Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Intégration numérique avec erreur bornée en précision arbitraire

THÈSE

présentée et soutenue publiquement le 4 décembre 2006

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Laurent Fousse

Composition du jury

Président :	Jean-Claude BAJARD, Professeur, Université de Montpellier 2
Rapporteurs :	Fabienne JÉZÉQUEL, Maître de conférences, Université Panthéon-Assas, Paris Bruno SALVY, Directeur de recherche INRIA, Rocquencourt
Examinateurs :	Didier GALMICHE, Professeur, Université Henri-Poincaré, Nancy Norbert MÜLLER, Professeur, Universität Trier Paul ZIMMERMANN, Directeur de recherche INRIA, Nancy (Directeur de thèse)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Version du 28 février 2007. Mis en page avec la classe thloria.

Remerciements

J'ai bénéficié lors de la préparation de ma thèse de l'aide de nombreuses personnes. J'aimerais leur exprimer ma profonde gratitude par ces quelques lignes.

Mes premiers remerciements vont naturellement à Paul Zimmermann pour avoir accepté de m'encadrer avec patience et bonne humeur tout au long de ces trois ans. Déjà en stage de DEA il m'a fait découvrir (et apprécier !) l'arithmétique des ordinateurs, et ses conseils précieux m'ont guidé pendant ma préparation.

Je remercie chaleureusement Fabienne Jézéquel et Bruno Salvy pour avoir accepté d'être les rapporteurs de ce manuscript. Leur relecture minutieuse en a grandement amélioré la qualité.

Je tiens à remercier Jean-Claude Bajard pour m'avoir fait l'honneur de présider mon jury de thèse.

Je remercie Norbert Müller pour avoir accepté de faire partie de mon jury. Nos rencontres au gré des conférences et en particulier notre participation en tant que concurrents à la compétition de calcul *Many Digits* ont été l'occasion d'une émulation et de discussions fort enrichissantes.

Je remercie Didier Galmiche pour avoir accepté de faire partie de mon jury. Je lui suis notamment reconnaissant de m'avoir prodigué ses conseils et encouragements concernant la fin et le juste-après thèse.

Je n'oublie pas les membres de l'équipe Spaces (maintenant Cacao) qui m'ont accueilli dans une atmosphère de travail motivante et qui m'ont apporté une aide scientifique précieuse. Merci donc à Pierrick, Guillaume, Vincent, Patrick, Damien. Je remercie tout particulièrement Emmanuel pour l'infatigable support technique LaTeX qu'il m'a apporté, et Marion dont le soutien et les encouragements ont beaucoup compté dans cette période compliquée qu'est la fin de thèse.

Ma reconnaissance va aussi au LORIA et à sa directrice pour m'avoir permis de travailler dans d'excellentes conditions. J'ai pu avancer sereinement et dans une ignorance quasi totale des tracasseries administratives grâce à notre assistante de projet Céline que je remercie pour son efficacité et son enthousiasme.

Je voudrais enfin remercier mes amis proches et ma famille, en particulier mes parents. Ils m'ont tous manifesté un soutien et une confiance tranquille y compris aux moments où je doutais moi-même.

Table des matières

Table des figures

1

Introduction

1 Problématique

1.1 Intégration numérique

La notion d'intégrale est essentielle en analyse et conduit à de nombreux développements théoriques aussi bien que pratiques. On peut citer naturellement le calcul différentiel comme prolongement évident, mais on retrouve le besoin de calculer des intégrales notamment en théorie analytique des nombres, dans le domaine des calculs graphiques sur ordinateurs, et bien entendu aussi dans des applications physiques.

L'intégration *numérique* en elle-même se distingue de l'intégration *symbolique* dont l'objet est de fournir une expression mathématique équivalente à une expression faisant intervenir une intégrale. Pour illustration, un système de calcul symbolique pourra calculer

$$\int_0^{17} \frac{\mathrm{d}x}{1+x^2} = \arctan(17)$$

tandis qu'une intégration numérique se contentera de répondre :

$$\int_0^{17} \frac{\mathrm{d}x}{1+x^2} \approx 1,512040504...$$

A priori la résolution symbolique de formules comportant des intégrales est préférable : on obtient une expression plus simple qu'il sera possible de continuer à manipuler par la suite en tant que sous-expression de l'expression globale. Par exemple le système de calcul symbolique saura montrer que

$$\int_0^{17} \frac{\mathrm{d}x}{1+x^2} - \arctan(17) = 0$$

tandis qu'un système numérique calculera une valeur approchée de l'expression gauche en se rapprochant toujours plus de zéro à mesure que la précision demandée grandit mais sans pouvoir décider de la nullité de l'expression.

