1,1] := tmp1 - iquo(tmp2,2) + a[1,3]*b[3,1];
  c[1,2] := tmp3 - iquo(tmp4,2) + a[1,3]*b[3,2];
  c[1,3] := tmp5 - iquo(tmp6,2) + a[1,3]*b[3,3];
  c[2,1] := tmp7 - iquo(tmp8,2) + a[2,3]*b[3,1];
  c[2,2] := tmp9 - iquo(tmp10,2) + a[2,3]*b[3,2];
  c[2,3] := tmp11 - iquo(tmp12,2) + a[2,3]*b[3,3];
  c[3,1] := tmp13 - iquo(tmp14,2) + a[3,3]*b[3,1];
  c[3,2] := tmp15 - iquo(tmp16,2) + a[3,3]*b[3,2];
  c[3,3] := tmp17 - iquo(tmp18,2) + a[3,3]*b[3,3]
end proc
```

FIG. 3. Algorithme de Waksman en taille 3

L'algorithme complet peut être décrit de la façon suivante. Pour chaque $t \in [\![1, \lfloor n/2 \rfloor]\!]$, on calcule successivement :

$$\begin{aligned}
A(i,j,t) &:= (a_{i,2t-1} + b_{2t,j})(a_{i,2t} + b_{2t-1,j}), && i,j \in [\![1,n]\!] \\
B(1,j,t) &:= A(1,j,t) + (a_{1,2t-1} - b_{2t,j})(a_{1,2t} - b_{2t-1,j}), && j \in [\![1,n]\!] \\
B(i,i,t) &:= A(i,i,t) + (a_{i,2t-1} - b_{2t,i})(a_{i,2t} - b_{2t-1,i}), && i \in [\![2,n]\!] \\
B(i,j,t) &:= B(i,i,t) - B(1,i,t) + B(1,j,t), && i \in [\![2,n]\!], j \in [\![1,n]\!] \setminus \{i\}.
\end{aligned}$$

On en déduit pour tous i et j

$$c_{ij} = \sum_{t=1}^{\lfloor n/2 \rfloor} \left(A(i,j,t) - \frac{1}{2} B(i,j,t) \right) + \mathbf{1}[n \bmod 2 = 1]\, a_{in} b_{nj}.$$

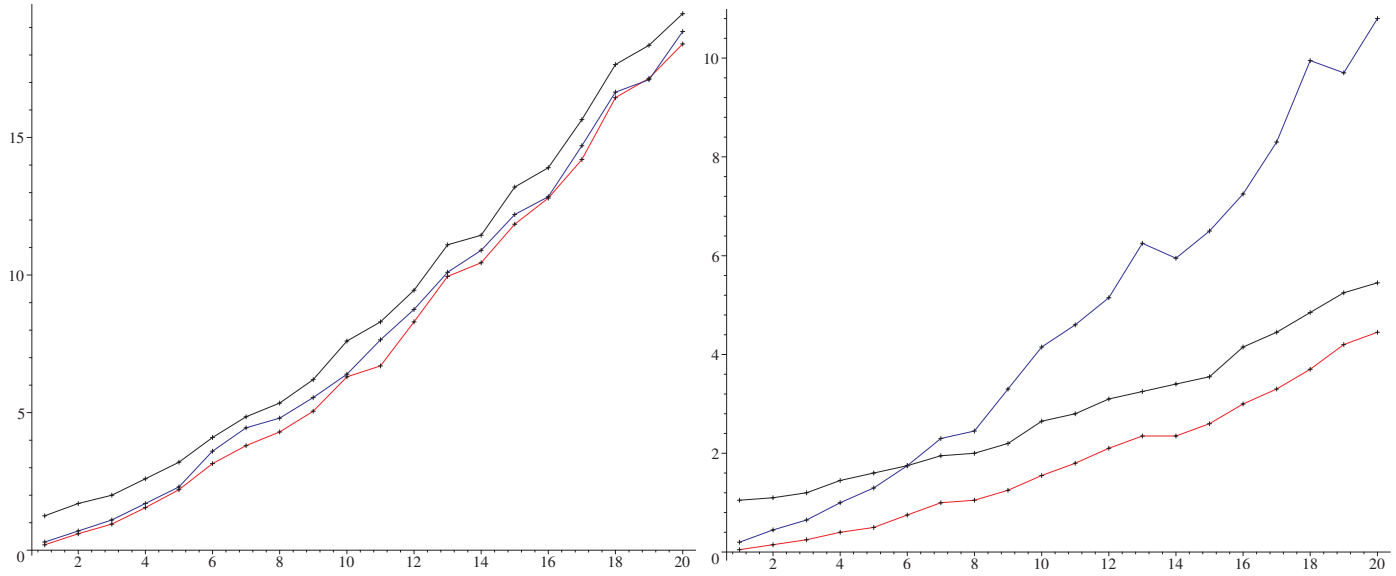
Il est aisé sous cette forme de vérifier que l'algorithme est correct, et qu'il effectue

$$(n^2 + 2n - 1) \left\lfloor \frac{n}{2} \right\rfloor + (n \bmod 2) n^2 = n^2 \left\lceil \frac{n}{2} \right\rceil + (2n-1) \left\lfloor \frac{n}{2} \right\rfloor \simeq \frac{n^3}{2} + n^2 - \frac{n}{2}$$

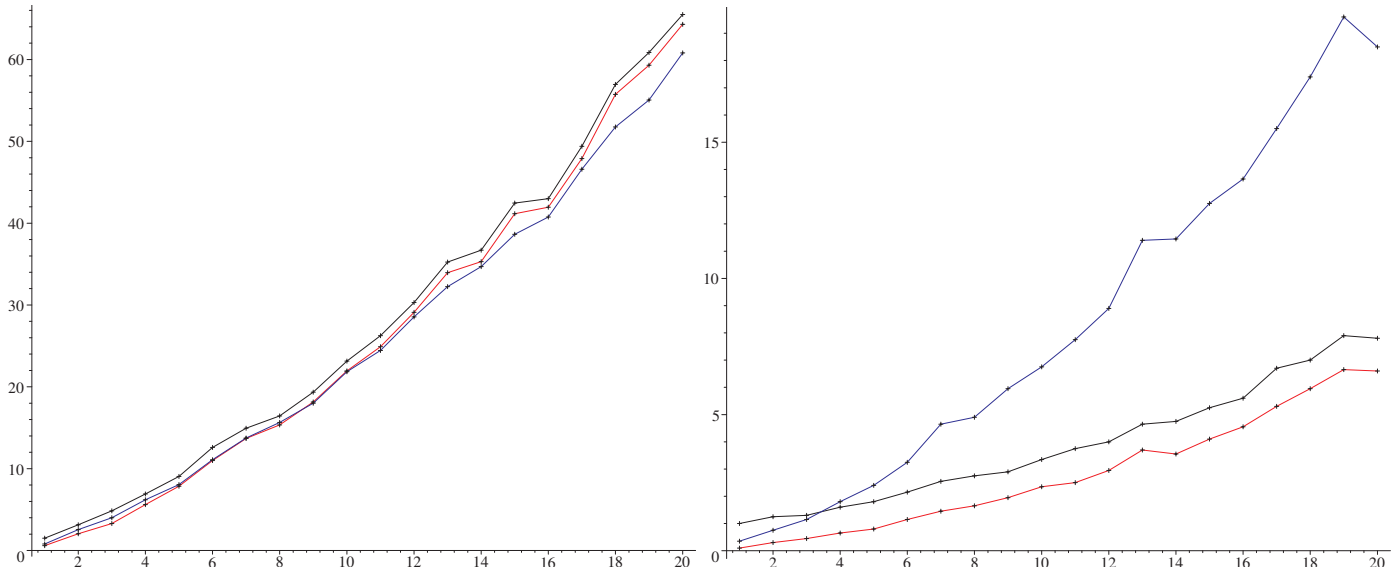
multiplications scalaires. En taille 2, contrairement à l'algorithme de Winograd, l'algorithme de Waksman égale les 7 multiplications du schéma de Strassen [1].

La procédure `waksman_product` du module Maple décrit à l'annexe A génère du code pour l'algorithme de Waksman en taille fixée. Un exemple de résultat est donné figure 3. Les figures 4a et 4b comparent la performance de ce code avec celui du produit matriciel « efficace » fourni par le module d'algèbre linéaire de bas niveau de Maple, et avec une fonction non documentée de Maple qui s'avère plus efficace que ce dernier (lui-même nettement plus rapide que la fonction de haut niveau `LinearAlgebra:-MatrixMatrixMultiply`). On constate que l'algorithme de Waksman est sensiblement plus efficace, à partir de la taille 4 environ, pour des matrices denses dont les entrées font quelques milliers de chiffres. (Cet avantage devient réellement intéressant, dès la taille 2, en poussant l'expérience jusqu'à des coefficients d'un million de chiffres ou plus.) En revanche, contrairement au produit naïf, il se comporte très mal vis-à-vis des matrices creuses. Pour l'application aux suites récurrentes, cet effet est très important, puisque les matrices à multiplier sont des matrices compagnon, dont les produits ne se « densifient » qu'assez lentement.

1. Il effectue cependant 23 additions, contre 18 pour l'algorithme de Strassen et 15 pour sa version améliorée par Winograd [**vzGG03**, p. 328]. Mais sur les entiers, le coût de l'addition est du même ordre que celui de la lecture de l'entrée, et je n'ai pu observer quasiment aucune différence en pratique.



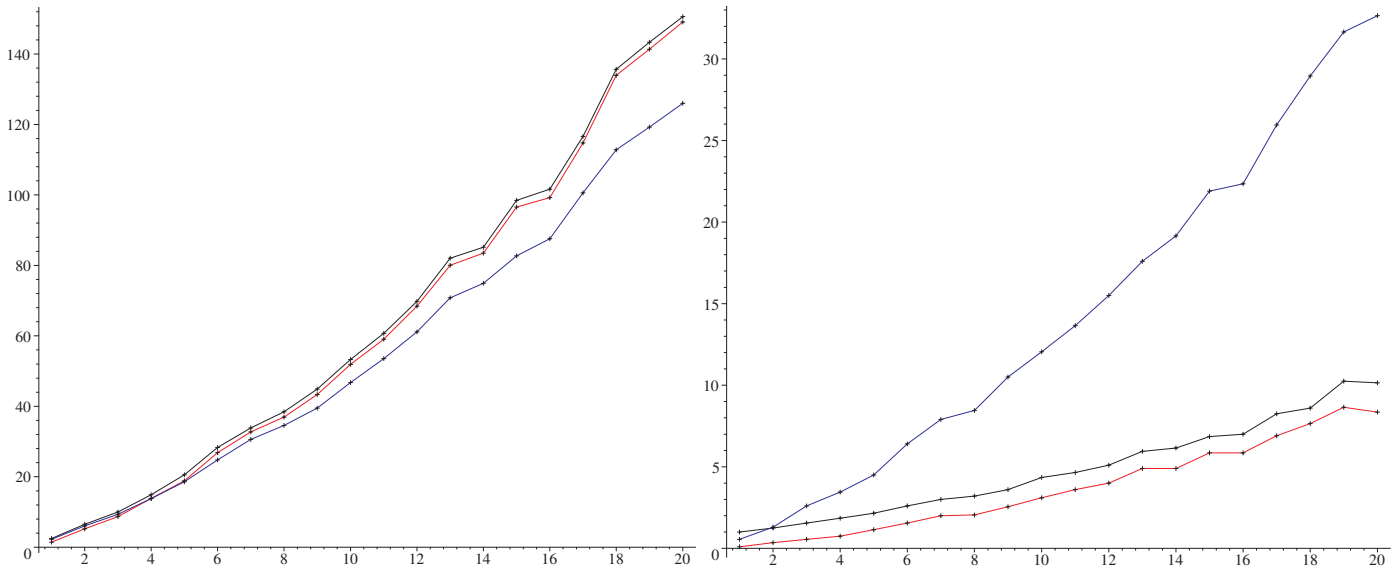
Matrices 2×2



Matrices 3×3

Noir : `LinearAlgebra:-LA_Main:-MatrixMatrixMultiply` (fonction documentée « rapide »)
Rouge : `mvMultiply` (fonction non documentée du noyau Maple)
Bleu : procédure renvoyée par `waksman_product` (cf. annexe A)

FIG. 4A. Coûts comparés de divers produits matriciels. *Temps d'exécution en millisecondes (moyenne sur 20 expériences) en fonction de la taille approximative des entrées de la matrice, en milliers de chiffres décimaux. Colonne de gauche, produit de matrices denses d'entiers ; colonne de droite, produit de matrices compagnon.*



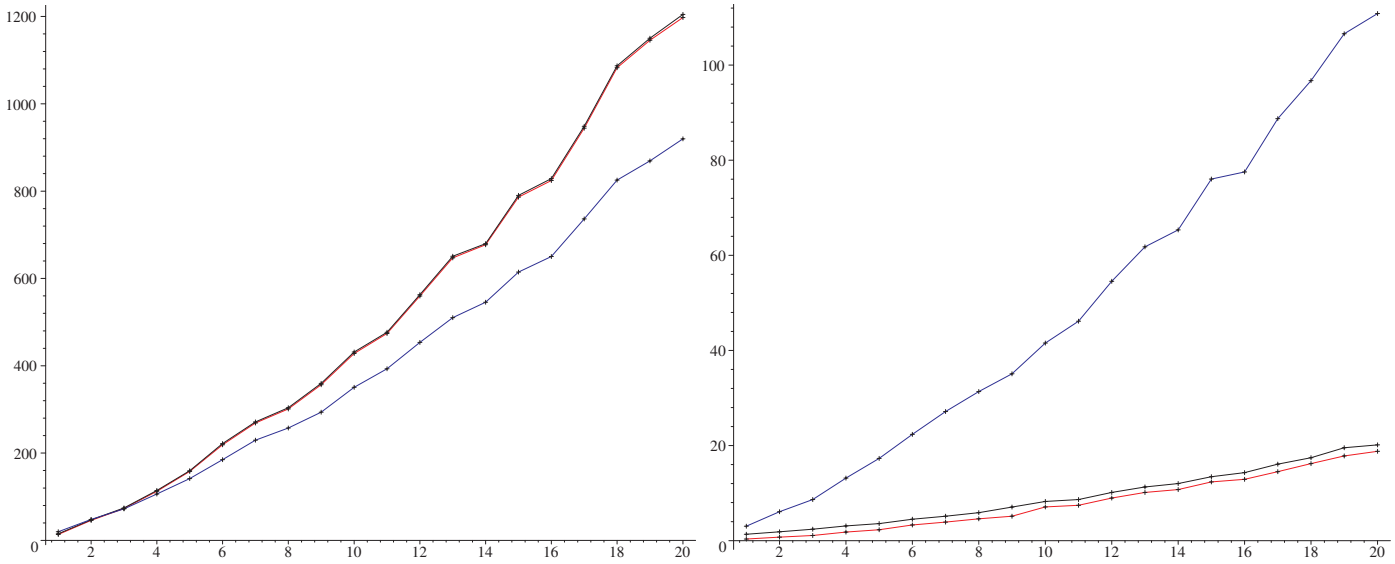


FIG. 4B. Coûts comparés de divers produits matriciels (suite)

Taille	2	3	4	5	6	7	8	9	10
Naïf	8	27	64	125	216	343	512	729	1000
Non-com.	**7**	**23**	49	100	161	273	343	529	700
Waksman	**7**	**23**	**46**	**93**	**141**	**235**	**316**	**473**	**595**
Mixte	**7**	**23**	**46**	**93**	**141**	**235**	**316**	**473**	**595**

Taille	11	12	13	14	15	16	17	18	19
Naïf	1331	1728	2197	2744	3375	4096	4913	5832	6859
Non-com.	992	1125	1580	1778	2300	2401	3218	3342	4369
Waksman	**831**	1002	**1333**	**1561**	**2003**	2296	**2865**	**3231**	**3943**
Mixte	**831**	**987**	**1333**	**1561**	**2003**	**2212**	**2865**	**3231**	**3943**

Taille	20	21	22	23	24	25	26	27	28
Naïf	8000	9261	10648	12167	13824	15625	17576	19683	21952
Non-com.	4380	5610	**5610**	7048	7048	**8710**	**8710**	**10612**	**10612**
Waksman	4390	**5261**	5797	**6843**	7476	8713	9451	10895	11746
Mixte	**4165**	**5261**	**5610**	**6843**	**6909**	**8710**	**8710**	**10612**	**10612**

TAB. 3. Coût du produit de petites matrices

2.2.3. Généralisations. — À ce stade, il est naturel de se demander si l'on pourrait obtenir des algorithmes « à la Winograd » plus efficaces en regroupant les termes de (15) par paquets plus gros — disons trois à trois ou quatre à quatre plutôt que deux à deux. La réponse est non [**Har72**]. Précisément, il n'existe pas de généralisation de (16) de la forme

$$a_1b_1 + a_2b_2 + \cdots = \lambda(a_1, b_1, a_2, b_2, \ldots) \cdot \mu(a_1, b_1, a_2, b_2, \ldots) + f(a) + g(b),$$

où λ, μ sont des formes linéaires et f, g des fonctions quelconques.

2.2.4. Schémas mixtes. — La première taille pour laquelle l'algorithme de Strassen pur (1) — en agrandissant la matrice à la puissance de deux supérieure si nécessaire — effectue moins de multiplications que celui de Waksman est 32. Cependant, dès la taille 12, un schéma mixte qui découpe la matrice en quatre blocs 6×6 auxquels on applique les formules de Strassen, les produits de blocs étant effectués par l'algorithme de Waksman, est moins coûteux que l'algorithme de Waksman pur, avec 987 multiplications contre 1002 (tandis que descendre par l'algorithme de Strassen jusqu'à des blocs de taille 3 que l'on multiplie par l'algorithme de Waksman prend 1029 multiplications).

La dernière ligne du tableau 3 donne les complexités que l'on peut atteindre pour le produit de petites matrices carrées en combinant de cette manière les meilleurs algorithmes non-commutatifs connus (d'après le recensement de W.D. Smith [**Smi02**]) avec l'algorithme de Waksman. En comparaison, les trois premières lignes donnent respectivement la complexité de l'algorithme naïf, la meilleure complexité connue sur un anneau non commutatif d'après [**Smi02**], et la complexité de l'algorithme de Waksman.

On observe que les schémas de Waksman et de Strassen suffisent jusqu'en taille 22. Plus précisément, jusqu'en taille 11, l'algorithme de Waksman pur donne les meilleurs résultats, et pour une matrice de taille comprise entre 12 et 21, la meilleure complexité du tableau est atteinte soit comme précédemment, soit en multipliant par l'algorithme de Waksman des blocs obtenus par une décomposition de Strassen à un seul niveau. La conclusion pragmatique est que l'algorithme de Waksman, utilisé soit directement, soit aux feuille d'un schémas de Strassen à un ou deux niveaux, est un bon choix pour le scindage binaire sur des récurrences d'ordre modéré, *à partir du moment où les matrices sont denses*.

Bien évidemment, cela n'épuise pas la question de minimiser le nombre de multiplications pour le produit de matrices sur un anneau commutatif, même en se limitant à la combinaison d'algorithmes connus. Par exemple, comme le remarque Landsberg [**Lan07**, exemple 14.1.1],

le schéma de Waksman en taille 3×3 permet de faire le produit 2×3 par 3×2 en 10 multiplications, simplement en complétant les entrées par des zéros et en faisant les simplifications évidentes. Il n'est pas exclu qu'un produit par blocs utilisant une décomposition en blocs non carrés (possibilité que nous n'avons considérée jusqu'ici que lorsqu'elle était valable en non commutatif) puisse exploiter ce fait.

2.3. Sommes partielles de séries et factorisation d'opérateurs

Terminons par quelques remarques qui permettent de diminuer légèrement, mais de façon assez sensible en pratique, la constante du scindage binaire pour le calcul de sommes partielles de séries. Soit (u_n) une suite récurrente linéaire, solution de

$$L \cdot u_n = u_{n+r} + a_{r-1}u_{n+r-1} + \cdots + a_0 u_n = 0.$$

Les sommes partielles de la série de terme général u_n,

$$v_n = \sum_{k=0}^{n-1} u_k$$

vérifient $v_{n+1} - v_n = u_n$, et donc

$$\hat{L} \cdot v_n = v_{n+r+1} + (a_{r-1} - 1)v_{n+r} + \cdots + (a_0 - a_1)v_{n+1} - a_0 v_n = 0. \tag{17}$$

On peut calculer v_n par scindage binaire *via* la traduction classique de (17) en une récurrence du premier ordre déjà exploitée §1.3 :

$$\begin{bmatrix} v_{n+1} \\ \vdots \\ v_{n+r} \\ v_{n+r+1} \end{bmatrix} = \begin{bmatrix} & 1 & & \\ & & \ddots & \\ & & & 1 \\ a_0 & a_1 - a_0 & \dots & 1 - a_{r-1} \end{bmatrix} \begin{bmatrix} v_n \\ \vdots \\ v_{n+r-1} \\ v_{n+r} \end{bmatrix}. \tag{18}$$

Cependant, une alternative au calcul de la récurrence (17) est de dérouler en parallèle les deux récurrences $L \cdot u_n = 0$ et $v_{n+1} = v_n + u_n$. Cette variante admet aussi une écriture matricielle :

$$\begin{bmatrix} u_{n+1} \\ \vdots \\ u_{n+r-1} \\ u_{n+r} \\ v_{n+1} \end{bmatrix} = \begin{bmatrix} & 1 & & & 0 \\ & & \ddots & & \vdots \\ & & & 1 & \vdots \\ -a_0 & -a_1 & \dots & -a_{r-1} & 0 \\ 1 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} u_n \\ \vdots \\ u_{n+r-2} \\ u_{n+r-1} \\ v_n \end{bmatrix}. \tag{19}$$

Comparons les coûts de ces deux formules. Soit $\mathrm{MM}(r)$ le nombre de multiplications scalaires effectuées pour multiplier deux matrices carrées d'ordre r. Un produit de matrices compagnons se densifie rapidement, aussi la multiplication de matrices utilisée pour dérouler par scindage binaire la récurrence (18) prend $\mathrm{MM}(r+1)$ multiplications d'entiers. À l'inverse, quand on multiplie les matrices données par (19), la dernière colonne reste constante. Le produit de matrices obtenues en itérant (19) peut se faire en une multiplication de matrices en taille r et une multiplication vecteur-matrice, soit $\mathrm{MM}(r) + r^2$ multiplications.

Bien souvent cependant, l'expression de u_r en fonction de $u_0, \dots, u_{r-1}$ est à coefficients rationnels, et l'on souhaite séparer numérateur et dénominateur dans le scindage binaire. Les formules (18) et (19) restent valables pour le numérateur en remplaçant les 1 au-dessus de la diagonale et celui en bas à droite de la matrice de (18) par le dénominateur. En comptant le calcul du dénominateur, les complexités montent à $\mathrm{MM}(r+1) + 1$ multiplications d'entiers pour (18) ; et à un produit matrice-matrice, un produit vecteur-matrice, un produit scalaire-vecteur et un produit de scalaires, soit $\mathrm{MM}(r) + r^2 + r + 1$ opérations, pour (19). (Le calcul

du dénominateur, qui apparaît dans le bloc réduit à une case en bas à droite, est incorporé dans le produit matriciel.)

Avec la multiplication naïve,

$$\mathrm{MM}(r+1)+1 = r^3+3r^2+3r+2 > r^3+r^2+r+1 = \mathrm{MM}(r)+r^2+r+1,$$

et avec celle de Waksman

$$\begin{aligned}\mathrm{MM}(r+1) &= \frac{r^3+5r^2+6r+2+(r+1 \bmod 2)r^2}{2} \\ &> \frac{r^3+4r^2+r+1+(r \bmod 2)(r-1)^2}{2} = \mathrm{MM}(r)+r^2+r+1\end{aligned}$$

donc la variante (19) semble préférable dans ces deux cas. On peut vérifier qu'il en va de même pour les complexités données à la dernière ligne du tableau 3 jusqu'à la taille 15 (au-delà, la meilleure formule est souvent celle qui fait les produits matriciels en taille paire).

La possibilité de dérouler la récurrence suivant (19) est en fait une instance d'un phénomène plus général. Considérons un opérateur aux différences L, d'ordre r, qui admet une factorisation $L = L_k \cdots L_1$, où l'on suppose tous les L_j unitaires en S (on avait ci-dessus $\hat{L} = L\Delta$). Soit $u^{[0]}$ satisfaisant $L \cdot u^{[0]} = 0$, et introduisons les suites intermédiaires

$$u^{[1]} = L_1 \cdot u, \quad u^{[2]} = L_2 \cdot u^{[1]}, \quad \ldots, \quad u^{[k-1]} = L_{k-1} \cdot u^{[k-2]}.$$

Alors $u^{[0]}$ peut se calculer par la formule

(20)
$$\begin{bmatrix} u^{[k-1]}_{n+1} \\ \vdots \\ u^{[k-1]}_{n+r_{k-1}} \\ u^{[k-2]}_{n+1} \\ \vdots \\ u^{[k-2]}_{n+r_{k-2}} \\ \vdots \\ u^{[1]}_{n+1} \\ \vdots \\ u^{[1]}_{n+r_1} \\ u^{[0]}_{n+1} \\ \vdots \\ u^{[0]}_{n+r_0} \end{bmatrix} = \begin{bmatrix} C_{k-1} & & & & \\ & C_{k-2} & & & \\ 1 & & \ddots & \ddots & \\ & & & C_1 & \\ & & 1 & & C_0 \\ & & & 1 & \end{bmatrix} \begin{bmatrix} u^{[k-1]}_{n} \\ \vdots \\ u^{[k-1]}_{n+r_{k-1}-1} \\ u^{[k-2]}_{n} \\ \vdots \\ u^{[k-2]}_{n+r_{k-2}-1} \\ \vdots \\ u^{[1]}_{n} \\ \vdots \\ u^{[1]}_{n+r_1-1} \\ u^{[0]}_{n} \\ \vdots \\ u^{[0]}_{n+r_0-1} \end{bmatrix}$$

où r_j désigne l'ordre de L_j et C_j sa matrice compagnon. Les produits de matrices donnés par (20) restent dans une sous-algèbre de matrices triangulaires par blocs avec $\sum_{i>j} r_i r_j$ composantes nulles. (Remarquons que ce nombre dépend seulement des degrés des facteurs, et pas de leur ordre.)

Concernant l'implémentation, on peut noter que la multiplication naïve de matrices tire naturellement parti [(2)] de la structure triangulaire par blocs. Il n'est donc pas forcément nécessaire d'écrire du code spécifique pour multiplier les matrices données par (20) plus rapidement que des matrices denses.

2. Tant que le coût du contrôle est négligeable devant celui des produits de scalaires — mais c'est l'hypothèse à la base de toute cette discussion.

II

ÉVALUATION NUMÉRIQUE DES FONCTIONS HOLONOMES

CHAPITRE 3

ÉQUATIONS DIFFÉRENTIELLES, FONCTIONS HOLONOMES

Ce chapitre est consacré aux prérequis mathématiques de la deuxième partie. Les objets qui nous intéressent en premier lieu sont les fonctions holonomes, c'est-à-dire solutions d'équations différentielles linéaires à coefficients polynomiaux. Mais de nombreux résultats qui sont en fait communs à toutes les solutions analytiques d'équations différentielles linéaires sont énoncés dans ce cadre plus général.

Les principales propriétés des fonctions holonomes sont connues au moins depuis le XIXe siècle : par exemple, le caractère holonome des fonctions algébriques (proposition 3.16 ci-dessous) est mentionné dans les notes d'Abel. Leur usage moderne en combinatoire et en calcul formel a été popularisé par Stanley [**Sta80**] puis Zeilberger [**Zei90**], qui ont souligné leur intérêt comme classe de séries close par plusieurs opérations importantes et bénéficiant d'une algorithmique riche et (déjà relativement) efficace. L'étude systématique, du point de vue algorithmique, des fonctions holonomes — et de leurs cousines les suites holonomes, solutions de récurrences linéaires à coefficients polynomiaux, que nous avons déjà abondamment manipulées — s'est poursuivie depuis. Sur le plan pratique, le module Maple `gfun` [**SZ94**] fournit ainsi un grand nombre d'opérations sur les suites et fonctions holonomes, représentées implicitement par les équations qu'elles vérifient. Il est abondamment mis à contribution par l'implémentation des algorithmes de cette partie.

Notons aussi, même si ceci ne nous concerne guère ici, que la notion d'holonomie admet des généralisations fructueuses à plusieurs variables [**Lip89**], et à des systèmes mixtes mélangeant dérivées partielles, récurrences multiples, q-analogues... Les algorithmes associés permettent de manipuler des expressions complexes dans les systèmes de calcul formel, et notamment de prouver, voire de calculer, des identités comme

$$\frac{1}{2}J_0(z)^2 + \sum_{\nu=1}^{\infty} J_\nu(z)^2 = \frac{1}{2}$$

(où les J_ν sont les fonctions de Bessel) entre sommes et intégrales de solutions de systèmes holonomes [**CS98, Chy98**].

3.1. Théorème de Cauchy

Cette section rappelle quelques résultats de base sur les équations différentielles linéaires à coefficients holomorphes. Pour une présentation plus complète et plus rigoureuse, on pourra consulter le tome 2 du traité d'analyse complexe d'Henrici [**Hen77**] ou le livre classique de Ince sur les équations différentielles [**Inc56**]. Les résultats sont énoncés en 0, ils s'étendent bien entendu à n'importe quel $z_0 \in \mathbb{C}$ par le changement de variable $z \leftarrow z - z_0$, et au point à l'infini de la sphère de Riemann *via* $z \leftarrow 1/z$.

3.1.1. Systèmes différentiels linéaires. — Comme dans le cas réel, voire plus simplement, on dispose d'un théorème d'existence de solutions pour les systèmes linéaires.

THÉORÈME 3.1. — Soit $D \subset \mathbb{C}$ un disque centré en 0 de rayon $\gamma > 0$. Soit $A : D \to M_r(\mathbb{C})$ une fonction analytique à valeurs matricielles, et soit $w \in \mathbb{C}^r$. Le système d'inconnue $y(z)$

$$y'(z) = A(z)y(z) \tag{21}$$

admet une unique solution $y(z)$ analytique au voisinage de 0 vérifiant la condition initiale $y(0) = w$. En outre, la série de Taylor de y en 0 a un rayon de convergence $\geqslant \gamma$.

Une particularité importante des équations différentielles linéaires est qu'une solution de (21) ne peut cesser d'être analytique qu'en une singularité de A. La preuve simple suivante est classique, mais elle contient une première apparition de plusieurs idées importantes dans la suite, en particulier la méthode des séries majorantes (chapitre 5).

Démonstration. — Tentons de résoudre l'équation (21) par la méthode des coefficients indéterminés. En développant en série les deux membres, celle-ci se réécrit (pour $|z| < \gamma$)

$$\sum_{n=0}^{\infty} (n+1)y_{n+1} = \sum_{n=0}^{\infty} \Big(\sum_{k=0}^{n} A_k y_{n-k} \Big) z^n.$$

La série $y(z)$ définie par $y_0 = w$ et la relation de récurrence

$$\forall n \geqslant 0, \quad (n+1)y_{n+1} = \sum_{k=0}^{n} A_k y_{n-k}$$

est donc l'unique solution formelle. Reste à montrer qu'elle converge.

Soit $r < \gamma$. Soit $\|\cdot\|$ une norme sur $\mathbb{C}^r$, on note $|||\cdot|||$ sa norme subordonnée. Puisque le rayon de convergence de A est au moins γ, pour un certain M, on a $\forall k, |||A_k||| \leqslant M/r^k$, d'où

$$(n+1)\,\|y_{n+1}\| \leqslant \sum_{k=0}^{n} \frac{M}{r^k}\,\|y_{n-k}\|\,.$$

Introduisons la suite réelle (ω_n) définie par $\omega_0 = \|w\|$ et

$$\forall n \geqslant 0, \quad (n+1)\omega_{n+1} = \sum_{k=0}^{n} \frac{M}{r^k}\omega_{n-k},$$

et posons

$$\omega(z) = \sum_{n=0}^{\infty} \omega_n z^n.$$

Par récurrence immédiate, $\|y_n\| \leqslant \omega_n$ pour tout n. Or par définition de ω,

$$\omega'(z) = \sum_{n=0}^{\infty} (n+1)\omega_{n+1} z^n = \sum_{n=0}^{\infty} \sum_{k=0}^{n} \frac{M}{r^k}\omega_{n-k} z^n = \frac{M}{r}\frac{1}{1-\frac{z}{r}}\omega(z).$$

C'est là une équation différentielle réelle du premier ordre, facile à résoudre explicitement : son espace de solutions est $\mathbb{R} \cdot (r-z)^{-M}$, et

$$\omega(z) = r^M \|w\| \frac{1}{(r-z)^M}.$$

La série $\omega(z)$ converge donc pour $|z| < r$, donc $y(z)$ aussi. Ceci étant vrai pour tout $r < \gamma$, on a le résultat. □

COROLLAIRE 3.2. — Les fonctions analytiques au voisinage de 0 solutions de (21) forment un $\mathbb{C}$-espace vectoriel de dimension r.

DÉFINITION 3.3. — Une matrice dont les colonnes forment une base de solutions est appelée matrice fondamentale de (21).

Une matrice fondamentale $Y(z)$ est donc elle-même solution de $Y' = AY$. Toute solution s'écrit comme la produit d'une matrice fondamentale (quelconque) par un vecteur de constantes. Les matrices fondamentales s'obtiennent les unes à partir des autres par multiplication par des matrices constantes inversibles.

3.1.2. Équations scalaires. — Soient $a_0, \dots, a_r : \mathbb{C} \to \mathbb{C}$ des fonctions analytiques au voisinage de 0. Considérons l'équation différentielle scalaire homogène d'ordre r, de fonction inconnue $y(z)$

$$a_r(z)\, y^{(r)}(z) + \cdots + a_1(z)\, y'(z) + a_0(z)\, y(z) = 0. \tag{22}$$

La réduction usuelle des équations scalaires d'ordre quelconque aux systèmes du premier ordre et le théorème de Cauchy motivent la définition suivante.

DÉFINITION 3.4. — On dit que z_0 est une singularité, ou un point singulier, de l'équation (22) si z_0 est une singularité d'un des coefficients $a_0, \dots, a_r$ ou un zéro de a_r. Un point qui n'est pas singulier est appelé un point ordinaire.

On obtient alors la version scalaire du théorème de Cauchy.

COROLLAIRE 3.5. — Si $a_r(0) \neq 0$ (*i.e.* si 0 n'est pas une singularité), l'équation (22) admet un espace vectoriel de dimension r de solutions analytiques au voisinage de 0, et celles-ci sont en fait analytiques sur le disque centré en 0 s'étendant jusqu'à la plus proche singularité de l'équation.