D'autre part le calcul numérique d'une intégrale fait appel à un nombre plus ou moins important d'évaluations de la fonction à intégrer, ce qui a des implications autant sur le temps de calcul que sur la précision. À titre d'exemple on peut définir pour $0 \leq x \leq 1$:

$$\begin{aligned} f(t) &= 2\sqrt{1-t^2} \\ F(x) &= \int_0^x f(t)\mathrm{d}t = \int_0^x 2\sqrt{1-t^2}\mathrm{d}t = \arcsin(x) + x\sqrt{1-x^2}. \end{aligned}$$

Le calcul de $F(x)$ via l'expression de droite pour une valeur donnée de x demande le calcul d'une racine carrée et d'un arc-sinus en plus de quelques opérations élémentaires. En revanche l'évaluation numérique de l'intégrale demande le calcul de $2\sqrt{1-t^2}$ pour un certain nombre de valeurs de t choisies dans $[0, x]$. Le nombre exact de points à utiliser dépendra de la méthode d'intégration choisie et de la précision voulue mais il est légitime de penser que dans le cas où la fonction f est fournie comme une boîte noire à la routine d'intégration numérique alors le résultat fourni pour F sera moins précis et demandera plus de temps de calcul que via le calcul direct de l'expression symbolique sans intégrale, pour une même précision.

Dans ces conditions, pourquoi vouloir intégrer numériquement malgré tout ? On peut avancer principalement deux arguments.

Tout d'abord, l'intégration symbolique peut échouer. Comme exemple bien connu, la primitive

$$x \mapsto \int_0^x e^{-t^2}\mathrm{d}t$$

de la fonction $t \mapsto e^{-t^2}$ n'admet pas d'expression close à base de primitives algébriques, logarithmiques et exponentielles. Le recours à l'intégration numérique est donc parfois inévitable (il ne l'est pas dans ce cas particulier de la fonction $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} \mathrm{d}t$ car il existe d'autres méthodes numériques comme utiliser le développement de Taylor).

D'autre part l'intégration symbolique n'est pas toujours plus rapide ou plus fiable qu'une intégration numérique. L'exemple suivant, tiré de [12] l'illustre :

$$\int_0^x \frac{\mathrm{d}t}{1+t^4} = \frac{1}{4\sqrt{2}} \log \frac{x^2 + x\sqrt{2} + 1}{x^2 - x\sqrt{2} + 1} + \frac{1}{2\sqrt{2}} \left(\arctan \frac{x}{\sqrt{2} - x} + \arctan \frac{x}{\sqrt{2} + x} \right). \tag{1.1}$$

Un calcul numérique mené via la formule exacte est non trivial et fait apparaître suffisamment de fonction spéciales pour que l'intégration numérique soit plus rapide et plus fiable. L'évaluation de la formule exacte pour (1.1) donne $0,8669729867$ sous Maple 10 soit une erreur de $6,4$ ulp décimaux (tandis que tous les chiffres affichés pour son évaluation par intégration numérique sont exacts : $0,8669729873$).

Dans certains cas enfin le choix ne se pose même pas ; c'est le cas des fonctions connues uniquement par une boîte noire (on peut demander son image en certains points mais pas son expression sous forme close, même si elle en avait une) ou encore lorsque les valeurs de la fonction ne sont connues qu'en un nombre fini de points par des mesures expérimentales.

1.2 Arrondi correct

Si les outils permettant de calculer numériquement des expressions ne manquent pas, on constate hélas qu'ils restent en général muets concernant la précision des résultats calculés. Beaucoup de logiciels sont développés sans se poser la question de la correction du résultat retourné à l'utilisateur, que ce soit dans un langage de programmation généraliste ou dans un système conçu pour les calculs numériques et formels.

On utilise le terme de « précision » pour désigner deux choses différentes mais liées. D'une part la précision d'une *variable* indique la taille en nombre de chiffres (ou bits) utilisés pour conduire les calculs (on peut par exemple effectuer tous les calculs avec une précision de 53 bits) ; d'autre part on parle de la précision d'une *valeur* pour indiquer son écart à la valeur théorique exacte.

À titre d'exemple nous pouvons demander à Maple 10 la valeur de l'intégrale

$$I = \int_{17}^{42} e^{-x^2} \log x \mathrm{d}x.$$

```
fousse@tate:~$ maple10
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x), x=17..42));
                                          -126
                                0.2604007480 10
```

L'utilisateur est en droit de se demander combien des chiffres affichés dans ce résultat sont utiles ; par exemple si la valeur exacte était plus proche de $0,261 \cdot 10^{-126}$ alors seuls les deux premiers chiffres décimaux du résultat seraient corrects, donc la précision du résultat affiché serait de 2 chiffres décimaux.