À nouveau, les singularités des solutions sont parmi celles de l'équation. En particulier, les solutions d'une équation différentielle linéaire à coefficients polynomiaux ont un nombre fini de singularités. La position des singularités se lit sur l'équation, ainsi qu'un certain nombre de propriétés asymptotiques importantes des solutions en leur voisinage (voir section 3.2).

3.1.3. Prolongement analytique. — Soit U un ouvert de $\mathbb{C}$ ne contenant pas de singularité de l'équation (22), et soit $\gamma : [0;1] \to U$ un chemin tracé dans U. Au voisinage de tout point de γ, l'équation admet une base de solutions analytiques. Deux telles solutions dont les dérivées d'ordre $0, \dots, r-1$ coïncident en un point sont égales localement, et donc sur l'intersection de leurs domaines de définition. Étant donnée une solution f de (22) au voisinage de $\gamma(0)$, il existe donc une famille de solutions holomorphes de (22) prolongeant f dont la réunion des domaines de définition recouvre γ. Par compacité de γ, un nombre fini d'étapes suffit à atteindre $\gamma(1)$.

L'application « prolongement analytique le long de γ » ainsi définie est linéaire, c'est un isomorphisme de l'espace des solutions locales en $\gamma(0)$ dans l'espace des solutions locales en $\gamma(1)$. Elle ne dépend que de la classe d'homotopie du chemin γ. Sur un domaine simplement connexe, on peut identifier les solutions prolongement analytique les unes des autres (c'est-à-dire aussi restrictions d'une même solution maximale). On aboutit au résultat classique suivant.

THÉORÈME 3.6. — Soit V un ouvert simplement connexe de $\mathbb{C}$ (ou plus généralement d'un revêtement d'un ouvert de $\mathbb{C}$) sur lequel (resp. la base duquel) $a_0, \dots, a_r$ sont analytiques et a_r ne s'annule pas. Alors l'équation (22) admet un espace vectoriel de dimension r de solutions analytiques sur V.

Le procédé de prolongement esquissé ci-dessus est constructif : concrètement, à partir de l'équation (22) et de conditions initiales en $z_0 = 0$, on évalue la solution f correspondante et ses dérivées en nombre suffisant en un point z_1 « près du bord » du domaine de convergence. On obtient des conditions initiales en z_1 spécifiant une solution qui prolonge f, avec un développement en série (au voisinage de z_1) valable au-delà du disque de convergence de f en z_0. Et ainsi de suite (voir figure 5).

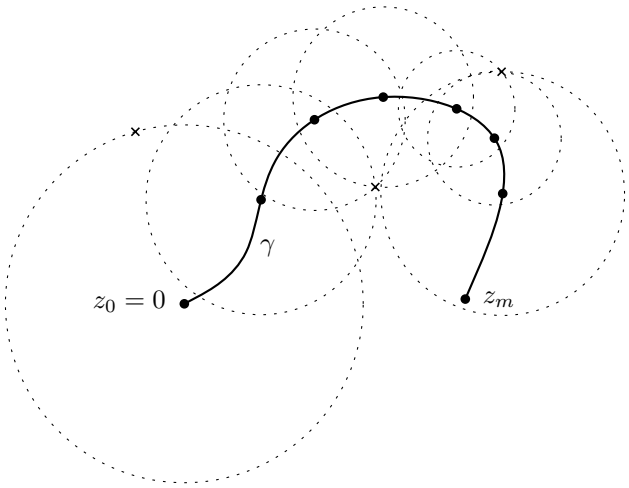


FIG. 5. Principe du prolongement analytique. *Les points représentent les étapes de prolongement, les croix, les singularités.*

3.2. Comportement au voisinage des singularités isolées

Soit D un disque centré en 0, et soient $a_0, \dots, a_{r-1}$ des fonctions analytiques sur le disque épointé $\dot{D} = D \setminus \{0\}$. Considérons l'équation différentielle

$$y^{(r)} + a_{r-1} y^{(r-1)} + \cdots + a_0 y = 0. \tag{23}$$

Si les a_j admettent un prolongement analytique à l'origine, celle-ci est un point ordinaire. On a vu section 3.1 que dans ce cas, (23) possède une base de solutions holomorphes au voisinage de 0, c'est-à-dire une base de solutions séries formelles de rayon de convergence non nul. Supposons au contraire que 0 soit un point singulier.

3.2.1. Singularités apparentes. — Il peut arriver que 0 soit une singularité de (23), mais que celle-ci admette néanmoins une base de solutions analytiques au voisinage de 0. On dit alors que 0 est une singularité apparente.

EXEMPLE. — La singularité en 0 de l'équation

$$-z(z-1)^2 y'' + (z-1)(z^2-2z-1)y' + (z^2-1)y = 0,$$

renvoyée par `gfun` comme annulant la somme $\frac{1}{1-z} + e^z$, est une singularité apparente. Intuitivement, cela peut s'interpréter ainsi : l'équation annule en fait chacun des deux termes

$$e^z = 1 + z + \frac{z^2}{2} + O(z^3)$$

$$\text{et} \qquad \frac{1}{1-z} = 1 + z + z^2 + O(z^3).$$

Par combinaison linéaire, elle a donc une solution de valuation 2. Or en un point régulier, une solution d'une équation d'ordre 2 est déterminée par les deux premiers termes de son développement de Taylor, et en particulier une solution de la forme $O(z^2)$ est nulle. Il faut donc, bien que toutes les solutions y soient analytiques, que 0 soit une singularité.

Le procédé de clôture de Weyl [**Tsa00**] permet, étant donné un opérateur $L \in \mathbb{C}[z, \partial]$, d'en trouver un multiple à gauche $\tilde{L} = KL$ (d'espace de solutions contenant donc celui de L) sans singularités apparentes [1]. Il en existe des implémentations. Quitte à les modifier ainsi, nous

1. Il y a un analogue pour les récurrences [**ABvH06**].

supposerons dans la suite, et en particulier dans les algorithmes, les équations différentielles sans singularités apparentes.

3.2.2. Points singuliers réguliers, points singuliers irréguliers. — Lorsque 0 est une singularité de (23), les solutions au voisinage de 0 ne sont pas déterminées de façon unique, au sens où le prolongement analytique le long d'un lacet contournant 0 ne redonne en général pas la même solution après un tour. Elles possèdent cependant une structure relativement simple.

THÉORÈME 3.7 (voir [**Hen77**, §9.4 et p. 124–125]). — Sur le disque épointé $\dot{D}$ (ou son revêtement universel), toute solution f de (23) s'écrit comme une somme finie

$$f(z) = \sum_j z^{\alpha_j} (\log z)^{k_j} s_j(z) \tag{24}$$

où les $\alpha_j \in \mathbb{C}$, les $k_j \in \mathbb{N}$ et les s_j sont des fonctions analytiques sur $\dot{D}$.

On peut penser à (24) comme une fonction analytique définie *autour* de la singularité, mais aussi comme une solution formelle *en* la singularité, ou un développement asymptotique. Ce théorème conduit à une classification des singularités des équations différentielles linéaires à coefficients analytiques suivant le comportement des s_j quand $z \to 0$.

DÉFINITION 3.8. — On dit que 0 est un point singulier régulier de (23) si pour toute solution f de (23), les assertions équivalentes suivantes sont vérifiées :

(i) f admet une écriture de la forme (24) pour laquelle chaque $s_j(z)$ est analytique au voisinage de zéro ;

(ii) il existe N tel que $f(z) = O(|z|^{-N})$ quand z tend vers 0 en restant confiné à un secteur angulaire de sommet 0.

Une singularité apparente est un cas particulier de point singulier régulier. Un point singulier qui n'est pas régulier est dit irrégulier.

Au voisinage d'un point singulier régulier, l'équation (23) admet une base de solutions formelles de la forme (24), où les s_j sont cette fois analytiques sur le disque D. Ces solutions fournissent une base de solutions analytiques sur tout secteur angulaire de sommet 0 contenu dans D. Les comportements asymptotiques possibles d'une fonction holonome au voisinage d'un point singulier régulier sont donc restrictifs. Par analyse de singularité, on en déduit des restrictions du même type sur les coefficients des développements [**FS07**, § VII.9.1, p. 494–496].

PROPOSITION 3.9. — Soit $f = \sum_n f_n z^n$ une fonction analytique au voisinage de 0, solution d'une équation différentielle linéaire à coefficients analytiques. Supposons que les singularités de f les plus proches de l'origine sont de module ρ et sont toutes des points singuliers réguliers de l'équation. Alors la suite f_n admet un développement asymptotique de la forme

$$f_n = \frac{1}{\rho^n} \sum_j^{\text{finie}} \lambda_j n^{\beta_j} (\log n)^{k_j} + o\Big(\frac{1}{\rho^n}\Big)$$

(où $\lambda_j, \beta_j \in \mathbb{C}$ avec les β_j algébriques sur le corps des coefficients de l'équation, et $k_j \in \mathbb{N}$) quand $n \to \infty$.

La situation est plus compliquée si 0 est un point singulier irrégulier. Il est encore possible de donner une base de solutions formelles, cette fois de la forme

$$e^{P(1/\sqrt[p]{z})} z^{\alpha} \sum_{n=0}^{\infty} Q_n(\log z) z^{n/p}$$

où P et les Q_n sont des polynômes (à coefficients complexes) de degré borné, $p \in \mathbb{N}$ et $\alpha \in \mathbb{C}$ (voir [**vdH01**, §3] et les références citées). Ces développements sont en général divergents, mais asymptotiques à des solutions sur de petits secteurs de sommet 0. Le comportement asymptotique des solutions demeure fortement contraint.

L'analyse de singularité ne s'applique pas dans ce cas. Pour déterminer néanmoins les comportement asymptotiques autorisés pour les coefficients d'une fonction holonome au voisinage d'un point singulier régulier, on peut repasser dans le monde des récurrences et faire appel au résultat suivant [**Odl95**, §9], conséquence du théorème de Birkhoff-Trjitzinsky sur les solutions formelles et asymptotiques de récurrences linéaires [**BT32**] [2].

THÉORÈME 3.10. — Soit $(u_n)_{n\in\mathbb{N}}$ une suite solution de la récurrence

$$b_r(n)\,u_{n+r} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0 \qquad (b_r(n) \neq 0 \text{ pour } n \in \mathbb{N}). \tag{25}$$

Supposons que tous les coefficients b_j admettent des développements asymptotiques

$$b_j(n) \sim_{n\to\infty} n^{K/t} \sum_{k=0}^{\infty} \frac{c_k}{n^{k/t}}$$

où $p \in \mathbb{N}^*$, $K \in \mathbb{Z}$, et $c_k \in \mathbb{C}$. Alors il existe $p_j, q, k_j, m \in \mathbb{Z}$, $\alpha_j, \lambda_j \in \mathbb{C}$, $P_j \in \mathbb{C}[z]$ tels que

$$u_n \sim \sum_j \lambda_j\, n!^{p_j/q}\, e^{P_j(\sqrt[m]{n})}\, n^{\alpha_j}\, (\log n)^{k_j}$$

(la somme est finie) quand $n \to \infty$.

Un dernier point important est que dans le cas d'une équation scalaire, la nature d'une singularité peut se lire sur les coefficients. Le théorème suivant est démontré dans [**Inc56**, §15.3, p. 365–370].

THÉORÈME 3.11 (Critère de Fuchs). — L'origine est un point singulier régulier de l'équation (23) si et seulement si pour tout $j \in [\![0, r-1]\!]$, la fonction a_j est analytique en 0 ou y admet un pôle d'ordre [3] $\leqslant n - j$.

Si les coefficients de l'équation sont des fractions rationnelles, le critère de Fuchs fournit un algorithme pour reconnaître les points singuliers réguliers.

3.3. Fonctions holonomes

Soit $\mathbb{K}$ un corps de caractéristique nulle, pour nous généralement un sous-corps de $\mathbb{C}$. Cette section définit les suites et fonctions holonomes d'une variable sur $\mathbb{K}$, et passe en revue, sans démonstration, leurs principales propriétés. On trouvera les preuves des théorèmes mentionnés, ainsi que de nombreux exemples d'applications algorithmiques de l'holonomie qui sortent du cadre de ce rapport, dans les notes de cours [**BCS07**].

DÉFINITION 3.12. — Une série $f \in \mathbb{K}[[x]]$ est dite *holonome*, ou *différentiellement finie* (abrégé en D-finie ou ∂-finie), si elle vérifie les conditions équivalentes suivantes :

(i) f est solution (formelle) d'une équation différentielle linéaire à coefficients et second membre fractions rationnelles : il existe $a_0, \ldots, a_r, b \in \mathbb{K}(z)$ tels que

$$a_r(z)\, f^{(r)}(z) + \cdots + a_1(z)\, f'(z) + a_0(z)\, f(z) = b(z);$$

(ii) f est solution (formelle) d'une équation différentielle linéaire homogène à coefficients polynomiaux ;

2. Voir [**WZ85**] pour une présentation de ce théorème plus accessible que l'article original.
3. « Pour chaque terme, ordre de dérivation + ordre du pôle ⩽ ordre de l'équation. »

(iii) le sous-espace vectoriel sur $\mathbb{K}(z)$ de $\mathbb{K}((z))$ engendré par les dérivées successives de f est de dimension finie.

De même, une fonction analytique f est dite holonome si elle est solution d'une équation différentielle linéaire homogène à coefficients polynomiaux.

Ainsi, une fonction analytique est holonome si et seulement si ses développements de Taylor en tout point le sont. Les fonctions $\exp z$, $\log(1+z)$, $\arctan z$, de nombreuses fonctions spéciales comme les fonctions hypergéométriques, la fonction d'erreur

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

ou les fonctions $I_\nu, J_\nu, K_\nu, Y_\nu$ de Bessel, des séries divergentes comme

$$\sum_{n=0}^{\infty} n! z^n$$

sont holonomes. À l'inverse, la fonction $\tan z$ n'est pas holonome (une fonction holonome n'a qu'un nombre fini de singularités).

DÉFINITION 3.13. — Une suite $u = (u_n)_n \in \mathbb{K}^{\mathbb{N}}$ est dite *holonome*, ou *polynomialement récursive* (P-récursive), si elle satisfait les conditions équivalentes suivantes [4] :

(i) u est solution, à partir d'un certain rang, d'une récurrence linéaire à coefficients et second membre polynomiaux : il existe $a_0, \dots, a_r, b \in \mathbb{K}[z]$ et $n_0 \in \mathbb{N}$ tels que

$$\forall n \geqslant n_0, \quad a_r(n)\,u(n+r) + \cdots + a_1(n)\,u(n+1) + a_0(n)\,u(n) = b(n);$$

(ii) il existe $a_0, \dots, a_r \in \mathbb{K}[z]$ tels que

$$\forall n \in \mathbb{N}, \quad a_r(n)\,u(n+r) + \cdots + a_1(n)\,u(n+1) + a_0(n)\,u(n) = 0.$$

La suite de Fibonacci (F_n) solution de $F_{n+2} = F_{n+1} + F_n$, la factorielle $n!$, les nombres harmoniques $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$, les nombres de Catalan $\frac{1}{n+1}\binom{2n}{n}$, ou encore d'innombrables suites issues de la combinatoire, fournissent des exemples de suites holonomes.

PROPOSITION 3.14. — Une série $f \in \mathbb{K}[[z]]$ est D-finie si et seulement si la suite $(f_n)_{n\in\mathbb{N}}$ de ses coefficients est P-récursive.

La démonstration est constructive, et sera le point de départ du procédé d'évaluation des fonctions holonomes par scindage binaire. (Voir aussi la section suivante.)

Sur un corps comme $\mathbb{Q}[i]$, toute fonction holonome peut être spécifiée par une description de longueur finie, à savoir une équation qui l'annule et un nombre suffisant de conditions initiales. Fait marquant, l'égalité à zéro des suites et séries holonomes est décidable : avec une description par équation et conditions initiales, il suffit de vérifier que les premiers coefficients en nombre convenable sont nuls. Cela en fait des objets intéressants pour le calcul formel, et ce d'autant plus que les classes des suites et séries holonomes sont closes par plusieurs opérations usuelles.

À nouveau, toutes ces propriétés de clôture sont effectives. La plupart sont implémentées dans `gfun`.

4. L'analogue en termes de décalés successifs de la condition (iii) de la définition des fonctions D-finies n'a pas de sens en présence de pôles entiers. Il devient correct pour les germes à l'infini de suites d'éléments de $\mathbb{K}$, auxquels la notion d'holonomie s'étend [**Sta80**].

PROPOSITION 3.15. — Les suites holonomes d'éléments de $\mathbb{K}$ forment une $\mathbb{K}$-algèbre. Les séries holonomes de $\mathbb{K}[[z]]$ forment une $\mathbb{K}$-algèbre close par dérivation, intégration, produit d'Hadamard

$$f \odot g = \sum_{n=0}^{\infty} f_n g_n z^n,$$

et en particulier transformée de Laplace formelle

$$\mathscr{L} f(z) = z \sum_{n=0}^{\infty} n! f_n z^n$$

directe et inverse.

PROPOSITION 3.16. — Les fonctions algébriques sont holonomes. Plus généralement, la classe des séries holonomes est close par substitutions algébriques sans terme constant : c'est-à-dire que si f est une série algébrique et g une série holonome, alors $g \circ f$ est holonome.

En revanche, l'inverse d'une série holonome ou la composée de deux séries holonomes ne sont holonomes que dans des cas très particuliers.

3.4. Équation indicielle et récurrence (coefficients polynomiaux)

Dans le cas d'une équation différentielle à coefficients polynomiaux, voyons maintenant comment déterminer les solutions au voisinage d'un point régulier (c'est-à-dire ordinaire ou singulier régulier) qui ne contiennent pas de facteur logarithmique. Pour obtenir tout du long des expressions explicites, on ne suppose plus que le point de développement est l'origine.

Si z_0 est un point régulier de l'équation, celle-ci s'écrit d'après le critère de Fuchs (théorème 3.11)

$$(z-z_0)^r a_r(z) y^{(r)}(z) + \cdots + (z-z_0) a_1(z) y'(z) + a_0(z) y(z) = 0 \tag{26}$$

où les a_i sont des polynômes et $a_r(z_0) \neq 0$. On en recherche les solutions séries formelles [5] d'exposant $\alpha \in \mathbb{C}$, *i.e.* s'écrivant

$$y(z) = (z-z_0)^{\alpha} \sum_{n=0}^{\infty} c_n^{[\alpha]} (z-z_0)^n, \tag{27}$$

avec $c_0^{[\alpha]} \neq 0$. On convient que $c_n^{[\alpha]} = 0$ pour $n < 0$.

En dérivant l'expression générale, on obtient :

$$y^{(j)}(z) = \sum_{n=0}^{\infty} c_n^{[\alpha]} \, (n+\alpha)(n+\alpha-1)\ldots(n+\alpha-j+1) \, (z-z_0)^{n-j+\alpha}$$

puis

$$(z-z_0)^j \, y^{(j)}(z) = \sum_{n=0}^{\infty} c_n^{[\alpha]} (n+\alpha)^{\downarrow j} \, (z-z_0)^{n+\alpha}.$$

Injectons cette dernière expression dans (26), et réécrivons les coefficients qui apparaissent comme des polynômes en $z-z_0$:

$$\sum_{n=0}^{\infty} \Big(\underbrace{(n+\alpha)^{\downarrow r} a_r(z) + \cdots + a_0(z)}_{=: u_d(n+\alpha)\,(z-z_0)^d + \cdots + u_0(n+\alpha),} \Big) c_n^{[\alpha]} (z-z_0)^n = 0$$

ce qui revient à poser

$$u_d(X)\,(z-z_0)^d + \cdots + u_0(X) = X^{\downarrow r} a_r(z) + \cdots + X\, a_1(z) + a_0(z),$$

5. En fait, les solutions que l'on obtient ainsi sont convergentes : voir [**Hen77**, théorème 9.5a], [**Inc56**, §16.2].

où $d = \max_i \deg a_i$ et les $u_j \in \mathbb{K}(z_0)[X]$. Notons que les u_i sont de degré borné par l'ordre r de l'équation différentielle. En extrayant le coefficient de $(z-z_0)^n$, il vient

$$\forall n \in \mathbb{Z}, \qquad u_0(n+\alpha)\, c_n^{[\alpha]} + \cdots + u_d(n-d+\alpha)\, c_{n-d}^{[\alpha]} = 0,$$

une récurrence vérifiée par les coefficients $c_n^{[\alpha]}$ d'un développement d'exposant α.

L'ordre de la récurrence est $s \leqslant d \leqslant r+D$, où D est le degré maximal des coefficients (polynomiaux) de l'équation. La première inégalité peut éventuellement être stricte, si jamais un des u_i s'annule ; la deuxième provient de ce que les degrés des a_i augmentent au plus de r par rapport aux degrés en z des termes correspondant de l'équation initiale.

On a convenu que $c_0^{[\alpha]}$ est le premier $c_n^{[\alpha]}$ non nul : aussi, pour $n = 0$, l'équation précédente se réduit à $u_0(\alpha) = 0$.

DÉFINITION 3.17. — L'équation $u_0(\alpha) = 0$ s'appelle l'équation indicielle de l'équation différentielle (26) en z_0.

On vient de montrer que les exposants possibles étaient parmi les racines de l'équation indicielle (en particulier, il y en a au plus r). Inversement, si α est une racine de u_0 telle que $u_0(\alpha+n) \neq 0$ pour n entier > 0, la récurrence admet une unique solution pour $c_0^{[\alpha]}$ donné, de sorte qu'il existe exactement une droite de solutions séries (27) d'exposant α. C'est vrai en particulier dans le cas où α est la « première » d'un ensemble de racines d'espacements entiers — on a donc trouvé autant de solutions de la forme (27) que le polynôme indiciel u_0 admet de racines distinctes modulo 1.

REMARQUE. — Si z_0 est un point ordinaire, pour tout i, $a_i(z)$ est divisible par $(z-z_0)^{r-i}$, et seul le terme $X^{\downarrow r} a_r(z)$ contribue à l'équation indicielle, qui est donc $\alpha^{\downarrow r} = 0$. Ses racines sont des entiers consécutifs, donc l'argument ci-dessus s'étend — on a une solution à la récurrence pour des conditions initiales spécifiées en ses entiers. On retrouve l'existence d'une base de solutions séries déterminées par leurs r premiers coefficients.

De même, en une singularité apparente, l'équation indicielle correspondante admet r solutions entières positives distinctes. (Cette condition n'est pas suffisante [**Inc56**, §16.4] : prendre $z^2y'' - 4y' + (4-z)y$.)

REMARQUE. — De façon analogue, les exposants maximaux de $z-z_0$ pouvant apparaître dans une solution série de Laurent à l'infini de la forme

$$(z-z_0)^\beta \sum_{n=-\infty}^{0} c_n (z-z_0)^n,$$

et en particulier dans une solution polynomiale, sont donnés par les racines de u_0 (si celui-ci est non nul, et du premier u_i non nul sinon).

CHAPITRE 4

ÉVALUATION ET PROLONGEMENT ANALYTIQUE NUMÉRIQUE

Nous voici maintenant au cœur du sujet. L'objet de ce chapitre est de montrer comment utiliser les algorithmes du chapitre 1 pour évaluer rapidement une fonction holonome (spécifiée par une équation différentielle et des conditions initiales) jusqu'à de grandes précisions de l'ordre de plusieurs dizaines de milliers de chiffres décimaux, le long d'un chemin de prolongement analytique quelconque.

Débutons par un exemple simple mais typique : $\arctan z$. Elle est solution de l'équation homogène $(1+z^2)y''+2zy'=0$, qui possède deux singularités complexes, en $\pm i$. Les algorithmes de ce chapitre, partiellement implémentés dans `NumGfun`, permettent tout d'abord d'en calculer des valeurs, dans le disque de convergence :