L'utilisateur aurait-il l'audace de ne pas faire confiance à Maple et de demander son avis à Pari/GP version 2.3.0, le résultat le laisserait sans doute perplexe :

```
laurent@tortoise:~$ gp
Reading GPRC: /etc/gprc ...Done.

                GP/PARI CALCULATOR Version 2.3.0 (released)
      powerpc running linux (portable C/GMP-4.2.1 kernel) 32-bit version
   compiled: Jun 29 2006, gcc-4.1.2 20060613 (prerelease) (Debian 4.1.1-5)
              (readline v5.1 enabled, extended help available)

                    Copyright (C) 2000-2006 The PARI Group

PARI/GP is free software, covered by the GNU General Public License, and comes
WITHOUT ANY WARRANTY WHATSOEVER.

Type ? for help, \q to quit.
Type ?12 for how to get moral (and possibly technical) support.

parisize = 4000000, primelimit = 500000
? intnum(x=17, 42, exp(-x^2)*log(x))
%1 = 2.5657285005610514829173563960 E-127
```

Les deux systèmes de calcul diffèrent dès le deuxième chiffre significatif. Sur cet exemple précis Pari/GP a raison (tous les chiffres du résultat affiché par Pari sont corrects, si l'on fait confiance aux calculs effectués avec une précision doublée par Pari, Mathematica 5 et CRQ) : il ne s'agit pas de jeter la pierre à Maple mais bien d'illustrer que lorsque l'on demande une approximation numérique à son système de calcul favori, il est bon de prendre le résultat avec une bonne dose de circonspection.

Au delà de l'intérêt évident de renvoyer des résultats corrects, faire l'effort de garantir une borne d'erreur sur le résultat apporte d'autres avantages. Un calcul numérique est rarement isolé, mais s'insère comme une étape dans un calcul global plus complexe. Pour connaître la précision obtenue sur le résultat final il est nécessaire de pouvoir borner les erreurs commises à chaque étape du calcul. En reprenant l'exemple de l'intégrale (1.1), calculer sa valeur pour une valeur de x choisie avec forme exacte (symbolique) demande de maîtriser l'erreur dans le calcul de $\sqrt{2}$, celui d'arc-tangente, du logarithme et bien entendu lors de chaque calcul élémentaire que sont addition, soustraction, multiplication et division.

Lorsque la différence entre la valeur exacte et la valeur renvoyée est plus petite que le poids du dernier chiffre renvoyé, on dit que la valeur renvoyée est un arrondi fidèle de la valeur exacte. Par exemple $0,2565728500561051 \cdot 10^{-126}$ et $0,2565728500561052 \cdot 10^{-126}$ sont deux arrondis fidèles pour une précision de 16 chiffres décimaux de la valeur $0,25657285005610514829173563960 \cdot 10^{-126}$. On peut exiger plus et demander la valeur de l'arrondi vers zéro, ou par exemple l'arrondi au plus proche de la valeur exacte. Il s'agit alors d'une valeur définie de manière unique (sauf pour l'arrondi au plus proche lorsque la valeur à arrondir se trouve au milieu de deux nombres flottants auquel cas on choisit par convention celle dont le dernier chiffre est pair).

Dans les débuts du développement de l'informatique, chaque constructeur définissait pour sa machine un format de stockage des nombres flottants. En particulier la plage d'exposant et la taille de la mantisse pouvaient varier d'un constructeur à l'autre. De même la sémantique des calculs n'était pas rigoureusement définie, même pour les opérations élémentaires. Une telle variété de format et de comportement rendait les programmes faisant appel à des calculs en

nombres flottants très peu portables ; les résultats variant d'un système à l'autre en fonction des opérations utilisées, de la précision interne utilisée par le système et du soin que le constructeur avait mis à programmer chacune de ces opérations.

L'avènement de la norme IEEE754 [22] en 1985 met fin à cette « tour de Babel du calcul flottant » en définissant quatre formats de nombres flottants et quatre modes d'arrondi. Des quantités spéciales sont ajoutées pour représenter les infinis et les résultats d'opérations non définies (NaN pour *Not a Number*) et surtout, la norme impose que les opérations de base respectent la règle de l'arrondi correct (voir la Définition 4). Les programmes deviennent portables entre systèmes respectant la norme car le résultat d'un calcul est uniquement défini pour toutes les opérations couvertes par la norme. De plus l'échange de valeurs flottantes est facilité entre systèmes hétérogènes car le format de représentation d'un nombre flottant en machine est imposé.

2 État de l'art algorithmique

Les méthodes d'intégration numérique proposées dans les systèmes et bibliothèques de calcul sont diverses. Les systèmes diffèrent autant par les méthodes à proprement parler que par la façon de les appliquer, que l'on pourrait appeler la stratégie d'intégration.

Nous allons faire un panorama de ces méthodes et stratégies.

2.1 Méthode de Newton-Cotes

La famille des méthodes de Newton-Cotes est une généralisation de la définition élémentaire de l'intégrale par les sommes de Riemann. La famille est paramétrée par le nombre n de points équidistants d'évaluation de la fonction ; la valeur de l'intégrale calculée étant égale à la valeur de l'intégrale du polynôme interpolateur passant par ces points.

Cette famille de méthodes est en général décrite pour des raisons historiques ou pédagogiques. Elle est communément tenue pour inférieure aux autres méthodes connues.

2.2 Méthode de Romberg

La méthode de Romberg est une application de la méthode de Richardson [31] à la règle des trapèzes. Pour un paramètre entier m la méthode des trapèzes sur $[a, b]$ de pas d'intégration $h = \frac{b-a}{m}$ est l'application de la méthode de Newton-Cotes à 2 points composée m fois, elle calcule

$$I(h) = h\left(\frac{1}{2}(f(a) + f(b)) + \sum_{i=1}^{m-1} f(a + ih)\right)$$

comme valeur approchée de $\int_a^b f(x)\mathrm{d}x$. On peut montrer (par exemple en appliquant la formule de sommation d'Euler-Maclaurin) que le terme d'erreur ne contient que des puissances paires de h :

$$\left|I(h) - \int_a^b f(x)\mathrm{d}x\right| = a_0h^2 + a_1h^4 + \mathcal{O}(h^6). \tag{1.2}$$

On pose

$$T_{k,0} = I\left(\frac{h}{2^k}\right)$$

et on commence par calculer $T_{0,0}, T_{1,0}, \ldots, T_{n,0}$ pour une certaine valeur de n.

La méthode d'extrapolation de Richardson calcule une pyramide de différences pour augmenter l'ordre de convergence :

$$\begin{array}{llll} T_{0,0} & T_{0,1} & \cdots & T_{0,n-1} \quad T_{0,n} \\ T_{1,0} & T_{1,1} & \cdots & T_{1,n-1} \\ \vdots & \vdots & & \\ \vdots & T_{n-1,1} & & \\ T_{n,0} & & & \end{array}$$

Des combinaisons linéaires permettent de passer d'une colonne à l'autre, le but étant à chaque colonne d'annuler un terme d'erreur supplémentaire dans l'équation (1.2). Plus précisément on obtient la formule

$$T_{k,l+1} = \frac{2^{2l+2} T_{k+1,l} - T_{k,l}}{2^{2l+2} - 1}$$

pour passer d'une colonne l à la colonne $l+1$. La valeur effectivement renvoyée par l'algorithme est $T_{0,n}$ et son erreur est $\mathcal{O}(h^{2(n+1)})$ si f est suffisamment régulière, toute considération d'erreur d'arrondi mise à part.

2.3 Méthode de Gauss

Les méthodes de Gauss sont des méthodes d'intégration pondérées, en ce sens que pour une fonction f et un domaine d'intégration $[a, b]$ elles calculent

$$I_w(f) = \int_a^b w(x) f(x) \mathrm{d}x$$

pour une certaine fonction de poids w vérifiant quelques propriétés. L'étude de la méthode I_w passe naturellement par celle du produit scalaire implicitement défini par w sur les fonctions $[a, b] \to \mathbb{R}$, qui définit une famille de polynômes orthonormaux.

Le cas particulier de la fonction de poids $w = 1$ est appelé méthode de Gauss-Legendre.

Dans un certain sens, les méthodes de Gauss restent « élémentaires » puisqu'elles s'écrivent aussi comme les méthodes de Newton-Cotes via l'interpolation de f en n points bien choisis. L'expérience semble montrer que ces méthodes sont les plus efficaces pour intégrer des fonctions régulières sur des segments de $\mathbb{R}$ [2].

Il est possible de calculer simultanément tous les points de la méthode par la détermination des valeurs propres d'une matrice associée aux polynômes orthonormaux [20]. La complexité de cet algorithme matriciel est $\mathcal{O}(n^3)$.

Méthode de Gauss-Kronrod

Les méthodes de Gauss-Kronrod sont une variation des méthodes de Gauss. Elles sont aussi paramétrées par une fonction de poids w et la méthode d'ordre n utilise $2n + 1$ points dont n correspondent aux points de la méthode de Gauss pour le même poids. Les $n + 1$ points restants sont déterminés par la condition que la méthode d'intégration doit être exacte pour tous les polynômes de degré au plus $3n + 1$.