```
> deq := diffeqtohomdiffeq(holexprtodiffeq(arctan(z),y(z)),y(z))
> ac_eval(deq, y(z), [0,1/2], 30);
```

$$0.46364760900080611621425623 1461$$

ou en-dehors :

```
> ac_eval(deq, y(z), [0,1/2,3/4,5/4], 30);
```

$$0.89605538457134395617480071 8029$$

Il est aussi possible de calculer la matrice de prolongement analytique (au sens détaillé §4.2.1) d'une équation différentielle le long d'un chemin de prolongement analytique à sommets dans $\mathbb{Q}(i)$. Toujours avec l'équation vérifiée par $\arctan z$, on peut par exemple contourner la singularité en i pour obtenir les valeurs des solutions fondamentales en $2i$:

```
> transition_matrix(deq, y(z), [0,(1+I)/2,3*(1+I)/4,1+I,
  1/2+7*I/4,2*I], 10);
```

$$\begin{bmatrix} 1.0000000000 & 1.5707963267+0.5493061443I \\ 0. & -0.3333333333 \end{bmatrix}$$

En faisant de même le long de boucles entourant chaque singularité, on obtient une méthode de calcul numérique de la monodromie (voir §4.2.3).

Du point de vue des performances, toujours sur l'exemple de l'arctangente, le temps de calcul est comparable à celui de `evalf` pour une précision de 100 000 chiffres

```
> time(ac_eval(deq, y(z), [0,1/3], 100000)),
  time(evalf[100000](arctan(1/3)));
```

$$18.919, 19.609$$

et nettement plus faible si on en demande cinq fois plus (132 secondes de calcul contre 441). Mais il s'agit d'une fonction élémentaire pour laquelle Maple dispose d'un algorithme

spécifique (une variante de celui de [**Bre76a**]). La comparaison est plus favorable pour des fonctions moins particulières. Prenons le cas des fonctions d'onde sphéroïdales, solutions de

$$(1-z^2)y'' - 2(b+1)zy' + (c-4qz^2)y = 0,$$

qui interviennent en physique dans l'étude de l'équation des ondes en coordonnées elliptiques ou en coordonnées sphéroïdales :

```
> b := 1: c := 0: q := 1:
> time(ac_eval({(1-z^2)*diff(y(z),z,z) - 2*(b+1)*z*diff(y(z),z)
  + (c-4*q*z^2)*y(z), y(0)=1, D(y)(0)=0}, y(z), [0,1/3],1000));
```

0.532

Maple ne dispose pas à ma connaissance de fonction pour calculer numériquement avec une précision garantie (ou simplement à plus de quelques chiffres significatifs) des solutions d'équations holonomes arbitraires. Dans ce cas particulier, on peut cependant obtenir une valeur numérique par `evalf`, en exprimant la fonction définie ci-dessus à l'aide des fonctions de Heun (solutions générales, à des changements de variable près, de l'équation d'ordre 2 à 4 points singuliers réguliers) :

```
> time(evalf[1000](subs(z=1/3,HeunC(0,-1/2,b,q,1/4-b/4-c/4,z^2))));
```

1.572

Le scindage binaire est ici près de trois fois plus rapide dès la précision 1 000.

Théoriquement, les algorithmes à base de scindage binaire donnent la meilleure complexité connue pour la classe complète des fonctions holonomes. En revanche, comme dans le cas des constantes, il est fréquent que l'on sache faire mieux dans des cas particuliers. Notamment, des algorithmes à base de moyenne arithmético-géométrique [**Bre76b**] permettent d'évaluer à précision absolue 10^{-n} les fonctions élémentaires (exp, log, les fonctions trigonométriques directes et inverses) sur des arguments flottants en temps $O(\mathrm{M}(n)\log(n))$. Le scindage binaire égale cette complexité pour l'évaluation de séries « à convergence rapide » en des points pris dans un corps de nombres. Au-delà, l'alternative à base de scindage binaire est l'algorithme *bit burst*, qui s'applique à toutes les fonctions holonomes, mais demande $\Theta(\mathrm{M}(n\log n\log\log n))$ opérations dans les cas favorables, et $\Theta(\mathrm{M}(n\log^2 n\log\log n))$ opérations en général. La partie du manuel du logiciel YACAS consacrée aux algorithmes qu'il implémente [**Yac07**] décrit plusieurs algorithmes efficaces pour l'évaluation de nombreuses fonctions usuelles.

Mais revenons au résultats précédents, et voyons plus en détail comment on y aboutit.

4.1. Point algébrique dans le disque de convergence

Soit f une fonction holonome, spécifiée par une équation différentielle linéaire homogène à coefficients polynomiaux

$$a_r(z)f^{(r)}(z) + \cdots + a_0(z)f(z) = 0 \tag{28}$$

et des conditions initiales $f(z_0), \ldots, f^{(r-1)}(z_0)$ données en un point ordinaire z_0 de (28). On suppose que la singularité de l'équation la plus proche de z_0 se trouve à distance $\rho > 0$, de sorte (§3.1) que toute solution formelle de (28) a un rayon de convergence au moins égal à ρ. On cherche à calculer une approximation $\widetilde{f(z_1)}$ de f en un point z_1 à distance $|z_1 - z_0| < \rho$ de z_0, à la précision absolue

$$|\widetilde{f(z_1)} - f(z_1)| \leqslant \varepsilon = 10^{-p}.$$

Le point de départ z_0, le point d'arrivée z_1 et les coefficients des polynômes $a_0,\dots,a_r$ sont pris dans un corps de nombres [1] $\mathbb{K}\subset\mathbb{C}$. Il n'y a pas de difficulté supplémentaire à supposer en revanche les conditions initiales données par des programmes d'évaluation.

Pour calculer $f(z_1)$, on l'approxime par une somme partielle

$$\sum_{n=0}^{m-1} f_n^{[z_0]}\,(z_1-z_0)^n \tag{29}$$

du développement en série de f en z_0. Nous verrons au chapitre 5 comment déterminer automatiquement un nombre de termes m convenable. Bornons-nous pour l'instant à constater que d'après la règle de Cauchy donnant le rayon de convergence d'une série entière,

$$\frac{1}{\rho}\geqslant \limsup_{n\to\infty}|f_n^{[z_0]}|^{1/n}$$

de sorte qu'il suffit de prendre

$$m\sim \frac{1}{\log_{10}\frac{\rho}{|z_1-z_0|}}\left(p+\log_{10}\frac{1}{1-\frac{|z_1-z_0|}{\rho}}\right)\sim_{p\to\infty}\frac{p}{\log_{10}\frac{\rho}{|z_1-z_0|}} \tag{30}$$

pour obtenir

$$\Big|\sum_{n=m}^{\infty} f_n^{[z_0]}\,(z_1-z_0)^m\Big|\leqslant\varepsilon.$$

Notons d une borne sur les degrés des polynômes a_j apparaissant dans l'équation (28), ℓ une borne sur la taille en bits des points de départ z_0 et d'arrivée z_1, et $q=[\mathbb{K}:\mathbb{Q}]$ le degré de $\mathbb{K}$. La suite $f_n^{[z_0]}$ vérifie une récurrence linéaire à coefficients dans $\mathbb{K}[n]$,

$$b_s(n)\,f_{n+s}^{[z_0]}+\cdots+b_0 f_n^{[z_0]}=0, \tag{31}$$

d'ordre $s\leqslant r+d$, à coefficients de degré $\leqslant r$ (voir sa construction §3.4). Le terme général $u_n=f_n^{[z_0]}(z_1-z_0)^n$ de (29) satisfait donc

$$b_s(n)\,u_{n+s}+(z_1-z_0)b_{s-1}(n)\,u_{n+s-1}+\cdots+(z_1-z_0)^s b_0(n)\,u_n=0. \tag{32}$$

Après réduction de toute l'équation au même dénominateur, les entiers apparaissant dans (32) sont de taille $O(h)$. À partir de cette récurrence, on peut calculer les sommes partielles (29) au moyen de l'algorithme par scindage binaire (§1.3) en utilisant une des deux traductions matricielles décrites §2.3. D'après la proposition 2.1, la complexité du calcul est

$$\begin{aligned}&\leqslant\big((2q-1)L+1\big)\,\mathrm{M}\Big(\frac{m}{2}\big(r\log_2^2 m+O(\ell\log m)\big)\Big)\big(1+o(1)\big)\\ &\qquad\simeq\frac{\big((2q-1)L+1\big)}{2\log_{10}\frac{\rho}{|z_1-z_0|}}\,\mathrm{M}\Big(p\big(r\log_2^2 p+O(\ell\log p)\big)\Big)\big(1+o(1)\big)\end{aligned} \tag{33}$$

où $p,\ell\to\infty$. Dans ces formules, L représente la complexité en nombre de multiplications d'entiers du produit de matrices de rationnels de taille $s+1$ denses ou de la forme (19). D'après les sections 2.2.2 et 2.3, on peut prendre

$$L\leqslant s^2\left\lceil\frac{s}{2}\right\rceil+(2s-1)\left\lfloor\frac{s}{2}\right\rfloor+s^2+s+1.$$

1. Dans l'implémentation, je me suis limité à $\mathbb{K}=\mathbb{Q}(i)$. Ce choix a plusieurs raisons : premièrement, la manipulation efficace de matrices d'algébriques semble, après quelques essais, assez délicate en Maple. Deuxièmement, c'est une restriction moins importante qu'on pourrait le croire, car l'algorithme *bit burst* (§4.4) fournit une alternative, peut-être même plus efficace, pour l'évaluation en des points quelconques de solutions d'équations dont les *coefficients* sont dans $\mathbb{Q}(i)$.

La constante [2] cachée dans l'expression $O(\ell \log p)$ provient essentiellement des réductions au même dénominateur dans la récurrence (32).

Pour des séries de rayon de convergence infini (« à convergence rapide »), d'après le théorème 3.10, on peut prendre $m = O(p/\log p)$ au lieu de $m = \Theta(p)$. La complexité totale du calcul diminue en conséquence de $\Theta(\mathrm{M}(p(\log^2 p + \ell \log p)))$ à $O(\mathrm{M}(p(\ell + \log p)))$.

4.2. Prolongement analytique

L'algorithme précédent s'étend pour évaluer f en n'importe quel point de sa surface de Riemann. Suivant l'idée énoncée §3.1.2, à partir des conditions initiales données en $z_0 = 0$ dans la définition de f, on calcule de nouvelles conditions initiales définissant la même fonction au voisinage d'un point z_1 du disque de convergence, puis en un point z_2 du disque de convergence des séries solutions en z_1, et ainsi de suite aussi longtemps que nécessaire.

4.2.1. Matrices de passage. — Plus précisément, on calcule à chaque fois une matrice de passage de z_i à z_{i+1}, au sens suivant.

CONVENTION. — On appelle *base canonique* de solutions en z_0 de (28) la base

$$(f_{[z_0,0]}, \dots, f_{[z_0,k]})$$

des solutions séries de Laurent donnée par les conditions suivantes :

$$\operatorname{val} f_{[z_0,0]} < \cdots < \operatorname{val} f_{[z_0,k]} \quad \text{et} \quad [z^{\operatorname{val} f_{[z_0,i]}}] f_{[z_0,j]} = \mathbf{1}[i=j]\,.$$

Cette formulation a un sens y compris pour les solutions *séries* (sans facteurs logarithmiques) en un point singulier régulier. En une singularité apparente, on a $k = r-1$. En un point ordinaire, cette définition revient simplement à imposer que

$$\begin{array}{lcccccc}
f_{[z_0,0]}(z) & = & 1 & & & & +a_{0,1}(z-z_0)^r + \cdots \\
f_{[z_0,1]}(z) & = & & (z-z_0) & & & +a_{0,2}(z-z_0)^r + \cdots \\
& & & & \ddots & & \\
f_{[z_0,r-1]}(z) & = & & & & (z-z_0)^{r-1} & +a_{0,r-1}(z-z_0)^r + \cdots
\end{array}$$

On appellera *solutions fondamentales* de l'équation les éléments de la base canonique de solutions, et *conditions initiales* en z_0 les coordonnées d'une solution dans la base canonique en z_0.

Si z_1 est un point du disque de convergence des solutions séries en z_0, la matrice de passage $M_{z_0 \to z_1}$ de z_0 à z_1 est définie par

$$M_{z_0 \to z_1} = \left[\frac{1}{i!} f^{(i)}_{[z_0,j]}(z_1) \right]_{i,j} = \begin{bmatrix} f_{[z_0,0]}(z_1) & \cdots & f_{[z_0,r-1]}(z_1) \\ f'_{[z_0,0]}(z_1) & \cdots & f'_{[z_0,r-1]}(z_1) \\ \vdots & & \vdots \\ \frac{1}{(r-1)!} f^{(r-1)}_{[z_0,0]}(z_1) & \cdots & \frac{1}{(r-1)!} f^{(r-1)}_{[z_0,r-1]}(z_1) \end{bmatrix}$$

C'est la matrice de l'application « prolongement analytique » le long du segment $[z_0, z_1]$, exprimée dans les bases canoniques respectives des espaces de solutions locales en z_0 et z_1, au sens de la convention ci-dessus [3]. C'est aussi (éventuellement à des facteurs constants près) l'évaluation en z_1 d'une matrice fondamentale en z_0 du système associé à (28). Naturellement,

2. On peut la borner par $2q(d+1)(s+1)^2 \leqslant 2q(d+1)(r+d+1)^2$, mais la signification concrète de cette borne est douteuse...

3. Si l'on identifie les solutions qui sont des prolongements les unes des autres, c'est encore la matrice de passage entre la base canonique en z_0 et la base canonique en z_1, vues comme deux bases de l'espace des solutions maximales (sur un ouvert simplement connexe, disons).

la notion s'étend au prolongement analytique le long d'un chemin quelconque, et on a alors la règle de composition $M_{z_0\to z_1\to z_2} = M_{z_1\to z_2} M_{z_0\to z_1}$.

4.2.2. Calcul. — La matrice $M_{z_0\to z_1}$ (où z_1 est dans le disque de convergence des solutions en z_0) peut être calculée en appliquant l'algorithme d'évaluation par scindage binaire aux fonctions $f_{[z_0,j]}$ et à leurs dérivées — qui sont elles-mêmes des fonctions holonomes. L'algorithme de prolongement analytique numérique le long d'un chemin polygonal

$$0 = z_0 \to z_1 \to z_2 \to \cdots \to z_m$$

d'une fonction f se résume essentiellement à calculer ainsi les matrices de passage correspondant à chaque pas puis à former leur produit

$$M_{z_{m-1}\to z_m} \cdots M_{z_1\to z_2} \cdot M_{z_0\to z_1}.$$

L'organisation des calculs et le contrôle de la précision demandent néanmoins un peu de soin.

Les versions détaillées ci-dessous suivent de près l'implémentation. L'équation (28) est fixée dans tout l'algorithme, de même que le corps de base $\mathbb{K}$. On suppose précalculée une récurrence « générique » vérifiée par les coefficients de $f_{[a,j]}(z)$, exprimée en fonction du point a (et, pour ce qui est des conditions initiales, de l'indice j de la solution). Pour chaque pas de prolongement $z_i \to z_{i+1}$, on suppose aussi disponible une série majorante pour les $f_{[z_i,\cdot]}$, et une borne correspondante sur les restes des $f_{[z_i,\cdot]}(z_{i+1})$. (Le calcul de ces bornes fait l'objet du chapitre suivant.) Enfin, il est entendu tout du long que les opérations (produits matriciels, reconstruction de matrices à partir de lignes) sont faites en gardant séparés numérateurs et dénominateurs pour éviter les calculs de pgcd.

PROCÉDURE (Matrice de passage pour un pas). — Entrée : un pas de prolongement $z_0 \to z_1$, une précision absolue ε. Sortie : une approximation numérique $\tilde{M}$ de $M_{z_0\to z_1}$ telle que $|||\tilde{M} - M_{z_0\to z_1}|||_2 \leqslant \varepsilon$.

S1. Si $z_0 = z_1$, renvoyer l'identité.

S2. Calculer une récurrence (R) annulant la suite a_n des coefficients de Taylor des $f_{[z_0,\cdot]}(z)$ en spécialisant la récurrence générique par la substitution $a = z_0$. Soit s son ordre.

S3. Pour k (ordre de dérivation) de 0 à $r-1$

   a. Calculer la matrice $s \times r$ de conditions initiales de la récurrence

$$I = \left[\frac{1}{k!} (f^{(k)}_{[z_0,\cdot]})_i \, (z_1 - z_0)^i \right]_{\substack{i\in[\![0,s-1]\!] \\ j\in[\![0,r-1]\!]}}$$

   b. Calculer le nombre N de termes nécessaires pour évaluer $f^{(k)}(z_1)$ à précision ε/r à l'aide des algorithmes du chapitre suivant.

   c. Calculer par scindage binaire le vecteur ligne L représentant la forme linéaire « N1-ième terme » (en fonction des s premiers) des solutions de (R).

   d. Calculer $\ell_k := L \cdot I$.

   e. Remplacer (R) par une récurrence annulant $(n+1)a_{n+1}$, c'est-à-dire les coefficients de Taylor des $f^{(k+1)}_{[z_0,\cdot]}$. Ajuster les conditions initiales (en déroulant la récurrence pour calculer a_{r+k}).

S4. Renvoyer la matrice de lignes $\ell_1, \ldots, \ell_{r-1}$.

En n'effectuant que la première itération de la boucle, on obtient la matrice ligne des valeurs des solutions fondamentales en z_1. Multipliée par un vecteur colonne de conditions initiales en z_0, elle donne la valeur en z_1 de la solution correspondante.

Pour obtenir la matrice de passage le long du chemin de prolongement complet, on appelle la procédure précédente pour chaque pas, et on multiplie par scindage binaire les matrices obtenues. L'estimation de l'erreur sur un produit de matrices connues approximativement

$$|||\tilde{A}\tilde{B} - AB||| \leqslant |||\tilde{A}|||\,|||\tilde{B} - B||| + |||\tilde{B}|||\,|||\tilde{A} - A|||$$

justifie les ajustements de précision effectués pour les appels récursifs.

PROCÉDURE (Matrice de passage le long d'un chemin de prolongement analytique)
Entrée : un chemin $P = z_0 \to \cdots \to z_m$, une précision absolue ε. Sortie : une approximation numérique $\tilde{M}$ de M_P telle que $|||\tilde{M} - M_P|||_2 \leqslant \varepsilon$.

P1. Si $m \leqslant 0$, renvoyer l'identité. Si $m = 1$, appeler la procédure S et renvoyer le résultat.

P2. Soient $P[0] = z_0 \to \cdots \to z_{\lfloor m/2 \rfloor}$ et $P[1] = z_{\lfloor m/2 \rfloor + 1} \to \cdots \to z_m$. Pour $i \in \{0; 1\}$
   a. Calculer une borne B_{1-i} sur $|||M_{P[1-i]}|||_2$ (voir ci-dessous).
   b. Calculer récursivement la matrice $M_{P[i]}$ à précision $\varepsilon/(2B_{1-i})$.

P3. Renvoyer $M_P := M_{P[1]} M_{P[0]}$.

Enfin, pour obtenir une valeur en z_m, il suffit de calculer la matrice de passage de z_0 à z_{m-1}, la première ligne de celle de z_{m-1} à z_m, d'en faire le produit et de multiplier le résultat par le vecteur des conditions initiales, au sens des conventions de la section précédente.

PROCÉDURE (Évaluation numérique). — Entrée : un chemin $z_0 \to \cdots \to z_m$, une équation différentielle (E) non singulière en z_0, avec des conditions initiales $f(z_0), \ldots, f^{(r-1)}(z_0)$ définissant une unique solution f, un nombre de chiffres décimaux $p \geqslant 1$. Sortie : une approximation $\widetilde{f(z_m)}$ de $f(z_m)$ telle que $|\widetilde{f(z_m)} - f(z_m)| \leqslant 10^{-p}$.

E1. Calculer le vecteur $I = {}^t[f^{(j)}(z_0)/j!]_{j \in [\![0, r-1]\!]}$ des conditions initiales définissant f.

E2. Poser $\varepsilon = 10^{-p}/(m\,\|I\|_2)$. Calculer une borne B sur $|||M_{z_0 \to \cdots \to z_{m-1}}|||_2$ (voir ci-dessous).

E3. Calculer la première ligne Q de $M_{z_{m-1} \to z_m}$ à précision $< \varepsilon/B$ en appelant la procédure S.

E4. Calculer la matrice $M_{z_0 \to \cdots \to z_{m-1}}$ à précision $< (m-1)\varepsilon/\,\|Q\|_2$ en utilisant la procédure P.

E5. Calculer $L = Q \cdot M_{z_0 \to \cdots \to z_{m-1}}$. C'est une matrice ligne qui représente la forme linéaire qui, à des conditions initiales exprimées dans la base canonique en z_0, associe la valeur de la solution qu'elles définissent en z_m.

E6. Calculer une valeur approchée $\tilde{I} \in \mathbb{K}^r$ de I à précision $< \varepsilon/\,\|L\|_2$.

E7. Renvoyer une approximation flottante à p chiffres après la virgule de $L \cdot I$.

Pour calculer les bornes sur les matrices de passage, une solution est d'utiliser les majorants du chapitre suivant. Précisément, si $g(z)$ est une série majorante pour $f(z_i + z)$, alors

$$|||M_{z_i \to z_{i+1}}|||_2 \leqslant r \max_{j=0}^{r-1} \big(g^{(j)}(|z_{i+1} - z_i|)\big),$$

et on obtient une borne sur la matrice de passage le long d'un chemin quelconque en multipliant des facteurs de ce type correspondant à chaque pas [4]. Une autre possibilité, *a priori* plus fine mais plus délicate à implémenter, consisterait à calculer à petite précision la matrice de passage que l'on souhaite borner.

La complexité globale du prolongement analytique est (essentiellement) de $(m-1)r + 1$ appels à l'algorithme d'évaluation par scindage binaire pour le calcul de la valeur en z_m de n'importe quelle solution, et de mr appels pour celui de la matrice de passage complète. Mais le coût des étapes est variable, et dépend du choix du chemin. Nous reviendrons là-dessus §4.3.

4. Une façon de faire commode est de mimer la structure récursive de l'algorithme de prolongement analytique en remplissant une table de mémoïsation pour ne calculer qu'une fois la borne pour chaque pas.

4.2.3. Application : monodromie numérique. — Considérons une équation différentielle linéaire à coefficients analytiques

$$a_r y^{(r)} + \cdots + a_0 y = 0. \tag{34}$$

Notons $S \subset \mathbb{C}$ l'ensemble de ses points singuliers, supposés isolés. Soit $\gamma : [0;1] \to \mathbb{C} \setminus S$ un lacet de base $\gamma(0) = z_0$, et soit f une solution de l'équation au voisinage de z_0. On peut prolonger f le long de γ suivant la procédure de la section 3.1.2, obtenant ainsi une fonction g, définie sur un voisinage de z_0 et solution de (34). Si γ est homotope au lacet nul, g coïncide avec f, mais si γ entoure une ou plusieurs singularités, g est en général une solution de (34) différente de f.

Le prolongement analytique le long de γ, ou de tout lacet homotope à γ, définit un isomorphisme $f \mapsto g$ de l'espace $E \simeq \mathbb{C}^r$ des solutions en z_0. La représentation linéaire du groupe fondamental de point base z_0 de $\mathbb{C} \setminus S$

$$\begin{array}{rcl} M : \pi_1(\mathbb{C} \setminus S, z_0) & \to & GL(E) \\ \gamma & \mapsto & f \mapsto g \end{array}$$

ainsi définie est appelée monodromie [5] (de base z_0) de l'équation (34). Son image $\operatorname{im} M \subset GL(E)$ est appelée le groupe de monodromie de l'équation. À conjugaison près, ces notions ne dépendent pas du point base.

Si ζ est une singularité et si γ est un lacet (de base z_0) qui fait une fois le tour de cette singularité, en laissant les autres à l'extérieur (formellement, d'indice 1 par rapport à ζ et d'indice nul par rapport aux autres singularités), on appelle monodromie locale en ζ la transformation $M(\gamma)$ subie par les solutions lors du prolongement analytique le long de γ. Les monodromies locales (de même point base) en les différentes singularités engendrent le groupe de monodromie. La matrice de passage d'un point ordinaire z_0 à lui-même le long d'un chemin contournant une singularité est précisément la matrice de la monodromie locale dans la base canonique de solutions en z_0. Ainsi, l'algorithme de prolongement analytique de ce chapitre fournit un moyen de calculer numériquement la monodromie locale des équations à coefficients polynomiaux :

```
> deq := diffeqtohomdiffeq(holexprtodiffeq(arctan(z),y(z)),y(z));
```

$$\left\{ y(0) = 0, \mathtt{D}(y)(0) = 1, -2z\frac{d}{dz}y(z) + \left(-1-z^2\right)\frac{d^2}{dz^2}y(z) \right\}$$

```
> local_monodromy(deq, y(z), I, 0, 30);
```

$$\begin{bmatrix} 1.00000000000000000000000000000 & 3.14159265358979323846264338327\\ 0 & 1.00000000000000000000000000000 \end{bmatrix}$$

```
> local_monodromy(deq, y(z), -I, 0, 30);
```

$$\begin{bmatrix} 1.00000000000000000000000000000 & -3.14159265358979323846264338327\\ 0 & 1.00000000000000000000000000000 \end{bmatrix}$$

```
> b := 1: c := 1: d := 1: q := 1
> deq := {(1-z^2)*diff(y(z),z,z) - 2*(b+1)*z*diff(y(z),z)
  + (c-4*q*z^2)*y(z), y(0)=1, D(y)(0)=0}:
  local_monodromy(deq, y(z), 1, 0, 10);
```

$$\begin{bmatrix} 0.9999999999 - 0.2267571137I & 1.4105962414I\\ -0.0364518117I & 1.0000000000 + 0.2267571137I \end{bmatrix}$$

5. Sur la notion de monodromie, et pour quelques exemples de calculs *analytiques* de groupes de monodromie, voir [**vdPS03**, chap. 1 et 5] ou [**Inc56**, §15.9].

Une autre méthode de calcul de la monodromie d'une fonction holonome utilisant l'évaluation numérique par scindage binaire est esquissée dans [**vdH01**, §4.2.5]. Dans le cas particulier important des fonctions algébriques (où les matrices de monodromie sont des matrices de permutation, et peuvent donc être calculées exactement), un certain nombre d'approches concurrentes existent : voir le compte-rendu de l'exposé d'A. Poteaux annexe B.

4.3. Chemins de prolongement

On a vu que le nombre de termes de la série de Taylor à sommer pour l'évaluation d'une fonction holonomes était fonction du rapport des distances du point de développement en série au point d'évaluation et à la singularité de l'équation la plus proche. La complexité du prolongement analytique numérique dépend donc fortement du chemin suivi, alors que le résultat ne dépend que de la classe d'homotopie de ce chemin. Il y a un compromis à faire entre nombre et longueur des pas de prolongement d'une part, et rayon de convergence des solutions au point de départ de chaque pas d'autre part, pour trouver une « bonne » ligne polygonale qui discrétise un chemin donné.

L'objectif serait d'obtenir un programme qui, à partir d'une description minimale d'un point sur la surface de Riemann d'une fonction (soit par un chemin quelconque, soit en termes de branches de la fonction relatives à des coupes conventionnelles du plan complexe données dans sa description), choisisse automatiquement un chemin « optimal » pour atteindre ce point à partir de celui où la fonction est connue. Il est possible de donner une réponse relativement satisfaisante à ce problème, au moins dans le cas où l'équation ne possède qu'une singularité. (Cependant, le choix automatique du chemin n'est pas implémenté dans la version actuelle de `NumGfun`.) Nous allons voir en particulier que même à l'intérieur du disque de convergence, il est souvent préférable d'évaluer une fonction en plusieurs pas de prolongement analytique, surtout si le point visé est proche du bord du disque.

4.3.1. Équation avec une seule singularité finie. — Sans perte de généralité, supposons que la singularité se trouve à l'origine, et que les conditions initiales sont données au point 1.

DÉFINITION 4.1. — Soit $\gamma : [0,1] \to \mathbb{C}$ un arc rectifiable non nécessairement fermé. On appellera indice de γ autour de $a \notin \operatorname{im}\gamma$ le réel

$$\operatorname{ind}_a \gamma = \frac{1}{2\pi i} \int_\gamma \frac{\mathrm{d}z}{z-a}.$$

Intuitivement, $\operatorname{ind}_a \gamma$ représente le nombre de tours que fait γ autour de a. On a

$$2\pi \operatorname{ind}_a \gamma \equiv \arg \frac{\gamma(1)-a}{\gamma(0)-a} \quad [2\pi].$$

L'indice caractérise la classe d'homotopie à extrémités fixées de γ dans $\mathbb{C} \setminus \{a\}$.

HEURISTIQUE. — Pour calculer le prolongement analytique de f de 1 à $K \in \mathbb{C}$ le long d'un chemin d'indice φ, utiliser le chemin polygonal

$$1 \to (1+\tilde{w}) \to (1+\tilde{w})^2 \to \cdots \to (1+\tilde{w})^{m-1} \to K$$

où $w \in \mathbb{C}$ est l'unique solution de la forme

$$w = e^{(\log|K| + i\varphi)t} - 1, \quad t \in [0;1] \tag{35}$$

de l'équation

$$|w| \log|w| + \operatorname{Re}\left(|w| \frac{1+w}{w} \log(1+w)\right) = 0, \tag{36}$$

et $\tilde{w} \in \mathbb{Q}[i]$ est une approximation de petite taille de w. (Voir figure 6.)

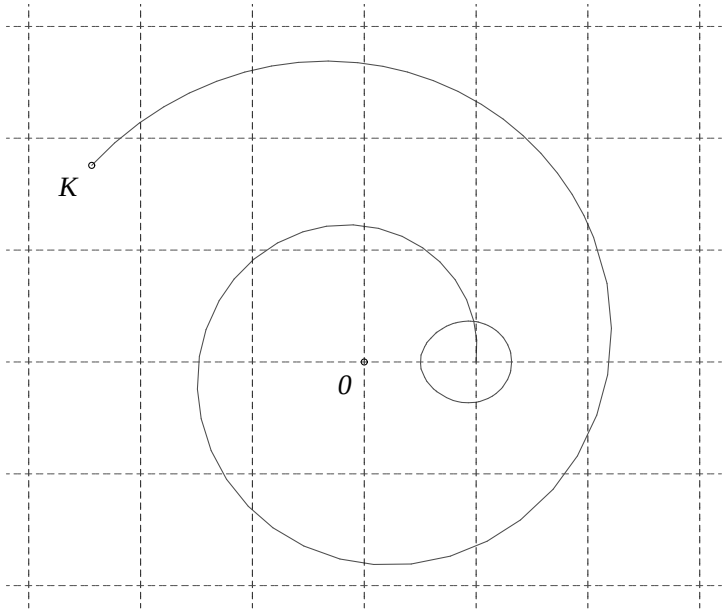


FIG. 6. Choix du chemin de prolongement analytique. *Le rapport* $1+w$ *de deux points successifs est l'intersection de l'ovale d'équation* (36)*, et de la spirale paramétrée par* (35) *(au changement de coordonnées* $z=1+w$ *près). Cette dernière représente une version « continue » du chemin de prolongement.*

Intuition. — Considérons un chemin de prolongement

$$z_0 = 1 \to z_1 \to \cdots \to z_m = K.$$

Le coût d'un pas de prolongement analytique de z_j à z_{j+1} est approximativement proportionnel au nombre de termes de la série de Taylor de la solution que l'on somme, lui-même donné par (30). Le nombre total de termes calculés lors du prolongement analytique est asymptotiquement proportionnel à

$$\sum_{j=0}^{m-1} \frac{1}{\log \frac{|z_j|}{|z_{j+1}-z_j|}} \tag{37}$$

quand la précision d'évaluation grandit.

Posons

$$w_j = \frac{z_{j+1}-z_j}{z_j} = r_j e^{i\theta_j} \qquad \text{et} \qquad g(r) = \frac{1}{\log r}.$$

Minimiser (37) se ramène à *maximiser* $\sum_{j=1}^{m} g(r_j)$ sous les contraintes

$$r_j \in [0;1[\,, \qquad \theta_j \in \,]-\pi,\pi[\,, \qquad f(\vec{r},\vec{\theta}) := \prod_{j=1}^{m}(1+r_j e^{i\theta_j}) - K = 0.$$

Pour un m fixé, introduisons le lagrangien

$$\Lambda(\vec{r},\vec{\theta},\lambda,\mu) = \sum_{j=1}^{m} g(r_j) + \lambda \operatorname{Re} f(\vec{r},\vec{\theta}) + \mu \operatorname{Im} f(\vec{r},\vec{\theta}), \quad \lambda,\mu \in \mathbb{R},$$

qui s'écrit encore

$$\Lambda = \sum_{j=1}^{m} g(r_j) + \operatorname{Re}(\alpha f) \qquad \text{avec} \qquad \alpha = \lambda - i\mu.$$

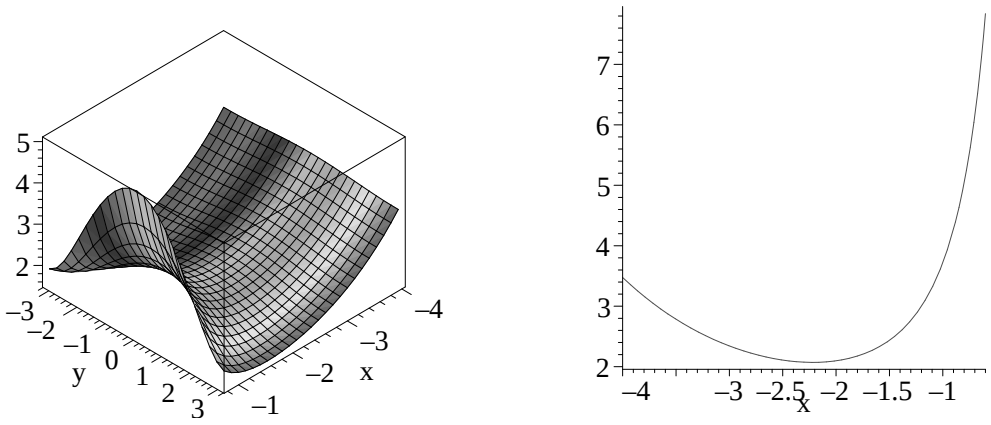


FIG. 7. La fonction $h : w \mapsto (1+w)/(w\log^2|w|)$. *À gauche,* $|h(w)|$ *(et* $\arg h(w)$ *en nuances de gris) en fonction de* $x = \log|w|$ *et* $y = \arg w$. *À droite, coupe pour* $w \in \mathbb{R}_+^*$.

Notons

$$P_j = \prod_{k\neq j}(1+r_k e^{i\theta_k}) = \frac{K}{1+w_j}.$$

En un maximum local, on a

$$\frac{\partial\Lambda}{\partial r_j} = g'(r_j) + \mathrm{Re}(\alpha e^{i\theta_j} P_j) = 0 \tag{38}$$

$$\frac{\partial\Lambda}{\partial\theta_j} = \mathrm{Re}(\alpha e^{i\theta_j} i r_j P_j) = 0. \tag{39}$$

L'équation (39) entraîne que $\alpha e^{i\theta_j} P_j \in \mathbb{R}$, et (38) se réécrit donc $g'(r_j) + \alpha e^{i\theta_j} P_j = 0$, soit encore

$$\forall j, \quad \frac{1+w_j}{w_j}|w_j| g'(|w_j|) = -\alpha K.$$

Il suffirait que la fonction de w_j au membre gauche soit injective pour conclure que w_j est indépendant de j. Malheureusement, pour la fonction de coût $g(r) = 1/\log r$ qui nous intéresse, c'est faux (figure 7).

Admettons cependant que les w_j sont constants, *i.e.* que le chemin optimal est donné par

$$z_j = (1+w)^j, \quad j = 0,\ldots,m$$

pour un $w = re^{i\theta}$. Il s'agit alors de déterminer la « bonne » valeur de m. En faisant maintenant de Λ une fonction de m, l'analogue du calcul précédent donne

$$\frac{\partial\Lambda}{\partial r} = m g'(r) + \mathrm{Re}(m\alpha e^{i\theta}(1+w)^{m-1}) = 0 \tag{40}$$

$$\frac{\partial\Lambda}{\partial\theta} = \mathrm{Re}(i\alpha w(1+w)^{m-1}) = 0 \tag{41}$$

$$\frac{\partial\Lambda}{\partial m} = g(r) + \mathrm{Re}(\alpha(1+w)^m \log(1+w)) = 0. \tag{42}$$

De (40) et (41), on tire $e^{-i\theta} g'(r) = \alpha(1+w)^{m-1}$, d'où en reportant dans (42)

$$g(r) + \mathrm{Re}(e^{-i\theta} g'(r)(1+w)\log(1+w)) = 0$$

qui devient (36) en remplaçant $g(r)$ par sa valeur. La valeur de K et la classe d'homotopie du chemin recherché (donnée par φ) finissent de déterminer w. □

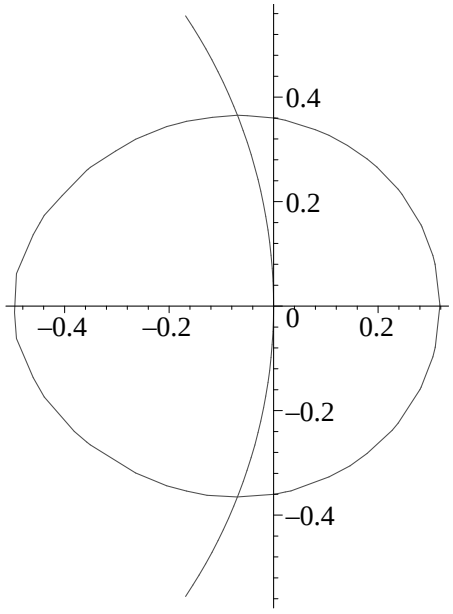


FIG. 8. Chemins « optimaux » particuliers. *Aux intersections de l'ovale avec l'axe des abscisses, on trouve les valeurs de w pour un chemin radial s'éloignant ou s'approchant de la singularité : respectivement* $w \simeq 0{,}32$ *et* $w = -0{,}5$. *À l'intersection avec le cercle de centre* -1 *et de rayon* 1, *on lit* $w \simeq -0{,}07 + 0{,}36i \simeq e^{2\pi i/17} - 1$, *qui donne le pas de discrétisation approprié pour contourner la singularité.*

Cette règle est énoncée pour le *prolongement analytique*. Pour l'*évaluation* d'une fonction, il faut prendre en compte le coût moindre du dernier pas, où l'on ne calcule que la première ligne de la matrice de passage.

4.3.2. Cas particuliers. — Pour un chemin qui s'éloigne radialement de la singularité (K réel > 1), l'heuristique commande de prendre $\theta = 0$ et l'équation (36) se simplifie en

$$(1+w)\log(1+w) + w\log w = 0,$$

équation équivalente à celle donnée par Chudnovsky et Chudnovsky dans ce cas [**CC87**, prop. 4.1] [**CC90**, prop. 4.3]. Numériquement, on obtient $1 + w \simeq 1{,}318$ (voir figure 8). De même, pour un chemin allant droit sur la singularité ($0 < K < 1$, $\theta = \pi$), il vient

$$(w-1)\log(w-1) + w\log w = 0,$$

d'où $1 + w = 1/2$, comme indiqué par van der Hoeven [**vdH99**, §4.1]. Enfin, si le point de départ et le point d'arrivée sont situés sur un cercle centré en la singularité, la valeur de $1+w$ obtenue est très proche de $e^{2\pi i/17}$: on retrouve le fait, mentionné dans les deux articles cités, que le chemin optimal pour faire un tour complet autour de la singularité est un heptadécagone régulier. Sans surprise, le coût d'un tour est indépendant du rayon du cercle. Cette règle englobe donc celles données précédemment pour le prolongement analytique optimal des fonctions holonomes [6].

6. J. van der Hoeven avait cependant remarqué que l'heuristique proposée ici était une généralisation naturelle des recettes qu'il donnait pour le cas d'un chemin radial ou circulaire.

4.3.3. Remarques. — Le raisonnement s'applique en l'état dès que l'on cherche à minimiser la somme des ordres de développement le long d'un chemin de prolongement, indépendamment de l'algorithme de prolongement ou du fait que la fonction à prolonger soit holonome. En passant, il dépend aussi beaucoup plus de la contrainte sur la forme du chemin que de la fonction de coût g. Il est tentant de conjecturer que les chemins obtenus sont effectivement les chemins optimaux dans le cas d'une singularité. (Les cas particuliers mentionnés ci-dessus sont donnés sans preuve détaillée, et je n'ai pas réussi à en reconstituer de preuve complète. Les problèmes que j'ai rencontrés sont pour l'essentiel les mêmes que ceux qui font que « l'intuition » ci-dessus n'est pas une démonstration.)

Par ailleurs, suivant une remarque d'A. Poteaux dans le cadre de la monodromie des fonctions algébriques [**Pot07**], il peut être préférable de parcourir « à l'envers » un chemin de prolongement qui s'éloigne de la singularité, puis d'inverser la matrice de passage obtenue. Le coût élevé de l'inversion de matrices de grands rationnels et les problèmes supplémentaires de contrôle de la précision qui apparaissent rendent cependant douteux l'intérêt de cette stratégie pour l'évaluation numérique à précision arbitraire.

Le problème du choix de chemins est plus difficile en présence de plusieurs singularités, mais une heuristique raisonnable dans le cas général est nécessaire pour écrire un programme qui évite à l'utilisateur de spécifier explicitement les chemins. L'estimation de complexité (37) demeure valable pour des pas dont le point de départ se situe dans la cellule de Voronoï d'une même singularité. Une approche possible serait de chercher un plus court chemin dans le graphe formé du point de départ, du point d'arrivée et des points du plan équidistants de deux (ou de deux ou trois) singularités, reliés par tous les chemins « en spirale » décrits plus haut possibles. On voit cependant dès l'exemple de deux singularités que cela ne donne pas toujours des chemins proches de l'optimal. Dans le cas particulier de la monodromie, une variante intéressante consisterait à prendre, pour chaque singularité a, le plus grand cercle centré en a contenu dans la cellule de Voronoï (fermée) de a, et à rendre le chemin formé par la réunion de ces cercles connexe en les reliant le long des arêtes de la triangulation de Delaunay, de façon à minimiser le coût total de prolongement le long des segments ajoutés (c'est un calcul d'arbre couvrant minimal).

4.4. *Bit burst*

L'algorithme d'évaluation par scindage binaire (§4.1), et partant l'algorithme de prolongement analytique numérique (§4.2) ne sont efficaces que quand la taille en bits du point d'évaluation (ou, pour le prolongement analytique, des points d'évaluation successifs) est modérée. Quand la taille ℓ de z_0, z_1 tend vers l'infini, la complexité (33) de l'évaluation par scindage binaire n'est plus quasi-optimale, et devient même relativement mauvaise. Concernant les points intermédiaires du prolongement analytique, une petite perturbation par homotopie du chemin suffit généralement à réduire leur taille. Reste le point d'arrivée : les algorithmes vus pour l'instant sont limités à l'évaluation en des points à coordonnées rationnelles de petite taille.

Une idée simple permet de les étendre à l'évaluation à précision n en des points à coordonnées également de taille n. Il s'agit de l'algorithme *bit burst* [**CC90, vdH99**]. Le principe est le suivant : pour évaluer une fonction holonome f en un point z_m de taille binaire $\Theta(n)$, même situé dans le disque de convergence, on fait le calcul le long d'un chemin de prolongement analytique formé d'approximations de z_m dont la précision double à chaque pas [7]. Le nom vient de ce qui se passe si z_m est, disons, un rationnel dyadique compris entre 0 et 1, et si l'on prend comme approximations successives les troncatures à l'ordre $1, 2, \ldots, 2^{m-1}$ du

7. Naturellement, la même idée s'applique si le point de départ du prolongement analytique, celui où sont données les conditions initiales, est un point de grande taille.

développement binaire

$$z_m = 0{,}a_1a_2a_3\ldots$$

de z_m — pour reprendre l'expression de [**CC90**] : *"adding more bits of a number, but computing with the same accuracy"*.

La diminution des longueurs des pas de prolongement analytique, et donc des nombres de termes à sommer, équilibre la croissance géométrique des tailles des sommets du chemin de prolongement. Par cet algorithme, la complexité de l'évaluation à $O(n)$ chiffres d'une fonction holonome en un point de hauteur $O(n)$ d'un corps de nombres est [**vdH99, vdH07**] celle mentionnée en introduction, soit

$$O(\mathrm{M}(n\log^2 n\log\log n))$$

en général, et

$$O(\mathrm{M}(n\log n\log\log n))$$

pour les séries de rayon de convergence infini.

REMARQUE. — Le *bit burst* peut être vu comme une généralisation d'une méthode appelée « astuce de Brent » [**Bre76a**] (voir aussi [**BB87**, §10.2, exercice 8]) pour le calcul de l'exponentielle. Dans ce cas, le produit des matrices de passage se simplifie en la règle de calcul :

$$e^{z_m} = e^{z_1+(z_2-z_1)+(z_3-z_2)+\cdots+(z_m-z_{m-1})} = e^{z_1}\cdot e^{z_2-z_1}\cdot e^{z_3-z_2}\cdots e^{z_m-z_{m-1}}$$

et son interprétation comme un prolongement analytique n'est pas apparente.

À ce stade, on dispose d'un algorithme pour évaluer efficacement les fonctions holonomes en des points eux-mêmes spécifiés par des approximations décimales à grande précision. Une dernière généralisation serait de savoir les évaluer en des points donnés par des *programmes d'évaluation* capables d'en fournir des décimales à la demande. Cela est utile dans une optique d'analyse effective, mais aussi simplement dans une architecture comme le `evalf()` de Maple, où la procédure numérique qui évalue la racine d'une expression complexe a la charge de déclencher l'évaluation des sous-expressions *en ajustant la précision des calculs* de sorte que le résultat global soit correct. Pour pouvoir utiliser le *bit burst* dans ce contexte, il suffirait d'être capable de calculer automatiquement à quelle précision il est nécessaire de connaître le point d'évaluation d'une fonction holonome pour garantir une certaine précision sur le résultat. Des bornes données par van der Hoeven [**vdH99**, §4.2 et 4.3] apportent une première réponse à cette question.

CHAPITRE 5

BORNES POUR LES FONCTIONS HOLONOMES

Venons-en pour terminer au contrôle de la précision des développements en série. Soit f une fonction holonome, et soit ρ le rayon de convergence de sa série de Taylor en 0. On a vu au chapitre précédent (équation (30)) qu'un nombre de termes de l'ordre de $p\log_{10}\left(\rho/|z|\right)$ est suffisant pour calculer $f(z)$ à précision 10^{-p}. En général (sauf téléscopage ou autre simplification « exceptionnelle »), cette borne est du bon ordre de grandeur, c'est-à-dire que le nombre de termes correspondant est effectivement nécessaire.

Par ailleurs, d'après la formule de Cauchy, pour tout $r < \rho$,

$$|f_n| \leqslant \frac{\max_{|z|=r}|f(z)|}{r^n}.$$

J. van der Hoeven a donné plusieurs algorithmes [**vdH99, vdH01**] qui prennent en entrée une équation différentielle et calculent pour ses solutions des bornes de la forme ci-dessus. Le rayon r peut être pris arbitrairement proche de ρ, cependant, à r fixé et $n \to \infty$, la borne s'éloigne exponentiellement de (30).

L'objet de cette partie est d'expliquer comment calculer automatiquement des bornes sur les développements en série de fonctions holonomes en respectant l'ordre de croissance exponentiel $1/\rho^n$ des coefficients. La méthode utilisée est un raffinement de celle de [**vdH01**, §2.4] et [**vdH03**, §3.5]. Pour un rayon de convergence fini, le nombre de termes à sommer pour évaluer f à 10^{-p} près reste $O(p)$, mais le coefficient du terme linéaire est (génériquement) optimal. Si f a un rayon de convergence infini, le résultat est plus spectaculaire : la décroissance asymptotique des coefficients est plus rapide que toute exponentielle, donc en $O(1/n!^{\kappa})$ pour un certain κ d'après le théorème 3.10, et le nombre de termes à sommer tombe à $O(p/\log p)$.

L'intérêt de ces bornes est multiple. Pour l'évaluation numérique de fonctions holonomes, l'usage de bornes fines peut réduire sensiblement le temps de calcul. De façon peut-être plus importante, les bornes obtenues sont simples. En particulier, elles évitent de choisir le rayon r en fonction du point d'évaluation z, et font disparaître tout problème de compromis entre domaine de validité de la borne et absorption des facteurs sous-exponentiels. Enfin, des bornes sur les coefficients et les valeurs de fonctions spéciales peuvent être utiles dans divers autres contextes (méthodes d'intégration numérique, par exemple), voire représentent un résultat « compréhensible », intéressant pour lui-même (combinatoire), et qu'il convient de renvoyer à l'utilisateur.

Concernant ce dernier point, on obtient en corollaire des résultats sur les équations différentielles un moyen de borner automatiquement certaines suites vérifiant des récurrences linéaires à coefficients polynomiaux. Par exemple, il est possible de déterminer automatiquement que la suite d'équation

$$\{u(n+2) = u(n+1) + u(n), u(0) = 3, u(1) = 2\}$$

est bornée par $5\varphi^n$ (où φ est le nombre d'or), ou encore que celle vérifiant

$$\{(n+1)u(n+1) = (2n+3)u(n), u(0) = 1\}$$

est bornée par $(n+1)2^n$.

5.1. Séries majorantes

La méthode des séries majorantes, ou méthode de Cauchy-Kovalevskaïa, permet de démontrer la convergence de solutions formelles d'équations différentielles (voir la preuve du théorème 3.1) ou aux dérivées partielles et de donner des bornes sur leurs valeurs ou leurs restes. On se limite ici au cas des équations différentielles linéaires. L'idée est d'associer à une équation différentielle $y^{(r)} = L \cdot y$ une *équation majorante* $y^{(r)} = \hat{L} \cdot y$ telle que L majore $\hat{L}$ au sens de la définition suivante. Pour des choix convenables de conditions initiales, les solutions de l'équation majorante majorent alors celles de l'équation de départ.

DÉFINITION 5.1. — Soient $f \in \mathbb{C}[[z]]$ et $g \in \mathbb{R}_+[[z]]$. On dit que g majore f, et l'on note $f \leqslant g$, lorsque pour tout $n \in \mathbb{N}$, $|f_n| \leqslant g_n$. On étend la définition (et la notation) au cas où F, resp. G sont des vecteurs, des matrices, ou des opérateurs différentiels à coefficients dans $\mathbb{C}[[z]]$, resp. $\mathbb{R}_+[[z]]$, en convenant que G majore F si la définition précédente est satisfaite coefficient par coefficient. Remarquons que ceci a un sens pour des matrices de constantes.

Si $f \leqslant g$, le rayon de convergence ρ de f est supérieur ou égal à celui de g. En particulier, si $f \leqslant 1/(1-\alpha z)$, on a $\rho \geqslant 1/\alpha$. Inversement, si $1/\alpha < \rho$, alors d'après la formule de Cauchy, $f \leqslant O(1)/(1-\alpha z)$. Observons aussi la compatibilité de la relation $\leqslant$ avec les principales opérations sur les séries, notamment la dérivation et le passage au reste.

PROPOSITION 5.2. — Supposons $f_1 \leqslant g_1$ et $f_2 \leqslant g_2$. Alors

$$(f_1)_{i;j} \leqslant (g_1)_{i;j}, \qquad f_1' \leqslant g_1', \qquad |f_1(z)| \leqslant g_1(|z|),$$