Les coefficients peuvent être calculés par une méthode matricielle, comme pour les coefficients de Gauss [8, 23].

2.4 Méthode de Clenshaw-Curtis

La méthode de Clenshaw-Curtis est encore une méthode de type interpolatoire, paramétrée par le nombre de points n, avec un choix des points d'intégration sur l'intervalle $[-1, 1]$ de

$$x_j = \cos\left(\frac{j\pi}{n}\right)$$

et les poids correspondants calculés pour intégrer correctement les polynômes de degré au plus $n-1$.

Du fait de ce choix de points, la méthode n'est pas optimale au sens de l'approximation polynomiale de plus haut degré comme c'est le cas des méthodes de Gauss, et c'est ce qui lui vaut d'être négligée en pratique au profit de celles-ci.

Une étude comparative des méthodes de Gauss et de Clenshaw-Curtis est faite dans [35], qui montre en particulier que le facteur 2 en terme de meilleure approximation polynomiale n'est pas confirmé en pratique : si l'on considère le nombre minimal de points à utiliser pour atteindre une précision donnée, alors la méthode de Gauss n'est pas meilleure d'un facteur 2 (mais tout de même supérieur à 1) par rapport à la méthode de Clenshaw-Curtis.

2.5 Méthode doublement exponentielle

Les méthodes doublement exponentielles (DE) reposent sur le choix d'un changement de variable qui transforme un domaine d'intégration borné à $\mathbb{R}$, et sur l'optimalité de la méthode des trapèzes sur $\mathbb{R}$ pour les fonctions analytiques. Pour une fonction f intégrable sur $\mathbb{R}$ et un pas d'intégration h la méthode des trapèzes consiste à calculer

$$I_h = h \sum_{i=-\infty}^{+\infty} f(hi)$$

comme valeur approchée de $\int_{-\infty}^{+\infty} f(x)\mathrm{d}x$. Il s'agit d'une extension à $\mathbb{R}$ de la méthode des trapèzes sur $[a, b]$ qui compose m fois la méthode de Newton-Cotes à 2 points (formule donnée plus haut). Découverte en 1973 par Takahasi et Mori, la transformation doublement exponentielle est couramment utilisée en logiciel. Une introduction à la transformation et à sa découverte est donnée dans [25] ; l'essentiel en est restitué ici.

Pour calculer

$$I = \int_{-1}^{1} f(x)\mathrm{d}x$$

il est possible d'utiliser une transformation

$$x = \phi(t)$$

où ϕ est analytique sur $\mathbb{R}$ et vérifie

$$\lim_{t\to-\infty} \phi(t) = -1, \lim_{t\to\infty} \phi(t) = 1$$

et alors

$$I = \int_{-\infty}^{\infty} f(\phi(t))\phi'(t)\mathrm{d}t.$$

En appliquant la formule des trapèzes à cette dernière expression on calcule

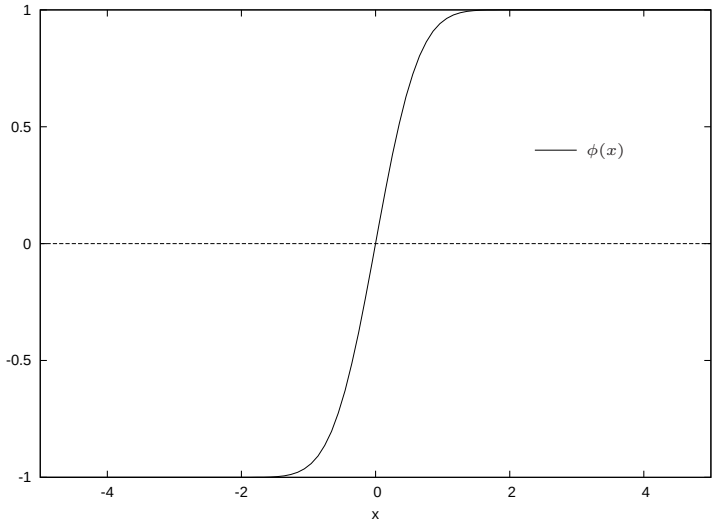

FIG. 1.1 – La transformation $\phi(x) = \tanh(\frac{\pi}{2}\sinh x)$ utilisée dans la méthode doublement exponentielle.

$$I_h^N = \sum_{k=-N}^{N} f(\phi(kh))\phi'(kh)$$

pour un certain pas de discrétisation h et en tronquant la somme aux termes de $-N$ à N. Le choix de transformation proposé est

$$\phi(t) = \tanh\left(\frac{\pi}{2}\sinh t\right)$$

qui donne la formule

$$I_h^N = \sum_{k=-N}^{N} f\left(\tanh\left(\frac{\pi}{2}\sinh kh\right)\right)\frac{\frac{\pi}{2}\cosh kh}{\cosh^2(\frac{\pi}{2}\sinh kh)}.$$