$$f_1 + f_2 \leqslant g_1 + g_2, \qquad f_1 f_2 \leqslant g_1 g_2$$

pour tous $i, j \in \mathbb{N} \cup \{\infty\}$.

L'intérêt des séries majorantes pour l'étude des équations différentielles réside dans la proposition suivante, qui exprime que les majorations des coefficients passent aux solutions.

PROPOSITION 5.3. — Soient $a_0, \dots, a_{r-1} \in \mathbb{C}[[z]]$ et $b_0, \dots, b_{r-1} \in \mathbb{R}_+[[z]]$ telles que $a_0 \leqslant b_0, \dots, a_{r-1} \leqslant b_{r-1}$. Si $f, g \in \mathbb{C}[[z]]$ vérifient les équations

$$f^{(r)} = a_{r-1} f^{(r-1)} + \cdots + a_0 f \tag{43}$$

$$g^{(r)} = b_{r-1} g^{(r-1)} + \cdots + b_0 g \tag{44}$$

et si

$$|f(0)| \leqslant g(0), \quad \dots \quad |f^{(r-1)}(0)| \leqslant g^{(r-1)}(0)$$

alors ($g \in \mathbb{R}_+[[z]]$ et) $f \leqslant g$.

Démonstration. — Il s'agit simplement de constater que les coefficients f_n, g_n des solutions de (43) et (44) vérifient des récurrences analogues, dont l'une « domine » l'autre, de sorte que les inégalités sur les conditions initiales se transmettent par récurrence.

Plus formellement, posons $F = (f, f', \dots, f^{(r-1)})$ et réécrivons (43) comme un système différentiel du premier ordre $F' = A \cdot F$ (où A est une matrice à coefficients dans $\mathbb{C}[[z]]$), soit encore

$$\sum_{n=0}^{\infty} (n+1) F_{n+1} z^n = \sum_{n=0}^{\infty} \sum_{k=0}^{n} A_k F_{n-k}. \tag{45}$$

De même, (44) devient $G' = B \cdot G$. Par identification des coefficients, on a pour tout $n \in \mathbb{N}$:

$$F_{n+1} = \frac{1}{n+1} \sum_{k=0}^{n} A_k \cdot F_{n-k} \qquad \text{et} \qquad G_{n+1} = \frac{1}{n+1} \sum_{k=0}^{n} B_k \cdot G_{n-k}. \tag{46}$$

Par hypothèse, $F_0 \trianglelefteq G_0$. Supposons $F_{;n}(= F_0 + \cdots + F_n z^n) \trianglelefteq G_{;n}$. Observons que si $M, N, \hat{M}, \hat{N}$ sont des matrices de scalaires telles que $M \trianglelefteq \hat{M}$ et $N \trianglelefteq \hat{N}$, alors $M+N \trianglelefteq \hat{M}+\hat{N}$ et $MN \trianglelefteq \hat{M}\hat{N}$. Les relations (46) donnent donc $F_{n+1} \trianglelefteq G_{n+1}$, d'où $F_{;n+1} \trianglelefteq G_{;n+1}$. Par récurrence $F \trianglelefteq G$, et en particulier $f \trianglelefteq g$. □

Pour borner les solutions d'équations différentielles, notre stratégie consistera à calculer une équation majorante admettant une solution aussi simple que possible *de même rayon de convergence*, puis à extraire de cette solution des bornes convenables pour les coefficients et les restes des solutions. La première étape est de borner les coefficients de l'équation, qui sont des fractions rationnelles.

5.2. Majorants pour les fractions rationnelles

Considérons une fraction rationnelle sans pôle en 0

$$r(z) = \frac{N(z)}{D(z)} = \sum_{n=0}^{\infty} r_n z^n \qquad (D(0) \neq 0).$$

On cherche à calculer une série majorante fine pour r. Conformément à l'idée que le comportement des coefficients r_n est déterminé par la nature (ici, valeur et multiplicité) des singularités de r de module minimal, on cherche le majorant sous la forme $M/(1-\alpha z)^m$. Suivant le contexte, α et m pourront être imposés par des contraintes extérieures, ou choisis pour donner l'égalité dans les conditions (47), (48) ci-dessous.

La suite $(r_n)_{n>\deg r}$ satisfait une récurrence linéaire à coefficients constants, de polynôme caractéristique le polynôme réciproque de D. Une telle récurrence admet une base de solutions de la forme $p(n)\,\mu^{-n}$, où p est un polynôme et μ une racine de D de multiplicité au moins $1 + \deg p$. On peut calculer explicitement [1] une expression

$$r_n = \sum_j p_j(n)\, \mu_j^{-n},$$

de (r_n) sur cette base. (Si $\deg r \geqslant 0$, on calcule séparément les valeurs initiales exceptionnelles $r_0, \ldots, r_{\deg r}$.)

Posons

$$r_n = \binom{n+m-1}{m-1} \alpha^n v_n.$$

D'après le paragraphe précédent, dès que

$$\alpha \geqslant \min\Big\{ \frac{1}{|\mu|} \,\Big|\, D(\mu) = 0 \Big\} \tag{47}$$

$$\text{et} \quad m \geqslant \max\{ \operatorname{ord}_D(\mu) \,|\, D(\mu) = 0 \wedge |\mu| = \alpha^{-1} \} \tag{48}$$

on a $r_n = O_{n\to\infty}(n^{m-1}\alpha^n)$, autrement dit la suite (v_n) est bornée. Si

$$M \geqslant \max_{n\in\mathbb{N}} |v_n|,$$

on obtient

$$r(z) = \sum_{n=0}^{\infty} \binom{n+m-1}{m-1} \alpha^n v_n z^n \trianglelefteq \sum_{n=0}^{\infty} \binom{n+m-1}{m-1} \alpha^n z^n = \frac{M}{(1-\alpha z)^m}.$$

1. Si $r \in \mathbb{K}(z)$, tous les calculs ont lieu dans le corps de décomposition de D sur $\mathbb{K}$. En pratique, il peut être commode de commencer par faire le changement de variable $\tilde{r}(\alpha z) = r(z)$, qui ramène l'ordre de croissance exponentiel à 1.

Or

$$|v_n| \leqslant \sum_i \left| \frac{p_i(n)}{\binom{n+m-1}{m-1}} \frac{1}{(\alpha\mu_i)^n} \right|.$$

Chacun des termes de cette dernière somme est de la forme $|\varphi(n)\beta^n|$ avec $\varphi \in \mathbb{C}(n)$ (sans pôle entier positif) et $\beta \leqslant 1$. En-dehors des zéros et des pôles de φ, $f : x \mapsto |\varphi(x)\beta^x|$ est une fonction réelle dérivable : il suffit donc de chercher son maximum parmi $f(0)$, $f(\lfloor x \rfloor)$ et $f(\lceil x \rceil)$ pour $x \in \mathbb{R}_+$ pôle de φ ou zéro de f' (un zéro de φ correspond nécessairement à un minimum).

REMARQUE. — Esquissons une variante du calcul précédent, que l'on peut préférer suivant les outils disponibles pour l'implémentation. Avec les mêmes notations que ci-dessus, on écrit la récurrence sur (r_n) sous la forme matricielle

$${}^t[r_n, \dots, r_{n+\deg D-1}] = P^{-1}T^nP\,{}^t[r_0, \dots, r_{\deg D-1}] = O(\|T\|^n)$$

où T est en forme de Jordan (la réduction de Jordan remplace le calcul d'une formule pour r_n). Soient δ, ν les parties diagonale, resp. nilpotente de T (elles commutent). Pour un certain $m' \geqslant m$:

$$T^n = (\delta+\nu)^n = \delta^n \sum_{j=0}^{m'} \binom{n}{j} (\delta^{-1}\nu)^j = O(\|\delta\|^n n^{m'}), \tag{49}$$

et la constante implicite dans le $O(\cdot)$ est calculable.

Cependant, si la racine de multiplicité maximale de D n'est pas une racine dominante, $m' > m$, et la borne donnée par (49) est moins bonne que celle obtenue précédemment. On peut retrouver le « bon » degré en séparant les blocs de Jordan de T et en absorbant les surcoûts polynomiaux associés aux valeurs propres non dominantes de multiplicité élevée dans la partie exponentielle, de façon analogue à ce que l'on a fait plus haut.

5.3. Singularités à distance finie

Passons aux développements en série de fonctions holonomes. Commençons par ce qui est presque le cas général, à savoir celui d'une équation

$$a_r y^{(r)} + a_{r-1} y^{(r-1)} + \cdots + a_0 y = 0, \quad a_0, \dots, a_r \in \mathbb{C}[z] \tag{50}$$

avec un point singulier, éventuellement irrégulier, à distance finie. Nous verrons plus loin comment la méthode s'adapte si la seule singularité est à l'infini, et comment on peut tirer bénéfice de la présence de points singuliers réguliers.

DÉFINITION 5.4. — On appelle singularité dominante de l'équation (50) le pôle de multiplicité maximale parmi ceux de module minimal des fractions rationnelles

$$\frac{a_{r-1}}{a_r}, \dots, \frac{a_0}{a_r}.$$

Si celles-ci sont des polynômes, on dira que la singularité dominante de (50) est à l'infini. Si a_r est premier avec $\operatorname{pgcd}(a_{r-1}, \dots, a_0)$, ce que nous supposerons désormais, la singularité dominante de (50) n'est autre que celle de a_r.

La singularité dominante est donc celle qui détermine les paramètres α et m des bornes que l'algorithme de la section précédente est capable de calculer sur les coefficients. Supposons-la finie et non nulle, et soit $1/\alpha$ son module. On s'attend, pour les coefficients des solutions, à des développements asymptotiques de la forme donnée par le théorème 3.10, de partie dominante α^n.

5.3.1. Équation majorante. — Commençons par réécrire (50) sous la forme

$$y^{(r)} = -\frac{a_{r-1}}{a_r} y^{(r-1)} - \cdots - \frac{a_0}{a_r} y, \tag{51}$$

et calculons par la méthode de la section précédente un majorant pour chacun des coefficients obtenus :

$$\frac{a_j}{a_r} \preccurlyeq \frac{M_j}{(1-\alpha z)^{m_j}}, \qquad j = 0, \ldots, r-1.$$

La majoration reste valable si l'on augmente M_j ou m_j. Il existe donc $M > 0$ et $N \in \mathbb{N}$ tels que

$$y^{(r)} = \frac{M}{(1-\alpha z)^N} \sum_{j=0}^{r-1} \binom{r-1}{j} N^{\uparrow(r-j)} \Big(\frac{\alpha}{1-\alpha z}\Big)^{r-j} y^{(j)} \tag{52}$$

soit une équation majorante de (51). Or (52) admet la solution simple [2]

$$g(z) = \exp \frac{M}{(1-\alpha z)^N}. \tag{53}$$

D'après la proposition 5.3, toute solution f de (51) vérifie donc

$$f(z) \preccurlyeq \max_{j=0}^{r-1} \left| \frac{f^{(j)}(0)}{g^{(j)}(0)} \right| \exp \frac{M}{(1-\alpha z)^N}. \tag{54}$$

5.3.2. Borne sur les coefficients. — À partir de (54), on se propose de donner des bornes sur les coefficients des solutions de (50). Conformément à l'objectif initial, on recherche des bornes de la forme

$$|f_n| \leqslant \alpha^n \varphi(n)$$

où φ est sous-exponentielle, c'est-à-dire

$$\forall \varepsilon > 0, \quad \varphi(n) = o((1+\varepsilon)^n).$$

Comparé aux majorants plus grossiers $(1-\alpha' z)^{-N}$, un inconvénient de (53) est que son coefficient général n'admet pas d'expression simple. La série majorante suffit pour certaines opérations ; mais, notamment pour être en mesure de déterminer efficacement un rang n à partir duquel $|f_n|$ est inférieur à une constante imposée, il est souhaitable de disposer d'une borne plus explicite.

PROPOSITION 5.5. — Soit g définie par (53). Pour $n \geqslant MN$,

$$g_n \leqslant \alpha^n \varphi(n), \quad \text{où} \quad \varphi(n) = \left(1 - \Big(\frac{MN}{n}\Big)^{\frac{1}{N+1}}\right)^{-n} \exp \frac{M}{\big(\frac{MN}{n}\big)^{\frac{1}{N+1}}},$$

et $\varphi(n)$ est une fonction sous-exponentielle.

Démonstration. — On utilise la méthode du col, dans sa « forme dégénérée » applicable aux séries à coefficients positifs [**FS07**, prop. 4.1, voir aussi chap. 6]. Comme $g \in \mathbb{R}_+[[z]]$, pour tout $r \in]0, \alpha[$, on a $g_n r^n \leqslant g_0 + g_1 r + \ldots$, et donc

$$g_n \leqslant \frac{g(r)}{r^n}. \tag{55}$$

2. Aucun miracle : elle est construite pour, en augmentant artificiellement l'ordre d'une équation du premier ordre annulant g. Précisément, (52) s'écrit encore $L \cdot y = 0$ pour

$$L = \big[\big((1-\alpha z)\partial + (N+r-1)\alpha\big) \ldots \big((1-\alpha z)\partial + N\alpha\big)\big]\big((1-\alpha z)^{p+1}\partial + Nr\alpha\big) \in \mathbb{C}[z, \partial].$$

Estimons en fonction de n la valeur de r qui minimise le second membre. L'équation

$$\frac{\mathrm{d}}{\mathrm{d}r}\log\frac{g(r)}{r^n} = \frac{\alpha MN}{(1-\alpha r)^{N+1}} - \frac{n}{r} = 0,$$

soit $\alpha MNr = n(1-\alpha r)^{N+1}$, admet la solution approchée [3]

$$r_n = \frac{1}{\alpha}\left(1 - \left(\frac{MN}{n}\right)^{\frac{1}{N+1}}\right). \tag{56}$$

Pour $n \geqslant MN$, on a $r_n < 1/\alpha$, et donc $g_n \leqslant g(r_n)\, r_n^{-n}$ est la borne annoncée. Enfin,

$$\varphi(n) = \exp\bigl(M^{\frac{1}{N+1}} N^{-\frac{N}{N+1}} n^{\frac{N}{N+1}} - n\log\bigl(1-(MN)^{\frac{1}{N+1}} n^{-\frac{1}{N+1}}\bigr)\bigr) = \exp O(n^{\frac{N}{N+1}}),$$

est bien sous-exponentielle. □

REMARQUE. — En utilisant l'inégalité

$$\forall x \in]0;1[, \quad -\log(1-x) \leqslant \frac{x}{(1-x)},$$

on peut remplacer la dernière estimation de φ par

$$\varphi(n) \leqslant \exp\bigl(3(MN)^{\frac{1}{N+1}} n^{\frac{N}{N+1}}\bigr)$$

pour $n \geqslant 2^{N+1}MN$.

5.3.3. Borne sur les restes. — Subsiste la question de borner les *restes* des séries solutions de (50). Dans le cadre du prolongement analytique numérique, on souhaite également borner les restes de leurs dérivées.

Soient $f \leqslant g$ deux séries. Sous réserve que $r_n > |z|$, on a

$$|f_{n;}^{(j)}(z)| \leqslant g_{n;}^{(j)}(|z|) \leqslant \sum_{k=n}^{\infty} \frac{g^{(j)}(r_n)}{r_n^k}|z|^k = g^{(j)}(r_n)\left(\frac{|z|}{r_n}\right)^n \frac{1}{1-\frac{|z|}{r_n}}. \tag{57}$$

Cette formule est générale, mais le choix de r_n dépend du majorant g. Pour celui obtenu ci-dessus (équation (53)), la valeur de r_n donnée par (53) convient encore (les dérivées $g^{(j)}$ ont le même comportement asymptotique que g, à des facteurs polynomiaux près), et la condition $r_n > |z|$ se traduit par

$$n \geqslant \frac{MN}{(1-\alpha|z|)^{N+1}} =: n_0.$$

Pour obtenir $|f_{;n}(z) - f(z)| \leqslant \varepsilon$, il suffit donc [4] de prendre $n \geqslant \min(n_0, n_1)$, où n_0 est défini ci-dessus et n_1 est un entier quelconque rendant le membre droit de (57) inférieur à ε.

3. Dans un programme qui utiliserait cette méthode pour calculer des bornes numériques, on pourrait obtenir des résultats légèrement meilleurs en prenant pour r_n une solution numérique de l'équation.

4. Naturellement, les restes des dérivées des majorants sont des fonctions holonomes. Il serait *a priori* possible de les évaluer à faible précision éventuellement par l'algorithme objet de ce rapport lui-même (si nécessaire avec d'autres bornes plus grossières), pour estimer le nombre de termes à sommer. Cependant (outre qu'il n'est pas évident d'écrire sans erreur un programme qui s'appelle récursivement pour contrôler la précision des calculs qu'il effectue) il semble délicat d'obtenir un contrôle correct de l'ordre et des singularités des équations qu'ils vérifient.

5.4. Fonctions entières (esquisse)

Les bornes de la section précédente s'adaptent au cas où la seule singularité de l'équation est à l'infini. Celle-ci se met alors sous la forme

$$y^{(r)} = -a_{r-1}y^{(r-1)} - \cdots - a_0 y, \tag{58}$$

où les a_j sont des polynômes. Pour des valeurs convenables de M et N,

$$y^{(r)} = M\big((1+z+\cdots+z^N)\,y^{(r-1)} + \cdots + (1+z+\cdots+z^{N-r+1})\,y\big), \tag{59}$$

est une équation majorante de (58). Les séries

$$g(z) = A\,\exp\Big(M\Big(z+\frac{z^2}{2}+\cdots+\frac{z^{N+1}}{N+1}\Big)\Big) \tag{60}$$

sont solutions de (59) ; en ajustant A, ce sont donc des séries majorantes des solutions.

En posant

$$r_n = \Big(\frac{n}{M}\Big)^{\frac{1}{N+1}}$$

(solution approchée de $M(r^{N+1}+\cdots+r^2+r) = n$), on obtient

$$g_n \geqslant g(r_n) r_n^{-n} = e^{-\frac{1}{N+1} n\log n + O(n)},$$

d'où, si $f \preccurlyeq g$, la borne sur les restes

$$|f_n^{(j)}(z)| \leqslant g^{(j)}(r_n)\Big(\frac{|z|}{r_n}\Big)\frac{1}{1-\frac{|z|}{r_n}},$$

finie si $n > M|z|^{N+1}$.

Sur les coefficients des solutions, l'échelle de majorants (60) correspond à des décroissances du type

$$[z^n]\exp\Big(z+\frac{z^2}{2}+\cdots+\frac{z^N}{N}\Big) \simeq \frac{1}{n!^{1/N}}$$

(à des facteurs polynomiaux près). Si elle répond au problème de respecter leur ordre de croissance exponentiel, elle « n'attrape » pas pour autant finement tous les comportements possibles des (valeurs absolues des) coefficients d'une série entière holonome. Par exemple, en utilisant la méthode ci-dessus, la fonction Ai d'Airy, solution de $y'' - zy = 0$, est bornée par une fonction du type $Ae^{z+z^2/2}$, alors que

$$[z^n]\,\mathrm{Ai}(z) \simeq \frac{1}{n!^{2/3}}.$$

Il serait intéressant de disposer d'une variante de cette méthode qui donne la « bonne » puissance de $n!$ dans la croissance des coefficients des solutions pour les équations sans singularité à distance finie.

5.5. Points singuliers réguliers

5.5.1. Séries majorantes. — Les bornes de la section 5.3 peuvent être à la fois simplifiées et raffinées quand la singularité dominante de l'équation est un point singulier régulier. Considérons à nouveau

$$a_r y^{(r)} + a_{r-1}y^{(r-1)} + \cdots + a_0 y = 0, \qquad a_0,\dots,a_r \in \mathbb{C}[z]. \tag{61}$$

Soit ζ sa singularité dominante. Supposons que $0 < |\zeta| < \infty$, et que ζ est un point singulier régulier de (61). (Il peut y avoir des points singuliers irréguliers de même module.)

D'après le critère de Fuchs (théorème 3.11), l'équation (61) se met alors sous la forme

$$y^{(r)} = b_{r-1}y^{(r-1)} + \cdots + b_0 y, \tag{62}$$

où ζ est pôle d'ordre au plus $1,\dots,r$ de $b_{r-1},\dots,b_0$ respectivement. En imposant exactement ces ordres dans l'algorithme de la section 5.2 (pourvu de disposer de sa version « fine », qui respecte l'ordre du pôle dominant et non seulement son module), on calcule une équation majorante de (62) sous la forme

$$y^{(r)} = \frac{M_{r-1}}{(1-\alpha z)^1} y^{(r-1)} + \cdots + \frac{M_1}{(1-\alpha z)^{r-1}} y' + \frac{M_0}{(1-\alpha z)^r} y, \quad \text{les } M_j \in \mathbb{R}_+, \tag{63}$$

où l'on a posé comme d'habitude $\alpha = 1/|\zeta|$.

Il s'agit là d'une équation d'Euler, dont la résolution explicite (par le changement de variable $z = e^w$, ou par transformée de Mellin [**Hen77**, p. 338]) est classique. Contentons-nous d'en chercher une solution sous la forme

$$g(z) = A \frac{1}{(1-\alpha z)^\lambda}. \tag{64}$$

En reportant

$$g^{(q)}(z) = A\lambda^{\uparrow q} \alpha^q \frac{1}{(1-\alpha z)^{\lambda+q}},$$

dans (62), on observe que $g(z)$ est solution si et seulement si λ satisfait l'équation algébrique

$$s(\lambda) := \alpha^z \lambda^{\uparrow r} - M_{r-1}\alpha^{r-1}\lambda^{\uparrow(r-1)} - \cdots - M_0 = 0. \tag{65}$$

Clairement $s(0) = -M_0 \leqslant 0$ et $s(\lambda) \xrightarrow[\lambda\to\infty]{} \infty$, ce qui montre l'existence d'un majorant de la forme (64) avec $\lambda \geqslant 0$. Le cas $\lambda = 0$ pose problème : il correspond bien à une série majorante, mais celle-ci a des conditions initiales qui s'annulent, de sorte qu'elle ne peut pas majorer n'importe quelle solution de (63). Ce cas ne peut se produire que si $M_0 = 0$; on l'évite en forçant $M_0 > 0$ même si $b_0 = 0$.

5.5.2. Aparté. — À ce stade, on peut tout de même se demander ce qui se passe si l'équation (65) a plusieurs racines. En effet, une équation de la forme (63) peut être majorée par elle-même, or

$$\frac{1}{(1-\alpha z)^{\lambda'}} \not\ll \frac{1}{(1-\alpha z)^\lambda}$$

si $\lambda' < \lambda$! Rassurons-nous en démontrant que l'équation (65) possède en fait une unique solution réelle strictement positive. Le résultat qui vient à notre secours est une généralisation de la règle des signes de Descartes (qui relie les racines réelles d'un polynôme aux changements de signe de ses coefficients) applicable à d'autres bases de $\mathbb{R}[x]$ que celle des monômes.

THÉORÈME 5.6 (Règle des signes de Runge). — Soient $\xi_1, \xi_2, \dots, \eta_1, \eta_2, \dots \in \mathbb{R}$, et

$$P_k = \prod_{j=1}^{k} (x - \xi_j), \qquad Q_k = \prod_{j=1}^{k} (x - \eta_j) \qquad (j \in [\![0, n]\!]).$$

Soit

$$f = \sum_{j=0}^{n} a_j P_j = \sum_{j=0}^{n} b_j Q_j \in \mathbb{R}[x]$$

de degré n. Si

$$\max_{j=1}^{n} \xi_j \leqslant a < b \leqslant \min_{j=1}^{n} \eta_j$$

alors le nombre de zéros de f dans l'intervalle $]a,b]$, comptés avec multiplicités, est inférieur ou égal à

$$V(a_0,\dots,a_n) - V(b_0,\dots,b_n)$$

où $V(x_0,\dots,x_n)$ désigne le nombre de changement de signes de la suite $x_0,\dots,x_n$ (en ignorant les termes nuls).

On trouvera la preuve dans [**RS02**, théorème 10.7.4, p. 334]. Pour appliquer ce théorème à $s(\lambda)$, adoptons $P_j = \lambda^j$ (c'est-à-dire $\xi_j = -j+1 \leqslant 0$) et $\eta_j = c > 0$ pour tout j. On obtient que le nombre de racines de $s(\lambda)$ dans l'intervalle $]0,c]$ est inférieur ou égal à

$$V(-M_0, \ldots, -M_{r-1}, 1) = 1,$$

d'où le résultat en faisant tendre c vers l'infini.

5.5.3. Coefficients et restes. — On a montré comment calculer, pour les solutions de (61), des majorants de la forme

$$g(z) = \frac{1}{(1-\alpha z)^N}.$$

Contrairement à ce qui se produit dans le cas général, on dispose ici d'une expression explicite simple pour les coefficients de $g(z)$ et de ses dérivées :

$$g_n^{(j)} = [z^n] \frac{\mathrm{d}^j}{\mathrm{d}z^j} \frac{1}{(1-\alpha z)^N} = \prod_{k=0}^{j-1} (N+n+k) \binom{N+n-1}{n} \alpha^{n+j}. \tag{66}$$

De plus, les restes des dérivées de $g(z)$ possèdent une expression close à l'aide de la fonction hypergéométrique

$${}_2F_1\left(\begin{matrix} a & b \\ c \end{matrix} \,\middle|\, z\right) = \sum_{n=0}^{\infty} \frac{a^{\uparrow n} b^{\uparrow n}}{c^{\uparrow n}} \frac{z^n}{n!},$$

à savoir

$$g_{n;}^{(j)}(z) = \left(\prod_{k=0}^{j-1} (N+n+k)\right) \alpha^i (\alpha z)^n \; {}_2F_1\left(\begin{matrix} 1 & N+n+j \\ n+1 \end{matrix} \,\middle|\, z\right).$$

Dans un système qui, comme Maple, dispose déjà d'une routine d'évaluation numérique (même à faible précision) des fonctions ${}_2F_1$, cela fournit une façon de borner les restes plus simple et un peu plus fine que celle exposée plus haut.

REMARQUE. — Et les points singuliers *réguliers* à l'infini ? La structure des solutions au voisinage d'un point singulier régulier (§3.2.2) montre qu'une équation dont *l'unique* point singulier sur la sphère de Riemann est un point singulier régulier à l'infini n'a que des solutions analytiques sur $\mathbb{C}$ avec au plus un pôle à l'infini, c'est-à-dire polynomiales. Leurs degrés possibles sont donnés par l'équation indicielle à l'infini (§3.4).

5.6. Algorithme complet, implémentation

5.6.1. Fonctions holonomes. — La figure 9 résume l'algorithme de calcul de bornes sur les fonctions holonomes développé dans les sections précédentes. La fonction `bound_diffeq` du module Maple que j'ai développé (cf. annexe A) implémente cet algorithme. Les résultats renvoyés ont l'allure suivante :

```
> bound_diffeq(holexprtodiffeq(1/exp(1)*exp(1/(1-z))+1/(1+z/3),
  y(z)),y(z)) ;
```

$$\exp\left(-\frac{3}{-1+z}\right)$$

```
> bound_diffeq({(1+z^2)*diff(y(z),z)-1},y(z));
```

$$-\frac{1}{1-z}$$

```
> bound_diffeq({diff(y(z),z)-y(z)}, y(z)) ;
```

$$\exp(z)$$

PROCÉDURE (Calcul de majorants). — Paramètres : un point z_0 ; une équation différentielle linéaire (E), à coefficients polynomiaux, non singulière en z_0. Résultat : une série majorante pour le développement de Taylor des solutions fondamentales (au sens de la convention fixée §4.2.1) de (E) au voisinage de z_0.

se ramener à $z_0 = 0$ par le changement de variable $z \leftarrow z - z_0$
$\zeta :=$ la singularité dominante de (E) ; $\alpha := |1/\zeta|$
si $\alpha \neq 0$ ▷ singularité à distance finie
  $K := \max_{j=0}^{r-1}\Big(j - r + \operatorname{ord}_\zeta \dfrac{a_j}{a_r}\Big)$
  $M_j^* :=$ un M tel que $a_j/a_r \trianglelefteq M/(1-\alpha z)^{K+r-j}$
  si $K \geqslant 1$ ▷ point singulier irrégulier
    $N := K$
    pour j de 0 à $r-1$
      $M_j = \dfrac{M_j^*}{\binom{r-1}{j-1} N^{\uparrow j} \alpha^j}$
      ▷ Ainsi $\dfrac{a_j}{a_r} \trianglelefteq \binom{r-1}{r-1-j} N^{\uparrow j} \alpha^j \dfrac{M_j}{(1-\alpha z)^{N+r-j}}$
    $M := \max_{j=0}^{r-1} M_j$
    $\mathscr{M} := \exp \dfrac{M}{(1-\alpha z)^N}$
  sinon ▷ point singulier régulier
    $M_j := M_j^*$ pour j de 0 à $r-1$
    si $M_0 = 0$ alors $M_0 \leftarrow 1$
    calculer $N \geqslant \Big($l'unique $\lambda > 0$ tel que $\alpha^z \lambda^{\uparrow r} = \sum_{j=0}^{r-1} M_j \alpha^j \lambda^{\uparrow j}\Big)$
    $\mathscr{M} := \dfrac{1}{(1-\alpha z)^N}$
sinon ▷ pas de singularité à distance finie
  $K := \max(0, \max_{j=0}^{r-1}(r - j + \deg a_j))$
  $M := \dfrac{1}{|a_r|} \max_{j=0}^{r-1} \max_{i=0}^{\infty} |[z^i] a_j|$
  si $M_0 = 0$ alors $M_0 \leftarrow 1$
  $\mathscr{M} := \exp\Big(M \sum_{j=1}^{K+1} \dfrac{z^j}{j}\Big)$
$A := \max_{j=0}^{r-1} \dfrac{1}{[z^j]\mathscr{M}}$
renvoyer $A\mathscr{M}$

FIG. 9. Séries majorantes pour les fonctions holonomes

Le tableau 4 compare les nombres de termes à sommer calculés par diverses méthodes pour l'évaluation de quelques fonctions (plus ou moins) usuelles. Dans tous les cas, la fonction est donnée par son équation différentielle « habituelle » et des conditions initiales en 0, le point d'évaluation se trouve dans le domaine de convergence de son développement en série au voisinage de 0, et l'évaluation est faite en une seule étape, sans prolongement analytique.

		$\arctan\frac{1}{2}$	$\arctan\frac{3}{4}$	$\frac{\cos z}{1-z}, z=\frac{1}{3}$	$(\alpha C+\beta S)(\frac{\pi}{3})$ $(*)$
100 ch.	ce chapitre	336 (104)	808 (104)	216 (103)	212 (104)
	[**vdH01**] (naïf)	583 (178)	1546 (196)	600 (286)	358 (174)
	[**vdH01**] (variante)	578 (177)	1526 (194)	347 (165)	341 (167)
1 000 ch.	ce chapitre	3324 (1004)	8012 (1005)	2106 (1004)	2098 (1006)
	[**vdH01**] (naïf)	5694 (1718)	14990 (1877)	3590 (1712)	3348 (1602)
	[**vdH01**] (variante)	5689 (1716)	14969 (1874)	3336 (1591)	3331 (1595)

$(*)$ Solution de $\left\{(1-z^2)y''(z)-zy'(z)+2(1-2z^2)y(z)=0, y(0)=1, y'(0)=0\right\}$ (forme algébrique de l'équation de Mathieu) prise en $z=1/2$.

		$\exp\frac{z}{1-z^2}, z=\frac{1}{3}$	$\mathrm{erf}\frac{z}{1-z}, z=\frac{1}{3}$	e^{-100}	$\mathrm{Ai}(4i+4)$
100 ch.	ce chapitre	240 (107)	293 (123)	453 (102)	356 (238)
	[**vdH01**] (naïf)	470 (213)	95872	732 (316)	338 (221)
	[**vdH01**] (variante)	338 (152)	344 (146)	732 (316)	477 (357)
1 000 ch.	ce chapitre	2182 (1015)	2418 (1081)	1404 (1003)	1608 (1764)
	[**vdH01**] (naïf)	3460 (1617)	98862	3722 (4232)	3328 (4349)
	[**vdH01**] (variante)	3328 (1555)	3334 (1501)	3722 (4232)	3467

TAB. 4. Nombre de termes à sommer d'après divers algorithmes de majoration. *Entre parenthèses, le nombre de chiffres corrects du résultat.*

Le tableau mentionne à chaque fois le nombre de termes obtenu et, entre parenthèses, le nombre de chiffres décimaux corrects dans le résultat de l'évaluation, le tout pour des précisions demandées de 100 et de 1 000 chiffres. Les quatre premiers exemples possèdent des singularités dominantes régulières, les suivants des points singuliers irréguliers (à distance 1 pour les deux premiers, à l'infini pour les deux autres).

La première ligne correspond à l'algorithme décrit dans ce chapitre, la deuxième, à une implémentation (très) naïve de l'algorithme de [**vdH01**]. Celle-ci utilise la formule de Cauchy sur un disque de rayon $1/\nu$ (avec $\nu > \alpha$) pour borner les fractions rationnelles, d'où certains comportements pathologiques pour des équations dont les coefficients prennent de grandes valeurs tout en ayant des coefficients de Taylor de taille modérée. Les résultats de la troisième ligne proviennent d'une variante du même algorithme qui calcule des bornes sur les *coefficients* des fractions rationnelles, toujours de la forme $A\nu^n$ avec $\nu > \alpha$, mais en utilisant la méthode de la section 5.2. Van der Hoeven a donné [**vdH99**] un autre algorithme, que je n'ai pas implémenté [5], pour borner les coefficients des séries holonomes.

Veiller à borner les coefficients des fractions rationnelles *via* la relation de récurrence qu'ils vérifient plutôt qu'en utilisant la formule de Cauchy suffit comme on le voit à éliminer les cas pathologiques ; en revanche, l'algorithme issu de [**vdH01**] calcule systématiquement (pour une singularité à distance finie) un nombre à peu près linéaire de chiffres superflus. Il est vraisemblablement possible de mitiger, voire d'éliminer cet effet en choisissant soigneusement le rayon du disque sur lequel on fait les calculs de bornes, éventuellement en fonction de l'équation ou de la précision demandée. Par ailleurs, concernant l'algorithme de ce chapitre, on observe le fait mentionné §5.4 que les majorants « fins » ne donnent pas la toujours bonne puissance de $n!$ pour des fonctions entières comme la fonction d'Airy,

5.6.2. Cas des suites. — En corollaire immédiat de ce qui précède, on obtient un algorithme pour borner les solutions des récurrences linéaires à coefficients polynomiaux dont

5. Il est un peu plus compliqué que celui de [**vdH01**] et renvoie ses résultats sous une forme moins commode tant pour le contrôle de leur qualité que pour le passage aux restes des dérivées. Par ailleurs, il me semble plus difficile à adapter à des situations plus générales comme celles de [**vdH03, vdH05**].

l'équation différentielle associée est non singulière en zéro. Il s'agit simplement de faire la traduction (à l'aide de `gfun:-diffeqtorec`) puis d'appeler l'algorithme pour les fonctions holonomes.

À nouveau, voici quelques exemples. Comme on le voit, j'ai choisi, en l'absence d'expression simple pour les coefficients de la série majorante, de renvoyer tout de même le majorant le plus fin disponible, comme coefficient extrait d'une série génératrice. Ce choix est discutable, une autre possibilité aurait été de donner la borne plus explicite fournie par la méthode du col.

```
> bound_rec({u(n+1)=u(n),u(0)=3},u(n));
```

$$3$$

```
> bound_rec({u(n+1)=u(n)+1, u(0)=1},u(n));
```

$$n+1$$

```
> bound_rec({u(n+2)=u(n+1)+u(n),u(0)=3,u(1)=2},u(n));
```

$$5(\mathrm{RootOf}(\_Z^2+\_Z-1,0.6180339887)+1)^n$$

```
> bound_rec({(n+2)*u(n+2)=u(n), u(0)=1, u(1)=1}, u(n));
```

$$2\,\mathrm{Coeftayl}\left(\exp\left(\frac{z(2+z)}{2}\right)\right)$$

```
> value(subs(n=10,%));
```

$$\frac{1187}{226800}$$

Pour ces quelques suites, l'approche très naïve adoptée marche plutôt bien. Elle a pourtant de nombreuses limitations, qui font qu'elle n'est, en l'état, guère plus qu'une illustration des bornes sur les équations différentielles :