La formule doit son nom à la décroissance doublement exponentielle de

$$\phi'(t) = \frac{\frac{\pi}{2}\cosh t}{\cosh^2(\frac{\pi}{2}\sinh t)}$$

quand $|t| \to \infty$ (voir Figure 1.2). Le principe de la transformation est de concentrer l'essentiel de la contribution de la fonction à intégrer autour de 0, d'où la forte décroissance quand $|t|$ croît. Il y a un compromis à trouver dans le choix des paramètres et de la transformation ϕ utilisée : une décroissance plus rapide que doublement exponentielle diminue l'erreur de troncature mais augmente l'erreur de discrétisation.

Depuis la découverte de la transformation DE, cette méthode est appliquée seule ou avec d'autres transformations, en fonction de la nature de l'intégrande, de ses singularités et du domaine d'intégration. Un tel exemple est l'expansion sinus cardinal « sinc » :

$$f(x) \approx \sum_{k=-N}^{N} f(kh)S(k,h)(x)$$

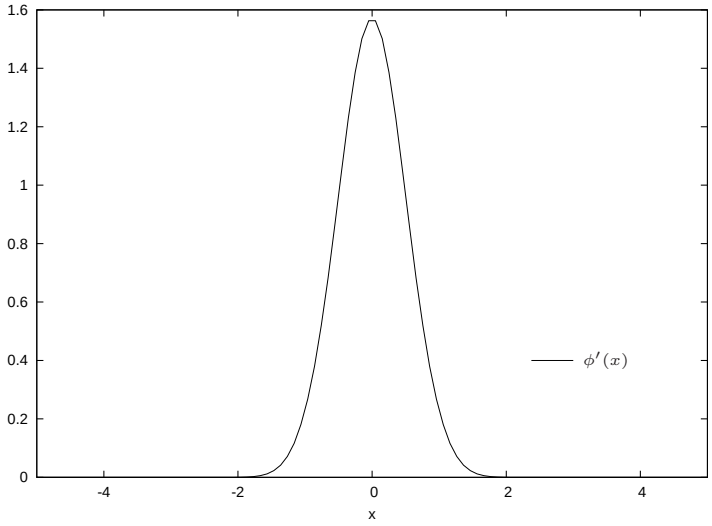

FIG. 1.2 – La décroissance de $\phi'(x)$.

où

$$S(k,h) = \frac{\sin(\pi(x-kh)/h)}{\pi(x-kh)/h}$$

qui est utilisée conjointement à la méthode doublement exponentielle dans [33], améliorant les formules précédentes qui utilisaient une transformation simplement exponentielle $\phi(x) = \tanh(x/2)$ à l'expansion « sinc ». La fonction sinc est définie par

$$\text{sinc} = \begin{cases} 1 & \text{si } x = 0, \\ \frac{\sin(\pi x)}{\pi x} & \text{sinon} \end{cases}$$

et son graphe est donné en Figure 1.3.

2.6 Stratégies adaptatives

Les méthodes d'intégration décrites jusqu'ici ne sont en général pas appliquées directement à l'intervalle d'intégration considéré. Le principe de la *composition* consiste à découper l'intervalle de départ en plusieurs sous-intervalles et appliquer une méthode d'intégration à chacun d'eux. Une stratégie d'intégration adaptative choisit à chaque étape de redécouper plus finement l'intervalle d'intégration, de modifier les paramètres de la méthode (voire de changer de méthode), jusqu'à ce qu'une condition d'arrêt soit satisfaite.

À titre d'exemple la stratégie utilisée par MuPAD est décrite dans [26] et suit les idées de [18].

3 État de l'art au niveau logiciel

Les méthodes d'intégration que nous avons présentées sont incluses dans les logiciels de calcul numérique, et jouissent d'une popularité variable. Nous évoquons principalement les logiciels qui seront testés dans le chapitre 6 « Résultats expérimentaux ». Les informations proviennent de la documentation de ces systèmes.