```
> bound_rec({u(n+1)=u(n)+1, u(0)=-1},u(n));
```

$$2^n$$

```
> bound_rec({u(n+1)=(n+1)*u(n), u(0)=1},u(n));
  Error, (in fail_if_singular) equation y(z)+(-1+3*z)*diff(y(z),z)
  +z^2*diff(diff(y(z),z),z) is singular in 0
```

CONCLUSION ET PERSPECTIVES

Ce stage a abouti à une implémentation générique, apparemment la première, de l'algorithme d'évaluation et de prolongement analytique numérique des fonctions holonomes par scindage binaire. Cette implémentation a un comportement relativement satisfaisant en pratique, puisqu'elle est capable de traiter des problèmes au-delà des possibilités natives de Maple, soit en raison de leur taille, soit parce que les fonctionnalités nécessaires n'étaient pas disponibles. Bien qu'à un stade expérimental, elle a donc une utilité immédiate qui dépasse la seule observation du comportement des algorithmes, et peut espérer des utilisateurs « extérieurs » à assez court terme.

Parallèlement à ce travail de développement, et en interaction avec lui, j'ai tenté d'éclaircir les points qui me semblaient encore imparfaitement compris, susceptibles de petites améliorations guidées par la pratique, ou simplement intrigants des algorithmes en jeu. Cela a débouché principalement sur le travail réalisé avec Bruno sur les séries majorantes pour les fonctions holonomes exposé au chapitre 5, qui montre que l'on peut estimer automatiquement avec une certaine finesse le nombre de termes à prendre en compte quand on manipule des fonctions holonomes par leurs développements en série, en fonction de la précision visée.

Au-delà du modeste travail de recherche effectué, et de façon sans doute plus importante, ces cinq et quelques mois m'ont beaucoup appris. J'ai découvert toute une gamme de problèmes et de techniques plus ou moins classiques du calcul formel et du calcul numérique à grande précision, et pris mes premiers repères dans la littérature du calcul formel. J'ai aussi eu la chance d'assister à une école de jeunes chercheurs (EJCIM, Nancy, du 19 au 23 mars), une conférence internationale (ISSAC, Waterloo, Ontario, du 29 juillet au 1er août), un cours d'*Analyse effective* donné par J. van der Hoeven au master de mathématiques de l'université Paris XI, et un peu plus d'une vingtaine d'exposés divers (séminaires, thèses...)

Le travail commencé pendant ce stage ouvre un certain nombre de perspectives. Les plus immédiates concernent l'implémentation : le code actuel demande à être nettoyé et testé systématiquement, et certaines fonctionnalités commodes pour l'utilisateur manquent encore, alors que les briques de base pour les ajouter sont disponibles. Un certain nombre d'idées pour améliorer son efficacité pratique (notamment celles présentées au chapitre 2), essayées au début puis abandonnées provisoirement en raison de problèmes avec les routines d'algèbre linéaire de Maple, mériteraient d'être évaluées à nouveau maintenant que le code n'est plus désespérément inefficace.

Plus largement, une « prochaine étape » naturelle serait d'étendre tout ce qui peut l'être, à commencer par les majorants et l'implémentation, au prolongement analytique « à travers » les points singuliers réguliers [**vdH01**]. Les principales motivations sont les suivantes : premièrement, concernant l'évaluation des fonctions spéciales, de nombreuses fonctions classiques, par exemple les fonctions de Hankel, sont définies non par des conditions initiales en un point ordinaire, mais par leur asymptotique en un point singulier régulier. Deuxièmement,

les « valeurs » des fonctions holonomes en des points singuliers réguliers (c'est-à-dire leurs expressions dans certaines bases de développements asymptotiques) apparaissent en combinatoire et en analyse d'algorithmes, comme constantes dans les équivalents asymptotiques de suites obtenus par analyse de singularités, et on ne sait généralement pas les calculer autrement que numériquement.

Beaucoup d'autres pistes mériteraient d'être explorées à plus long terme. Il serait intéressant de comprendre comment diminuer dans certains cas la taille (ordre et hauteur) des récurrences que l'on calcule (penser par exemple aux fonctions paires), ou encore de voir si l'on peut parfois prédire et éliminer un « gros » facteur commun entre le numérateur et le dénominateur calculés [**CHT$^+$07**]. Le travail fait ici devrait pouvoir être adapté à la précision relative, et par là à l'évaluation en virgule flottante. Il serait aussi souhaitable d'arriver à précalculer certaines données utiles à partir de la seule fonction à évaluer, et à détecter automatiquement, pour chaque fonction, les seuils à partir desquels les algorithmes pour la grande précision sont pertinents. Tous ces aspects pourraient relancer l'intérêt de la génération de code, mise de côté pendant ce stage au profit d'une implémentation générique.

Remerciements

Je ne voudrais pas conclure ce rapport sans remercier vivement mon directeur de stage Bruno Salvy, et avec lui Alin Bostan et Frédéric Chyzak, les deux autres piliers de la composante calcul formel du projet Algo. Merci aussi, pour de nombreuses discussions instructives et explications lumineuses, à Philippe Dumas, Philippe Flajolet, Frédéric Meunier, Rémy Oudompheng, Adrien Poteaux, Paul Zimmermann, et tous ceux que j'oublie. Merci encore à tous les membres d'Algo et des projets amis, pour l'ambiance agréable qui règne en salle café. Enfin, merci à Anne de m'avoir supporté pendant nos « vacances » communes consacrées à rédiger ce rapport !

Notes

L'historique du scindage binaire de l'introduction est tiré de [**vdH07, BCS05, Ber04**]. La présentation informelle de l'algorithme (§1.2) doit beaucoup à [**BCS07**] ainsi qu'au cours 2-22 du MPRI d'où les notes en question sont adaptées. Les rappels sur l'holonomie du chapitre 5 proviennent principalement de [**Sta80, FS07, BCS07**]. La démonstration du théorème de Cauchy §3.1 est une adaptation de [**Hen77**, théorème 9.3a], et la présentation de l'équation indicielle §3.4 est inspirée de [**CC87**, section 3].

ANNEXES

ANNEXE A

UTILISATION DU MODULE NUMGFUN

Cette annexe est la version française du manuel d'utilisation du module Maple `NumGfun`, version 0.1. Celui-ci regroupe la partie la « moins expérimentale » du code que j'ai écrit pendant mon stage.

Prolongement analytique numérique

ac_eval(*deq, y(z), path, precision*) — Évalue numériquement le prolongement analytique de la solution de l'équation *deq* le long du chemin *path*. Le point de départ doit être 0.

deq	équation différentielle linéaire à coefficients polynomiaux, non singulière en 0, avec conditions initiales données sur les premières dérivées en 0
y(z)	variables
path	liste d'éléments de $\mathbb{Q}[i]$ commençant par 0, p.ex. $[0, 1, 1+I]$
precision	précision absolue (nombre de chiffres décimaux corrects après la virgule) du résultat

transition_matrix(*deq, y(z), path, precision*) — Comme **ac_eval**, mais calcule la matrice de passage complète entre la base canonique de solutions en zéro (au sens défini §4.2.1) et celle à l'extrémité du chemin *path*.

deq	équation différentielle linéaire à coefficients polynomiaux non singulière en 0 (d'éventuelles conditions intiales sont ignorées)
y(z), path, precision	comme ci-dessus

diffeqtoproc(*deq, y(z)*) — Renvoie une procédure qui évalue la fonction définie par *deq* en un point à coordonnées rationnelles situé dans le disque de convergence de sa série de Taylor en 0.

deq	équation différentielle linéaire à coefficients polynomiaux non singulière en 0, avec conditions initiales
y(z)	variables

local_monodromy(*deq, y(z), sing, start, precision*) — Calcule la matrice de monodromie locale autour de *sing* de la fibre au-dessus de *start*. *Cette fonction est utilisée dans certains exemples de ce document, mais n'est pas prête à un usage général.*

deq	équation différentielle linéaire à coefficients polynomiaux non singulière en 0 (d'éventuelles conditions intiales sont ignorées)
start	point-base, plus près de *sing* que de toute (autre) singularité
sing	point au voisinage duquel on calcule la monodromie
y(z), precision	comme ci-dessus

N-ième terme de suites récurrentes

nth_term(*rec, u(n), N [, ringname]*) — Calcule le N-ième terme d'une suite récurrente.

rec	récurrence linéaire à coefficients polynomiaux, avec conditions initiales
u(n)	variables
N	entier > 0
ringname	(optionnel) si présent et égal à `ndmatrix`, calcule $\sum_{j=0}^{n-1} u_j$ au lieu de u_n au moyen de l'écriture « creuse » de la matrice de la récurrence (§2.3)

fnth_term(*rec, u(n), N, precision [, ringname]*) — De même, mais renvoie un résultat flottant tronqué à *precision* chiffres.

precision	entier $\geqslant 1$
rec, u(n), N, ringname	comme ci-dessus

rectoproc_binsplit(*rec, u(n)*) — Renvoie une procédure calculant la suite définie par *rec*.

rec, u(n)	comme ci-dessus

Bornes

bound_diffeq(*deq,y(z)*) — Calcule une série majorante valable pour toutes les solutions fondamentales (au sens défini §4.2.1) d'une équation différentielle.

deq	équation différentielle (les conditions intiales sont ignorées)
y(z)	variables

bound_rec(*rec, u(n)*) — Calcule une borne simple sur la solution de la récurrence *rec*.

rec	récurrence linéaire à coefficients polynomiaux (avec conditions initiales)
u(n)	variables

bound_ratpoly(*rat,z*) — Calcule une série majorante pour une fraction rationnelle. Asymptotiquement, la borne correspondante sur les coefficients est exacte à des facteurs logarithmiques près.

rat	fraction rationnelle en z
z	variable

Divers

waksman_product(r) — Renvoie une procédure qui multiplie deux matrices $r \times r$ par l'algorithme de Waksman

r	entier $\geqslant 1$

ANNEXE B

COMPTES-RENDUS DE SÉMINAIRES

Le projet Algo organise régulièrement un séminaire sur l'analyse d'algorithmes et divers sujets apparentés, en particulier le calcul formel. Les actes de ce séminaire, formés de comptes-rendus de chaque séance par un auditeur, sont publiés sur le web (`http://algo.inria.fr/seminars/`) et sous forme de rapports de recherche Inria.

Cette annexe reprend les deux comptes-rendus que j'ai écrits pendant ces quelques mois. Ils portent sur des sujets connexes à mon sujet de stage, et complètent utilement certaines parties de ce rapport.

Le premier concerne le calcul de la monodromie des équations algébriques, suivant un exposé d'Adrien Poteaux. On peut voir la méthode qu'il présente comme concurrente de celle consistant à déduire de l'équation algébrique une équation différentielle (proposition 3.16), dont on calcule numériquement la monodromie par les méthodes de ce rapport pour en déduire celle (exacte) de la courbe algébrique [**CC90**]. Plusieurs problèmes, comme le contrôle de la précision et le choix des chemins de prolongement analytique, sont similaires dans les deux cas.

Le second exposé, par Sylvain Chevillard, porte sur la production semi-automatique de formules de calcul exact à petite précision (principalement à l'aide de flottants machine) pour les fonctions réelles. On attendrait d'un générateur de procédures numériques complet qu'il produise du code capable de s'adapter à la précision demandée, en utilisant toute une gamme de méthodes allant de celles extrêmement rapides à petite précision fournies par les idées de l'exposé à celles asymptotiquement efficaces vues dans ce rapport.

Algorithms Seminar 2006–2007,
F. Chyzak (ed.), INRIA

Available online at the URL
http://algo.inria.fr/seminars/.

Computing Monodromy Groups Defined by Plane Algebraic Curves

Adrien Poteaux
Xlim-Dmi, umr Cnrs 6172, Université de Limoges

April 23, 2007

Summary by Marc Mezzarobba

Abstract

This work deals with the numerical and symbolic-numerical computation of the monodromy of complex algebraic curves, with special emphasis on correct error control in presence of close singularities. The speaker's approach uses Puiseux expansions above singularities and analytic continuation along the edges of a minimal Euclidean spanning tree connecting them. This work is a first step toward an effective version of the Abel-Jacobi theorem relating an algebraic curve to its Jacobian variety.

1. Motivation and Problem Statement

1.1. The Abel-Jacobi Theorem.

An irreducible polynomial

$$F = Y^d + a_{d-1}(X)Y^{d-1} + \cdots + a_0(X) \in \mathbb{K}[X, Y] \qquad (d \geq 1) \tag{1}$$

over some subfield $\mathbb{K}$ of the complex numbers defines a plane affine algebraic curve

$$\mathcal{C} = \{(x, y) \in \mathbb{C}^2 : F(x, y) = 0\},$$

along with a ramified covering $\pi : \mathcal{C} \to \mathbb{C}$ of the complex plane and a compact Riemann surface $\hat{\mathcal{C}}$ canonically associated[1] to the curve $\mathcal{C}$. We seek to compute the *monodromy group* defined by this curve. The speaker's long-term goal, and the origin of his interest in this problem, is an effective version of the Abel-Jacobi theorem. Applications include the algebraic case of the Risch algorithm for indefinite integration, differential Galois group computations, and the study of some solution families of partial differential equations arising in physics, such as Korteweg-de Vries equations.

One concrete way to introduce this theorem is the following: a compact Riemann surface such as $\hat{\mathcal{C}}$ admits meromorphic functions. These necessarily have as many zeros as poles, when counted with multiplicities. Conversely, one may ask whether there exists a meromorphic function on $\mathcal{C}$ with prescribed zeros and poles satisfying the previous condition. For example, if the curve has genus zero, its set of meromorphic functions is $\mathbb{C}(X)$, so the answer is always positive.

Turning to the general case, recall that a divisor of $\mathcal{C}$ is a formal sum $\sum_P n_P P, n_P \in \mathbb{Z}$ of points of the curve. The degree of a divisor is the sum of its coefficients. A divisor that is the sum of the zeros of a meromorphic function (with multiplicities, and counting poles as zeros with negative multiplicity) is called a function divisor. Function divisors form a subgroup, denoted $\mathrm{Prin}(\mathcal{C})$, of the (Abelian) group $\mathrm{Div}^0(\mathcal{C})$ of degree zero divisors. So the former question amounts to decide whether a given divisor is a function divisor.

[1] Denote by S the set of critical points of π, as defined in §1.2. Then $\mathcal{C}' = \mathcal{C} \setminus (S \times \mathbb{C})$ is a (connected) finitely punctured compact Riemann surface and $\pi_{|\mathcal{C}'}$ is an unramified covering of $\mathbb{C} \setminus S$. "Stuffing the holes" (including that at infinity), we obtain $\hat{\mathcal{C}}$ and a map $\hat{\pi} : \hat{\mathcal{C}} \to \mathbb{P}^1\mathbb{C}$ extending $\pi_{|\mathcal{C}'}$.

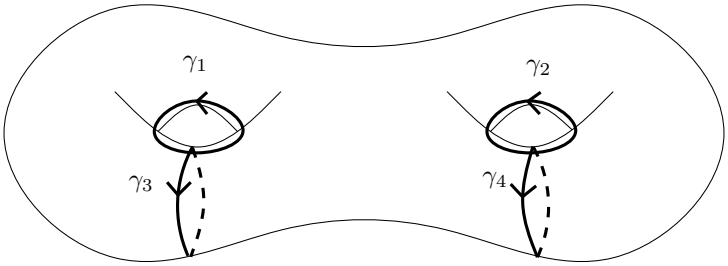


FIGURE 1. The canonical basis of homology, for a curve of genus 2.

As a complex manifold, $\hat{\mathcal{C}}$ admits holomorphic differential one-forms (a.k.a. differentials of the first kind or differentials without poles). These can be integrated along cycles, and the value of such an integral depends only on the homology class of the cycle and the cohomology class of the differential. Let $(\omega_1, \ldots, \omega_g)$ be a basis of the cohomology space $H^1(\mathcal{C})$ ("a basis of the holomorphic differentials") and let $(\gamma_1, \ldots, \gamma_{2g})$ be the canonical basis of $H_1(\mathcal{C})$, made of (the classes of) the loops depicted on Figure 1. Integrals of ($\mathbb{Z}$-combinations of) the ω_i along ($\mathbb{Z}$-combinations of) the γ_j form a lattice $\Gamma \subset \mathbb{C}$. Now we are in position to state the theorem.

Theorem 1. *The Abel-Jacobi map*

$$\begin{array}{rcl} \mathrm{Div}^0(\mathcal{C})/\mathrm{Prin}(\mathcal{C}) & \to & \mathrm{Jac}(\mathcal{C}) = \mathbb{C}^g/\Gamma \\ P & \mapsto & \left(\int_O^P \omega_1, \ldots, \int_O^P \omega_g\right) \end{array}$$

is a group isomorphism.

To decide whether a divisor is a function divisor using this theorem, we have to compute the period lattice Γ above. Using an algorithm due to C. and M. Tretkoff, this can be reduced to computing the monodromy of the curve [5].

1.2. **Monodromy.** Above all but finitely many points a of the x-plane, the fiber $\pi^{-1}(a)$ has cardinality d. In that case, a is called a *regular point*. Points that are not regular (that is, such that the univariate polynomial $F(a, Y)$ has multiple roots) are said to be *critical*. If a is a regular point, the implicit function theorem states that there exist d analytic functions $y_1(x), \ldots, y_d(x)$ such that $\{y_i(a)\} = \pi^{-1}(a)$ and $F(x, y_i(x)) = 0$ for all i. Each of them parametrizes one sheet of the covering in a neighborhood of a.

Consider a path $\gamma : [0, 1] \to \mathbb{C} \setminus S$, and choose one analytic solution of equation (1) above $\gamma(0)$. One may continue analytically this solution along γ, yielding an analytic function defined in a neighborhood of $\gamma(1)$ which is still a solution of the equation. Now take γ to be a loop with basepoint a: then analytic continuation along γ—or any other path homotopic to γ—induces a permutation of the fiber $\pi^{-1}(a)$. The action of the fundamental group $\pi_1(\mathbb{C} \setminus S, a)$ thus defined on $\pi^{-1}(a)$ is called the monodromy of $\mathcal{C}$ with basepoint a. Up to conjugation, it does not depend on the choice of the basepoint.

The monodromy action of a loop enclosing exactly one critical point c is called the local monodromy around c. The whole monodromy may be represented by the local monodromy around each critical point with respect to a common basepoint. This is the expected output of an algorithm for monodromy computation.

2. Certified monodromy computation

2.1. **General strategy.** Many methods for monodromy computation are based upon its definition by means of analytic continuation. The idea is to choose a broken line path homotopic to a bunch of loops encompassing one critical point each; then to compute (in a way to be precised) the fiber above each vertex; and finally to connect the fibers above adjacent vertices, that is, to identify values corresponding to the same solution branch.

For example, the Maple `monodromy` function, by M. van Hoeij and B. Deconinck [5], connects the fibers heuristically using first-order Taylor expansions, adapting the step length depending on the observed precision. M. van Hoeij and M. Rybowicz developed another version which provides correct error control using interval arithmetic, but it is much slower. Another idea, studied among others by D.V. and G.V. Chudnovsky [1, 2, 4], is to compute a differential equation satisfied by the algebraic function $y(x)$. This allows for fast high-precision evaluation by binary splitting [3, 10], however, according to the speaker, the size of the differential equation compared to that of the algebraic one and the conversion cost can be prohibitive. Moreover, in most cases, the high precision is not really needed.

The strategy exposed here fits into this "compute-fibers-and-connect" pattern. Following the discussion above, in a regular point, the equation admits d (convergent) formal power series solutions $y(x)$. One can show that their convergence radius is at least the distance from a to the nearest critical point. We use every other vertex (disks on Figure 2) of the path as an *expansion point*, above which we compute Taylor expansions of the d solutions. Between two successive expansion points, in the intersection of the convergence disks, lie *connection points*. Fibers above connection points are computed numerically and compared with the values assumed there by expansions above nearby expansion points to make the connection. However, when the curve has singularities close to each other, the path needs to go between them at some point, which requires high orders of truncation. Existing methods based on this sole idea tend to require unacceptable expansion orders or even to become inaccurate in those cases. To solve this problem, the method presented here allows expansion points to be singular points.

2.2. **Puiseux expansions above critical points.** Indeed, it is still possible to give formal solutions of the equation (corresponding to asymptotic expansions of "actual" algebraic functions) in the neighborhood of a critical point c, in the form of Puiseux series with nonnegative support. (Recall that F is monic.) More precisely, there exist s classes

$$y_i(x) = \sum_{k=0}^{\infty} y_{i,k} \, (x-c)^{k/e_i}, \qquad e_1 + \cdots + e_s = d$$

of e_i Puiseux series with conjugate algebraic coefficients satisfying $F(x, y_i(x)) = 0$. The e_i are called the ramification indices.

Puiseux expansions above critical points are computed using the Newton-Puiseux algorithm as improved by D. Duval [6]. Then the local monodromy can be read "for free" on the Puiseux expansions. Indeed, (almost) like power series exapansions above ordinary points, they have a sector of convergence extending up to the next critical point. This means that, once we choose a determination for the e_i-th root functions, each Puiseux expansion defines an analytic function in a slit disk centered in c. The process of analytic continuation around c permutes cyclically the branches corresponding to conjugate Puiseux expansions. As in ordinary points, numerical evaluation inside the sector of convergence allows to connect the fibers. We can also "sidestep" the critical point by evaluating the expansion in two connection points.

Example. Take $F(X,Y) = (Y^3 - X)((Y-1)^2 - X)(Y - 2 - X^2) + X^2Y^5$. Since $F(0,y) = 0 \Rightarrow y \in \{0,1,2\}$ while $\deg_Y F = 6$, the point $x_0 = 0$ is critical. The Puiseux expansions of F above 0 are:

$$\begin{aligned} Y_{1,1} &= 2 - 3X^2 - \frac{9}{2}X^3 + \cdots \\ Y_{2,k} &= 1 + X^{1/2} + (-1)^k X^{3/2} + \cdots, && k \in \{1,2\} \\ Y_{3,k} &= j^k X^{1/3} + \frac{1}{6}X^3 + \frac{5}{12}j^k X^{10/3} + \cdots, && k \in \{1,2,3\}, j^3 = 1. \end{aligned}$$

Denoting solutions by their indices above, the local monodromy is thus

$$\big((1,1)\ (1,2)\ (1,3)\big)\big((2,1)\ (2,2)\big) \in \mathfrak{S}_6.$$

All this still applies if a happens to be an ordinary point: then the Puiseux expansions are usual power series expansions, the ramification indices are all 1 and the local monodromy is the identity permutation.

2.3. **Error control for numerical connection.** In a connection point x_1, Smith's theorem on isolation of complex roots of univariate polynomials [9] allows to compute small disks $D(\tilde{y}(x_1), \rho)$ containing each exactly one root of $F(x_1, Y)$. From the minimal distance between two of these disks, we get a truncation order n in the corresponding expansion points that ensures the nearest $\tilde{y}(x_1)$ to $\sum_{k=0}^{n} y_k (x_1 - x_0)^{k/e}$ is indeed that belonging to the same branch.

Proposition 1. *Let $y(x_1) = \sum_{k=0}^{\infty} y_k (x_1 - x_0)^{k/e}$ be a Puiseux expansion of a solution of (1) above x_0. Let ρ be the distance from x_0 to the next critical point and let M be a bound on the values of y on the disk $D(x_0, \rho)$. Then for any $x_1 \in D(x_0, \rho)$, the tail of the Puiseux expansion satisfies*

$$n \geq \frac{\lg \frac{M(x_0)}{\epsilon} + \lg \frac{1}{1-\beta}}{\lg \frac{1}{\beta}} - 1 \quad \Rightarrow \Big| \sum_{k=n}^{\infty} y_k (x_1 - x_0)^{k/e} \Big| \leq \epsilon \tag{2}$$

where $\beta = \left(\frac{|x_1 - x_0|}{\rho}\right)^{1/e}$.

A suitable M may be computed from Mignotte bounds on the roots of univariate polynomials [7]. Similar bounds are used in other parts of the algorithm that require numerical values for roots of polynomials.

3. Efficiency issues

3.1. **Global monodromy and choice of the path.** Since we can obtain the local monodromy in any critical point simply by looking at the Puiseux expansions, all we need to compute the global monodromy is to express the local monodromies using the same base point. The complexity of the computation depends on the path we choose to do this connection.

First, we can bound the total number of continuation steps as follows. Suppose the base point a is given. Compute a minimal spanning tree for the usual Euclidean distance of the set formed by a and the critical points. Use the critical points and the middles of the edges as expansion points, and put one connection point near each end of each edge (figure 2). From the definition of the tree, expansions above the middle of an edge are convergent up to its ends, so this is sufficient for continuation along the edge. Continuation along several successive edges is performed by "sidestepping" the vertices as mentioned in §2.2. Now the monodromy matrix around c w.r.t. the common basepoint a is the conjugate of that w.r.t. a connection point e lying near c by the matrix connecting the fibers $\pi^{-1}(a)$ and $\pi^{-1}(e)$—which we obtain by analytic continuation along

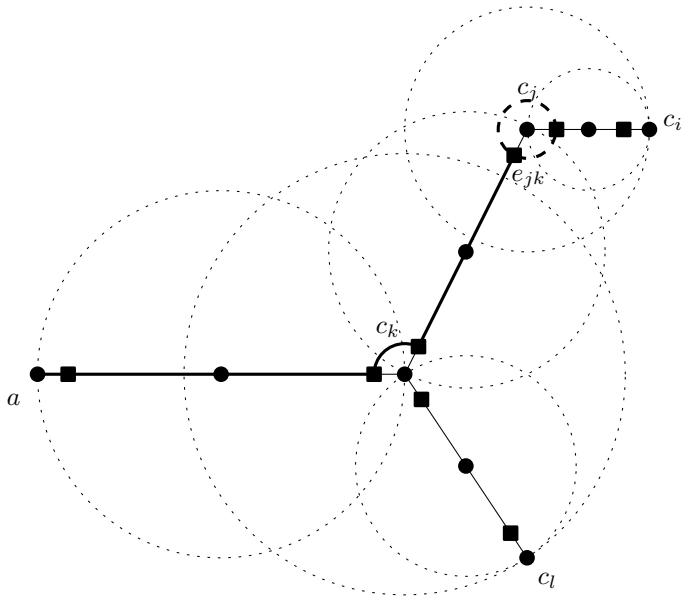


FIGURE 2. Choice of the path along which to perform analytic continuation. Disks ● are expansion points, squares ■ are connection points, dotted circles are convergence circles of Puiseux expansions. In bold, the path giving the local monodromy around c_j with basepoint a.

the path from a to e in the spanning tree. Thus, if $p \leq D^2$ (where D is the total degree of F) is the number of critical points, $2p+1$ expansions and $2p$ evaluations are enough to compute the global monodromy.

However, it turns out that minimizing the number of steps is not the best way to go. Indeed, bigger steps and steps reaching near singularities require high truncation orders. To decrease the sum of truncation orders along the path, we keep β away from 1 in Proposition 1, introducing additional steps (along the edges of the tree) as needed. Experiments show that setting $\beta = 1/2$ is a good compromise. Enforcing this value, Equation (2) becomes $n \geq \lg(M/\epsilon)$, and we get the following theorem.

Theorem 2. *Let p be the number of critical points of equation* (1)*; and let L_{min}, L_{max} be the smallest, resp. the largest distance between two of them. There is an algorithm that computes the monodromy defined by* (1) *using $O(p \lg(L_{max}/L_{min}))$ expansion and connection points, all expansion orders being bounded by $\lg(M/\epsilon)$, with the notations of Proposition 1.*

Note that M and ϵ depend on the behaviour of $y(x)$ in each expansion point.

3.2. **Towards an efficient algorithm for Puiseux series computation.** The main cost of the algorithm outlined above comes from the manipulation of Puiseux expansions above critical points. The Newton-Puiseux algorithm and its variant mentioned above compute them by performing changes of variable in the algebraic equation, guided by the form of the so-called Newton polygon of the equation, until they manage to reduce the expansion to the regular case. In order not to "miss" the multiplicities of roots of polynomials that determine the ramification type, one uses exact arithmetic on algebraic numbers. This is slow, due to the growth of the coefficients and the degrees of algebraic extensions involved. Even the numerical evaluation to connect fibers is costly, because

large cancellations can occur. However, the speaker does not wish to give up expansions above singularities: not only do they provide the local monodromy and a way to sidestep critical points, but they also give useful information for error control in other parts of the effective Abel-Jacobi theorem.

To overcome this, he suggests a hybrid numeric-modular algorithm. The idea is to compute the coefficients of the expansion numerically, while performing in parallel an exact computation modulo some suitable prime. Indeed, only the Newton polygons and the multiplicities of roots of characteristic polynomials defined by their edges really need to be exact: in other words, if all Newton polygons appearing in the course of the Newton-Puiseux algorithm are known, then it is possible to compute expansions with approximate coefficients but correct ramification by numerical methods. The difficult part is to ensure that the structure of the modular computation correctly mirrors that of the complex one. Work is in progress to devise an algorithm that outputs a "good" prime for which no degeneracy happens (at least with high probability), and to control the accuracy of the numerical part of the algorithm. A prototype heuristic implementation of these ideas in Maple already gives unexpectedly good results.

Other questions currently under investigation include understanding more precisely the good performance of the numeric-modular strategy and studying the complexity of the overall algorithm.

References

[1] Chudnovsky (David V.) and Chudnovsky (Gregory V.). – On expansion of algebraic functions in power and Puiseux series, I. *Journal of Complexity*, vol. 2, 1986.

[2] Chudnovsky (David V.) and Chudnovsky (Gregory V.). – On expansion of algebraic functions in power and Puiseux series, II. *Journal of Complexity*, vol. 3, 1987.

[3] Chudnovsky (David V.) and Chudnovsky (Gregory V.). – Computer algebra in the service of mathematical physics and number theory. In *Computers in mathematics (Stanford, CA, 1986)*. p. 109–232. – Dekker, New York, 1990.

[4] Cormier (Olivier), Singer (Michael F.), Trager (Barry M.), and Ulmer (Felix). – Linear differential operators for polynomial equations. *Journal of Symbolic Computation*, vol. 34, n° 5, 2002, pp. 355–398.

[5] Deconinck (Bernard) and van Hoeij (Mark). – Computing Riemann matrices of algebraic curves. *Physica D*, vol. 152/153, 2001, pp. 28–46. – Advances in nonlinear mathematics and science.

[6] Duval (Dominique). – Rational Puiseux expansions. *Compositio Mathematica*, vol. 70, n° 2, 1989, pp. 119–154.

[7] Mignotte (Maurice). – *Mathematics for Computer Algebra*. – Springer-Verlag, Berlin-Heidelberg-New York, 1992.

[8] Poteaux (Adrien). – Computing monodromy groups defined by plane algebraic curves. – 2007. To appear in SNC '07.

[9] Smith (Brian T.). – Error bounds for zeros of a polynomial based upon Gerschgorin's theorems. *Journal of the ACM*, vol. 17, n° 4, October 1970, pp. 661–674.

[10] Van der Hoeven (Joris). – Fast evaluation of holonomic functions. *Theoretical Computer Science*, vol. 210, n° 1, 1999, p. 199–216.

Algorithms Seminar 2006–2007,
F. Chyzak (ed.), INRIA

Available online at the URL
http://algo.inria.fr/seminars/.

Polynomial Approximation and Floating-Point Numbers

Sylvain Chevillard
Arenaire team, Lip, Ens Lyon

June 12, 2007

Summary by Marc Mezzarobba

Abstract

For purposes of numerical implementation of mathematical functions, one is interested in finding low degree polynomials with floating-point coefficients approximating a given function to a certain precision. The basic idea of the method presented here is to look for a constrained approximant interpolating the target function at the same points as the Chebyshev polynomial of given degree; which reduces to solving the shortest vector problem in a lattice. This method is simple, fast, flexible, and although heuristic, it usually gives solutions very close to the actual optimum. This is joint work with Serge Torres, and part of the speaker's PhD research, conducted under the direction of Nicolas Brisebarre and Jean-Michel Muller.

The general framework of this work—and the research subject of the Arenaire project—is the problem of certified numerical computation with mathematical functions. This includes hardware ("FPU"), fixed-precision software ("`libm`") and arbitrary precision software ("bigfloat") implementation of elementary functions as well as correct rounding for more complex operators, such as linear algebra operations. In all these cases, to implement a function f with inexact arithmetic, one may replace it by a simpler function $\tilde{f}$—here *simple* means easier to evaluate—, which in turn is computed using the available primitives, controlling the round-off error.

Here we are concerned with the choice of $\tilde{f}$ in the case where f is a real function we wish to evaluate in a fixed precision floating-point format using additions and multiplications. So we let $f : [a, b] \to \mathbb{R}$ (with $-\infty < a < b < \infty$) be a continuous real function, and we seek to approximate f by a polynomial $\tilde{f}$. (If the domain of the function we're trying to implement is too large, we may split it or use various range reduction technique to obtain a "small enough" segment.) Many usual functions come by definition with natural simple approximations, such as truncated Taylor series for analytic functions. However, they are usually inefficient: for instance, one needs 7 terms of the Taylor expansion of $\exp(1/2 + t)$ to approximate exp on $[-1, 2]$ with (absolute) error not exceeding 0.01, while there exists a polynomial of degree 4 achieving the same accuracy. So it is natural to look for polynomials of minimal degrees providing an approximation of a function to a given precision. We focus on *absolute* error; however, most of what follows adapts to the case of relative error.

Definition 1. Let $f : [a, b] \to \mathbb{R}$ be a continuous function and let $n \in \mathbb{N}$. A polynomial $\tilde{f} \in \mathbb{R}[x]$ of degree $\leq n$ minimizing the maximum absolute error

$$\|\tilde{f} - f\|_\infty = \sup_{x \in [a,b]} |\tilde{f}(x) - f(x)|.$$

is called a minimax polynomial (or a best approximation polynomial in the sense of Chebyshev) of degree n of f .

Polynomial approximation has been studied from the mathematical point of view since the 19th century. The basic result about minimax approximants is the classical theorem of Chebyshev.

Theorem 1. *A continuous function $f : [a, b] \to \mathbb{R}$ has a unique minimax polynomial. This polynomial can be characterised as the unique $\tilde{f} \in \mathbb{R}[x]$ of degree $\leq n$ such that there exists $\epsilon = \pm 1$ and $n + 2$ points $x_0 < \cdots < x_{n+1}$ satisfying*

$$\forall j, \quad \tilde{f}(x_j) - f(x_j) = \epsilon(-1)^j \|\tilde{f} - f\|_\infty.$$

In words, a polynomial $\tilde{f}$ is the minimax polynomial of degree n of f if and only if the error function $\tilde{f} - f$ reaches is maximal absolute value $n + 2$ times, with alternating signs.

E. Rémès [9] gave an algorithm to compute (arbitrarily good approximations of) the minimax polynomial with quadratic convergence. But this is not the end of the story. Indeed, what we are interested in is approximation by polynomial with floating-point coefficients. Recall a (binary, fixed-precision) floating-point number is a number of the form $m \cdot 2^e$, where the mantissa m is an integer with a fixed number t of bits. Usually the exponent e is also constrained to lie in a certain range, but we will ignore that here. The IEEE 754 standard defines several widely used floating-point formats, including "single precision" ($t = 24$) and "double precision" ($t = 53$). So our true problem is the following.

Problem 1. *Given the function f, a degree bound n and a mantissa size t (and maybe a base b, if we wish to consider floating-point numbers is bases other than 2), find a polynomial $\tilde{f} \in 2^{-\infty}\mathbb{Z}[x]$ of degree $\leq n$, with floating-point coefficients of the specified format, minimizing $\|\tilde{f} - f\|_\infty$.*

Notice that unicity is no more ensured. The naive approach of rounding each coefficient of the real minimax polynomial to obtain a floating-point polynomial may yield poor results.

Example. Take $f : [0, 1] \to \mathbb{R}, x \mapsto \lg(1 + 2^{-x})$ and look for a polynomial approximation of degree ≤ 6 with single-precision coefficients. Then the real minimax corresponds to an error of $8.3 \cdot 10^{-10}$. Rounding each of its coefficients to the nearest single-precision IEEE floating-point number gives an error of $119 \cdot 10^{-10}$, while the optimal polynomial approximation in this format achieves $10.06 \cdot 10^{-10}$.

Similar issues have already been tackled by D. Kodek [7] in the context of signal processing, but his approach is limited to small mantissas (typically $t < 10$) and low degree ($n < 20$) polynomials. W. Kahan also claims to have an efficient method; however his work is not available.

The first general method for determining the optimal floating-point minimax polynomial is due to N. Brisebarre, J.-M. Muller and A. Tisserand [3]. Their idea is the following: when looking for a polynomial of degree n, first guess the exponents $e_0, \ldots, e_n$ of the coefficients and an approximate value ϵ of $\|\tilde{f} - f\|_\infty$. Then let $\tilde{f}(x) = m_0 \cdot 2^{e_0} + \cdots + m_n \cdot 2^{e_n} x^n$ and search for $(m_0, \ldots, m_n) \in \mathbb{Z}^{n+1}$ minimizing $\|\tilde{f} - f\|_\infty$ under the additional constraints

$$-\epsilon \leq \tilde{f}(x_i) - f(x_i) \leq \epsilon \qquad \text{for "many" } x_i \tag{1}$$

using P. Feautrier's integer linear programming tool PIP [4]. Repeat until the guesses are correct.

This method provides certified results and is flexible, in the sense that it allows to find minimax polynomials with various additional constraints. However, it runs in exponential time, and it is quite sensitive to the choice of ϵ and the x_i: if ϵ is underestimated, there will be no solution, but if it is too much overestimated, the search time becomes unacceptable. The new method presented here is a fast (polynomial time) heuristic that aimed to provide good starting points in order to

speed up the method of Brisebarre *et al.* Actually the polynomial it gives turned out to be better than expected, and often good enough to be used directly.

We make the same simplification as above, namely to suppose the values of the exponents have been guessed. Starting from the exponents of the rounded real minimax is usually a good choice. So the setting is the same as above, excepted that we replace the (1) by a "approximate interpolation constraints" for $n+1$ points $x_0, \ldots, x_n$:

$$\begin{bmatrix} \tilde{f}(x_0) \\ \vdots \\ \tilde{f}(x_n) \end{bmatrix} = m_0 \begin{bmatrix} 2^{e_0} \\ \vdots \\ 2^{e_0} \end{bmatrix} + \cdots + m_n \begin{bmatrix} 2^{e_n} x_0^n \\ \vdots \\ 2^{e_n} x_n^n \end{bmatrix} \simeq \begin{bmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{bmatrix}. \tag{2}$$

This "discretisation" step is what makes the rigorous analysis of the method difficult (and the reason we call it heuristic). Let $\mathbf{b}_i = (2^{e_i} x_0^i, \ldots, 2^{e_i} x_n^i)$ and $\mathbf{f} = (f(x_0), \ldots, f(x_n))$. Equation (2) rewrites as

$$m_0 \mathbf{b}_0 + \ldots m_n \mathbf{b}_n \simeq \mathbf{f} \tag{3}$$

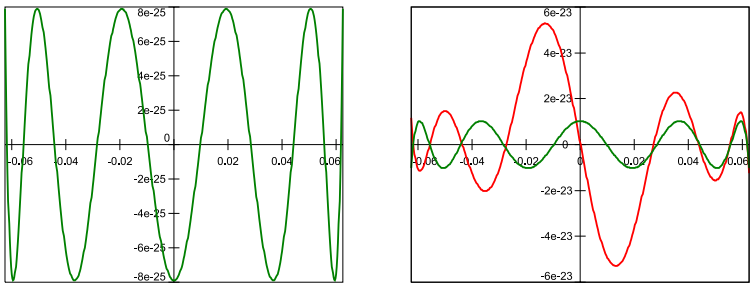
which can be understood as: find the vector closest to $\mathbf{f}$ in the lattice $\mathbb{Z}\mathbf{b}_0 + \ldots \mathbb{Z}\mathbf{b}_n \subset \mathbb{R}^{n+1}$. Let us recall the classical closest lattice vector problem (CVP).

Problem 2. *Let $\Gamma \subset \mathbb{R}^n$ be a lattice (that is, a discrete subgroup) of rank $k \leq n$, and let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. Given a basis $(\mathbf{b}_0, \ldots, \mathbf{b}_k)$ of Γ and a vector $x \in \mathbb{R}^n$, find $y^* \in \Gamma$ minimizing $\|y^* - x\|$.*

The Euclidean CVP is known to be NP-hard to solve exactly [10] or even within a subpolynomial (in n) factor, and *not* NP-hard to approximate to a factor $\sqrt{n/\log n}$ [5], still no algorithm is known that gives a polynomial approximation factor. On the practical side, it can be solved exactly using a (super-exponential) algorithm due to Kannan [6], but building on the celebrated LLL lattice reduction algorithm [8], Babai [1] gave a polynomial algorithm that finds a lattice vector y "pretty close" to a given x, namely such that $\|y - x\|_2 \leq 2^{n/2} \|y^* - x\|_2$. This is the algorithm we use. This exponential approximation factor may seem large, but (as does LLL) Babai's algorithm usually gives better results as one would expect looking at the bound.

The choice of the norm implied by the $\simeq$ sign in Equation (3) is somewhat arbitrary. What we really want is a uniform approximation; however, choosing the Euclidean norm allows us to use Babai's algorithm, and this is what we do. (If necessary, we may refine the result by looking for a closer vector w.r.t. $\|\cdot\|_\infty$ among those we get if we perturb the y we found by a linear combination with small coefficients of the "pretty short" vectors of the LLL-reduced basis of Γ.)

So the algorithm is as follows. We are given the function f, a degree n and a floating-point precision t, and we are looking for a floating-point polynomial $\tilde{f}$ (with this degree and this precision) close to f. We first compute (an approximation of) the real minimax $f^* = a_0^* + \cdots + a_n^* x^n$ by Rémès' algorithm. This gives us a first guess $e_i = 1 - t + |\lg |a_i^*||$ for the values of the exponents of $\tilde{f}$. The critical step is to choose the interpolation points x_i in order to avoid Runge's phenomenon. According to the intuition that $\tilde{f}$ will be close to f^*, and in view of Theorem 1, we use the $n+1$ points x such that $f^*(x) = f(x)$. We finally solve the closest vector problem 3 by Babai's algorithm as explained above. Of course, the guessed exponents may not be optimal. So we use the following heuristic: if one of the computed coefficients a_i has a mantissa m_i that doesn't fit on t bits (which suggests our choice of exponents was wrong), we repeat the process after shifting e_i by $|\lg m_i|$ (that is, replacing $a_i^a st$ by a_i in the formula used to guess the exponents) to take into account the order of magnitude of a_i. There is no proof that this converges; but it did after only two or three iterations on all the examples the speaker tried, even when the initial exponents were far off.



(a) Real minimax (b) First try

In practice, the running time of the algorithm is essentially that of Rémès' algorithm (which is called only once even if one wishes, say, to try several floating-point formats).

Example. Finally let us look at an example that shows how the above method can be tweaked to solve more complex cases with additional constraints. John Harrison (Intel) asked the speaker's team for a degree 9 polynomial approximating $f(x) = (2^x - 1)/x$ for $x \in [-1/16, 1/16]$ to a precision $\epsilon \lesssim 2^{-74} \simeq 5.30 \cdot 10^{-23}$. The coefficients were to be extended double precision ($t = 64$), except for the constant one, which was to be searched for as the sum of two "extended doubles".

Rémès' algorithm gives $\|f_8^* - f\|_\infty < 40.1 \cdot 10^{-23}$ and $\|f_9^* - f\|_\infty < 0.08 \cdot 10^{-23}$ for the minimax polynomials of degree 8 and 9 respectively, so degree 9 should indeed be a good choice for the target precision. It also provides interpolation points (Figure (a)). Applying the strategy above unchanged, we get $\tilde{f}$ s.t. $\|\tilde{f} - f\|_\infty < 5.32 \cdot 10^{-23}$: so the challenge was well chosen! (In comparison, rounding f_9^* gives $\|f_{\text{rnd}}^* - f\|_\infty \simeq 40.35 \cdot 10^{-23}$.) But look at Figure (b): $\tilde{f}$ does not respect the interpolation constraint! Indeed, we observe that the slope of $\tilde{f}$ in 0 is very constrained – a good polynomial $\tilde{f}$ should satisfy $\tilde{a}_1 \simeq (\lg 2)^2/2$. To take this into account, we look for $\tilde{g} = b_0 + b_2 x^2 + \ldots b_9 x^9$ approximating $g(x) = f(x) - a_1 x$. Adapting our strategy to enforce $g_1 = 0$, we finally get $\|\tilde{g} - g\|_\infty < 4.45 \cdot 10^{-23}$.

Although it worked on the last example above, Rémès' algorithm is not sufficient in general to compute real constrained minimax polynomials (such as g^* here), so a better algorithm is needed to generalize their use. Work is also in progress to extend the method exposed here to other types of approximants, such as rational polynomials and sums of cosines.

References

[1] Babai (László). – On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, vol. 6, n° 1, 1986, pp. 1–14.

[2] Brisebarre (Nicolas) and Chevillard (Sylvain). – Efficient polynomial L^∞-approximations. – Inria research report, 2006.

[3] Brisebarre (Nicolas), Muller (Jean-Michel), and Tisserand (Arnaud). – Computing machine-efficient polynomial approximations. *ACM Trans. Math. Softw*, vol. 32, n° 2, 2006.

[4] Feautrier (Paul). – Parametric integer programming. *Operations Research*, vol. 22, n° 3, 1988, pp. 243–268.

[5] Goldreich and Goldwasser. – On the limits of nonapproximability of lattice problems. *Journal of Computer and System Sciences*, vol. 60, 2000.

[6] Kannan (Ravi). – *Minkowski's convex body theorem and integer programming.* – Technical Report n° CMU-CS-86-105, Computer Science Dept., Carnegie-Mellon University, 1986.

[7] Kodek (Dušan M.). – An approximation error lower bound for integer polynomial minimax approximation. *Elektroteh. vestn.*, vol. 69, n° 5, 2002, pp. 266–272.

[8] Lenstra (Arjen K.), Lenstra, Jr. (Hendrik W.), and Lovász (László). – Factoring polynomials with rational coefficients. *Mathematische Annalen*, vol. 261, 1982, pp. 515–534.

[9] Rémès (Evgeny). – Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *Comptes Rendus de l'Académie des Sciences*, vol. 198, 1934, pp. 2063–2065.

[10] van Emde Boas (Peter). – *Another NP-complete Partition Problem and the Complexity of Computing Short Vectors in Lattices.* – Technical Report n° 81-04, Mathematics Department, University of Amsterdam, 1981.

I wish to thank Sylvain Chevillard for providing very useful comments on a draft of this summary and allowing me to use the graphs (a) and (b) from his presentation.

BIBLIOGRAPHIE

[ABvH06] Sergei A. Abramov, Moulay A. Barkatou, and Mark van Hoeij. *Apparent singularities of linear difference equations with polynomial coefficients*. Appl. Algebra Eng. Commun. Comput, 17(2):117–133, 2006.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing, 1974. Addison-Wesley Series in Computer Science and Information Processing.

[AL04] Jounaidi Abdeljaoued et Henri Lombardi. *Méthodes matricielles — Introduction à la complexité algébrique*. Springer, 2004.

[BB87] Jonathan M. Borwein and Peter B. Borwein. *Pi and the AGM*. John Wiley, 1987.

[BCS97] Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlagen der mathematischen Wissenschaften*. Springer Verlag, 1997.

[BCS05] Alin Bostan, Thomas Cluzeau, and Bruno Salvy. *Fast algorithms for polynomial solutions of linear differential equations*. In Manuel Kauers (editor): *ISSAC'05*, pages 45–52, New York, 2005. ACM Press. Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation, July 2005, Beijing, China.

[BCS07] Alin Bostan, Frédéric Chyzak et Bruno Salvy. *D-finitude : algorithmes et applications*. `http://algo.inria.fr/EJCIM07/`, notes de cours, École jeunes chercheurs informatique mathématique, 2007.

[Ber87] Daniel J. Bernstein. *New fast algorithms for π and e*, 1987. `http://cr.yp.to/bib/1987/bernstein.html`, paper for the Westinghouse competition, distributed widely at the Ramanujan Centenary Conference.

[Ber01] Daniel J. Bernstein. *Multidigit multiplication for mathematicians*. `http://cr.yp.to/papers/m3.pdf`, to appear, 2001.

[Ber04] Daniel J. Bernstein. *Fast multiplication and its applications*, 2004. `http://cr.yp.to/papers.html#multapps`, to appear in Buhler-Stevenhagen *Algorithmic number theory* book.

[BGS72] Michael Beeler, R. William Gosper, and Rich Schroeppel. *Hakmem*. Ai memo 239, MIT Artificial Intelligence Laboratory, 1972. `http://hdl.handle.net/1721.1/6086`.

[Blä03] Markus Bläser. *On the complexity of the multiplication of matrices of small formats*. J. Complex., 19(1):43–60, 2003, ISSN 0885-064X.

[Bre76a] Richard P. Brent. *The complexity of multiple-precision arithmetic*. The Complexity of Computational Problem Solving, pages 126–165, 1976. `http://wwwmaths.anu.edu.au/~brent/pub/pub032.html`.

[Bre76b] Richard P. Brent. *Fast multiple precision evaluation of elementary functions*. Journal of the ACM, 23:242–251, 1976. `http://wwwmaths.anu.edu.au/~brent/pub/pub034.html`.

[BT32] George D. Birkhoff and Waldemar J. Trjitzinsky. *Analytic theory of singular difference equations*. Acta Mathematica, 60:1–89, 1932.

[BZ07] Marco Bodrato and Alberto Zanoni. *Integer and polynomial multiplication: Towards optimal Toom-Cook matrices*. In C. W. Brown (editor): *ISSAC 2007: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, 2007.

[CC87] David V. Chudnovsky and Gregory V. Chudnovsky. *Computer assisted number theory with applications*. In *Number theory (New York, 1984–1985)*, volume 1240 of *Lecture Notes in Mathematics*, page 1–68. Springer, Berlin, 1987.

[CC90] David V. Chudnovsky and Gregory V. Chudnovsky. *Computer algebra in the service of mathematical physics and number theory*. In *Computers in mathematics (Stanford, CA, 1986)*, page 109–232, New York, 1990. Dekker.

[CHT+07] Howard Cheng, Guillaume Hanrot, Emmanuel Thomé, Eugene Zima, and Paul Zimmermann. *Time- and space-efficient evaluation of some hypergeometric constants*, 2007.

[Chy98] Frédéric Chyzak. *Groebner bases, symbolic summation and symbolic integration*. In B. Buchberger and F. Winkler (editors): *Groebner Bases and Applications (Proc. of the Conference 33 Years of Gröbner Bases)*, volume 251 of *London Mathematical Society Lecture Notes Series*, pages 32–60. Cambridge University Press, 1998. ISBN 0-521-63298-6.

[Coh93] Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, 1993.

[CS98] Frédéric Chyzak and Bruno Salvy. *Non-commutative elimination in Ore algebras proves multivariate holonomic identities*. Journal of Symbolic Computation, 26(2):187–227, August 1998.

[CW90] Don Coppersmith and Shmuel Winograd. *Matrix multiplication via arithmetical progressions*. J. Symbolic Computation, 9:251–280, 1990.

[DL89] Jan Denef and Leonard Lipshitz. *Decision problems for differential equations*. Journal of Symbolic Logic, 54(3):941–950, sep 1989.

[FS07] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Web edition. Ninth printing (Valentine's), 2007. `http://algo.inria.fr/flajolet/Publications/book070503.pdf`, chapters I to IX of a book to appear (Cambridge University Press).

[Für07] Martin Fürer. *Fast integer multiplication*. `http://www.cse.psu.edu/~furer/Papers/mult.pdf`, to appear, 2007.

[FZ77] Charles M. Fiduccia and Yechezkel Zalcstein. *Algebras having linear multiplicative complexities*. JACM: Journal of the ACM, 24, 1977.

[GKZ07] Pierrick Gaudry, Alexander Kruppa, and Paul Zimmermann. *A GMP-based implementation of Schönhage-Strassen's large integer multiplication algorithm*. In C. W. Brown (editor): *ISSAC 2007: Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, 2007.

[Gra07] Torbjörn Granlund. *GNU Multiple Precision Arithmetic Library*, 2007. `http://gmplib.org`.

[Har72] Richard Harter. *The optimality of Winograd's formula*. Comm. ACM, 15(5):352, 1972.

[Hen77] Peter Henrici. *Applied and Computational Complex Analysis*, volume II. Wiley-Interscience, 1977.

[HK71] John E. Hopcroft and Leslie R. Kerr. *On minimizing the number of multiplications necessary for matrix multiplication*. SIJAM: SIAM Journal on Applied Mathematics, 20, 1971.

[HP97] Bruno Haible and Thomas Papanikolaou. *Fast multiprecision evaluation of series of rational numbers*, October 08 1997. `http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/papanik/ps/TI-97-7.ps.gz`.

[Inc56] Edward L. Ince. *Ordinary Differential Equations*. Dover, New York, 1956.

[Kar91] E. A. Karatsuba. *Fast computation of transcendental functions*. Dokl. Akad. Nauk SSSR, 318(2):278–279, 1991, ISSN 0002-3264.

[Knu97a] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, third edition, 1997, ISBN 0-201-89683-4.

[Knu97b] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, third edition, 1997, ISBN 0-201-89684-2.

[KS73] Peter Kogge and Harold Stone. *A parallel algorithm for the efficient solution of a general class of recurrence equations*. IEEE Transactions on Computers, C-22:786–793, 1973.

[Lan07] Joseph M. Landsberg. *Geometry and the complexity of matrix multiplication*, March 12 2007. `http://arxiv.org/abs/cs/0703059`.

[Lip89] Leonard Lipshitz. *D-finite power series*. Journal of Algebra, 122(2):353–373, 1989.

[Odl95] Andrew M. Odlyzko. *Asymptotic enumeration methods*. In *Handbook of Combinatorics*, volume 2. 1995. `http://www.dtc.umn.edu/~odlyzko/doc/asymptotic.enum.pdf`.

[Pot07] Adrien Poteaux. *Computing monodromy groups defined by plane alebraic curves*. Submitted to SNC'07, 2007.

[Pro76] Robert L. Probert. *On the additive complexity of matrix multiplication*. SIAM Journal on Computing, 5(2):187–203, June 1976.

[Ric68] Daniel Richardson. *Some undecidable problems involving elementary functions of a real variable*. Journal of Symbolic Logic, 33(4):514–520, 1968.

[RS02] Qazi I. Rahman and Gerhard Schmeisser. *Analytic Theory of Polynomials*. Oxford University Press, 2002, ISBN 0198534930.

[SGV94] Arnold Schönhage, Andreas F. W. Grotefeld, and Ekkehart Vetter. *Fast algorithms — A multitape Turing machine implementation*. B.I. Wissenschaftsverlag, Mannheim-Leipzig-Wien-Zürich, 1994.

[Smi02] Warren D. Smith. *Fast matrix multiplication formulae — report of the prospectors*. http://www.math.temple.edu/~wds/prospector.pdf, draft, 2002.

[SS71] Arnold Schönhage und Volker 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.

[Str77] Volker Strassen. *Einige Resultate über Berechnungskomplexität*. Jber. Deutsch. Math.-Verein., 78(1):1–8, 1976/77.

[SZ94] Bruno Salvy and 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.

[Tsa00] Harrison Tsai. *Weyl closure of a linear differential operator*. J. Symbolic Computation, 29(4-5):747–775, 2000.

[vdH97] Joris van der Hoeven. *Automatic asymptotics*. PhD thesis, École polytechnique, France, 1997. http://www.math.u-psud.fr/~vdhoeven/Books/phd.ps.gz.

[vdH99] Joris van der Hoeven. *Fast evaluation of holonomic functions*. Theoretical Computer Science, 210(1):199–216, 1999. http://www.math.u-psud.fr/~vdhoeven/Publs/1997/TCS.ps.gz.

[vdH01] Joris van der Hoeven. *Fast evaluation of holonomic functions near and in regular singularities*. Journal of Symbolic Computation, 31(6):717–743, 2001. http://www.math.u-psud.fr/~vdhoeven/Publs/2000/singhol.ps.gz.

[vdH03] Joris van der Hoeven. *Majorants for formal power series*. Technical Report 2003-15, Université Paris-Sud, Orsay, France, 2003. http://www.math.u-psud.fr/~vdhoeven/Publs/2003/maj.ps.gz.

[vdH05] Joris van der Hoeven. *Effective analytic functions*. Journal of Symbolic Computation, 39(3-4):433–449, 2005. http://www.math.u-psud.fr/~vdhoeven/Publs/2003/effan.ps.gz.

[vdH06] Joris van der Hoeven. *On effective analytic continuation*. Technical Report 2006-15, Univ. Paris-Sud, 2006. http://www.math.u-psud.fr/~vdhoeven/Publs/2006/riemann.ps.gz.

[vdH07] Joris van der Hoeven. *Efficient accelero-summation of holonomic functions*. Journal of Symbolic Computation, 2007. http://www.math.u-psud.fr/~vdhoeven/Publs/2005/reshol.ps.gz.

[vdPS03] Marius van der Put and Michael F. Singer. *Galois Theory of Linear Differential Equations*, volume 328 of *Grundlehren der mathematischen Wissenschaften*. Springer, 2003.

[vzGG03] Joachim von zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, New York, 2nd edition, 2003.

[Wak70] Abraham Waksman. *On Winograd's algorithm for inner products*. IEEE Transactions on Computers, C-19(4):360–361, 1970.

[Win77] Shmuel Winograd. *Some bilinear forms whose multiplicative complexity depends on the field of constants*. Mathematical Systems Theory, 10:169–180, 1977.

[WZ85] Jet Wimp and Doron Zeilberger. *Resurrecting the asymptotics of linear recurrences*. Journal of Mathematical Analysis and Applications, 111:162–176, 1985.

[Yac07] Yacas Team. *The Yacas Book of Algorithms*, Yacas version 1.0.63 edition, january 2007. `http://yacas.sourceforge.net/Algo.book.pdf`.

[Zei90] Doron Zeilberger. *A holonomic systems approach to special functions identities*. Journal of Computational and Applied Mathematics, 32(3):321–368, 1990.