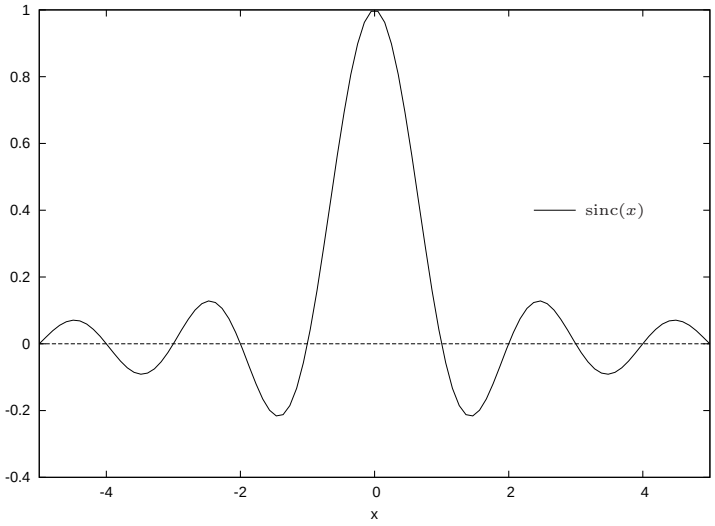

FIG. 1.3 – La fonction sinus cardinal.

Algorithme 1 Intégration adaptative – `quadrature`$(f, [a,b], Q_n, \epsilon)$

ENTRÉE : une fonction à intégrer f, un domaine d'intégration $[a,b]$, une méthode d'intégration Q_n, une tolérance d'erreur ϵ.
SORTIE : une valeur approchée $\widehat{I}$ de $\int_a^b f(x)\mathrm{d}x$ à ϵ près de façon heuristique uniquement.

$m \leftarrow (a+b)/2$
$Q_1 \leftarrow Q_n(f,a,b)$
$Q_2 \leftarrow Q_n(f,a,m) + Q_n(f,m,b)$
if $|Q_1 - Q_2| \leq \epsilon$ **then**
    return Q_2
else
    return `quadrature`$(f,[a,m],Q_n,\epsilon)$+ `quadrature`$(f,[m,b],Q_n,\epsilon)$.
end if

3.1 Maple 10

Maple utilise la bibliothèque NAG (*Numerical Algorithm Group*) pour le calcul numérique d'intégrales. Cette bibliothèque propose les méthodes de Gauss-Legendre et Gauss-Kronrod avec une stratégie adaptative, mais se limite à l'utilisation des nombres flottants disponibles en machine. La commande `evalhf(Digits)` donne le nombre de chiffres à partir duquel Maple n'utilise plus NAG (et vaut 15 sur une machine 32 bits).

En interne, Maple dispose aussi des méthodes de Clenshaw-Curtis, de Gauss-Legendre, de la méthode DE ainsi que de l'expansion sinc. La stratégie par défaut essaie d'abord la méthode de Clenshaw-Curtis. Si la vitesse de convergence est faible, Maple passe à une méthode DE adaptative, puis à une méthode de Gauss-Legendre adaptative, tout en essayant de détecter des singularités par des moyens numériques et symboliques.

Il est aussi possible de demander à utiliser la méthode de Newton-Cotes, mais avec un nombre de points fixé à 8. Sa présence est donc anecdotique.

3.2 Pari/GP 2.3.0

La transformation DE est la méthode utilisée par défaut dans Pari/GP depuis la version 2.2.9 [3]. La méthode d'intégration de Romberg est toujours disponible mais est jugée obsolète.

3.3 Mathematica 5.0.0

Les méthodes disponibles dans Mathematica sont la méthode de Gauss-Kronrod (utilisée par défaut) et la méthode DE. Mathematica utilise aussi une stratégie récursive de bissection avec une profondeur limitée à 6 par défaut[1].

3.4 MuPAD 2.5.3

MuPAD dispose des méthodes d'intégration de Gauss-Legendre, de Newton-Cotes et de Gauss-Tchebychev. La méthode utilisée par défaut est Gauss avec un nombre de points sélectionné automatiquement en fonction de la précision requise lorsqu'elle est inférieure à 200 chiffres. Au delà il est recommandé de préciser manuellement un nombre de points égal à la précision demandée.

La méthode de Gauss-Tchebychev est un cas particulier des méthodes de Gauss pour une fonction de poids $w = 1 - x^2$, elle demande donc une transformation de la fonction f à intégrer et s'adapte particulièrement aux fonctions de la forme $\frac{p(x)}{1-x^2}$ où $p(x)$ est un polynôme.

La fonction d'intégration de MuPAD utilise la stratégie adaptative décrite dans l'algorithme 1 (p. 10) sauf dans le cas de la méthode de Gauss-Tchebychev. Il est aussi possible de la désactiver.

4 Contributions et organisation du document

Les contributions de la thèse sont les études rigoureuses des méthodes de Newton-Cotes (chapitre 3) et de Gauss-Legendre (chapitre 4) et des algorithmes qui en découlent, dans le contexte de la précision arbitraire et avec l'objectif de borner l'erreur sur le résultat. Ces deux points (précision arbitraire et erreur bornée) constituent l'originalité du présent travail. Des algorithmes développés, les méthodes efficaces de calcul des points par un procédé d'isolation et de raffinement des racines réelles d'un polynôme (chapitre 5) sont apparues naturellement durant l'examen de la méthode de Gauss-Legendre. Il s'agit cependant d'un travail qui présente un intérêt pour lui-même et qui peut s'appliquer à la question de la recherche de racines polynomiale de façon plus générale.

L'utilisation d'une précision de calcul arbitrairement grande soulève des problèmes algorithmiques et numériques qui ne se posent pas en précision fixe. La précision atteinte lorsque l'on calcule en précision fixe est fatalement inférieure à la précision de travail, et l'accumulation des erreurs d'arrondi incite à ne regarder qu'un nombre de points relativement limité pour les méthodes d'intégration. Avec la précision arbitraire se repose la question du choix optimal de méthode, de nombre de points et de stratégie à appliquer en fonction de la précision voulue.

L'argument de la composition des calculs a déjà été cité pour justifier l'objectif de vouloir borner l'erreur. Cet objectif se comprend lui aussi dans le contexte de la précision arbitraire où il est possible d'augmenter la précision de calcul tant que le nombre de chiffres corrects demandés dans le résultat n'est pas atteint : c'est la base de la stratégie « en peau d'oignon » de Ziv [39].

L'étude de la méthode de Gauss-Legendre amène naturellement à s'intéresser au problème du calcul d'une racine d'un polynôme en précision arbitraire, traité dans les sections 17 et 18.

[1] D'après la documentation en ligne `http://documents.wolfram.com/mathematica/functions/NIntegrate`.

Les résultats théoriques et les algorithmes présentés s'accompagnent d'une réalisation logicielle sous forme d'une bibliothèque baptisée *Correctly Rounded Quadrature* (CRQ) que l'on peut traduire en français par « Intégration avec arrondi correct ». Le titre est optimiste, savoir borner l'erreur ne procure par nécessairement la propriété d'arrondi correct qui demande en plus de savoir détecter les résultats exactement représentables. Des mesures de performances de CRQ sont données dans le chapitre 6, ainsi que des comparaisons avec d'autres systèmes d'intégration numérique.

2

Éléments de calcul flottant

Après avoir effleuré le sujet du calcul flottant dans l'introduction, nous donnons ici des définitions précises et des notations qui seront utilisées par la suite, avant de parler des particularités de la bibliothèque MPFR [34]. Nous donnons aussi des résultats de bornes d'erreur élémentaires sur le calcul flottant qui seront utiles par la suite.

5 Modèle de calcul flottant

Nous donnons les rudiments de la représentation des nombres flottants en base $\beta = 2$, qui est la seule base qui nous intéressera par la suite. Des définitions similaires sont possibles pour une base β différente ; on pourra retrouver une description plus générale des flottants ainsi qu'un certain nombre de notions passées sous silence dans [19].

Dans la suite la précision p est un entier ≥ 2.

Définition 1 (Nombre flottant)**.** *Un nombre réel x est un nombre flottant en précision p s'il peut s'écrire $x = m \cdot 2^e$ où m, e sont des entiers et $|m| < 2^p$.*

Cette définition s'apparente à la notion d'« écriture scientifique » en base 10 où par exemple 1981 s'écrit $1,981 \cdot 10^3$: les flottants en base 2 ne sont rien d'autre qu'une suite finie de bits « décalée » d'une certaine puissance de 2.

Lorsque $2^{p-1} \leq |m| < 2^p$ on dit que le nombre flottant x (non nul) est normalisé.

Définition 2 (Exposant)**.** *Pour un nombre réel non nul x on définit* l'exposant $E(x) := 1 + \lfloor \log_2 |x| \rfloor$*, de telle sorte que $2^{E(x)-1} \leq |x| < 2^{E(x)}$.*

L'exposant d'un réel non nul x est donc l'exposant de la plus petite puissance de 2 qui est strictement supérieure à $|x|$. Par exemple

$$1024 = 2^{10} \leq 2006 < 2^{11} = 2048$$

donc $E(2006) = 11$. L'exposant dépend de la base mais pas de la précision.

Définition 3 (Ulp)**.** *Pour un nombre réel non nul x et une précision p on définit l'*ulp *ou* unit in the last place *de x par* $\mathrm{ulp}(x) := 2^{E(x)-p}$.

Pour un réel x non nul et une précision p on a toujours $2^{p-1}\mathrm{ulp}(x) \leq |x| < 2^p\mathrm{ulp}(x)$. Si x est un nombre flottant, alors $\mathrm{ulp}(x)$ est le poids du bit le moins significatif — qu'il soit nul ou pas — dans la mantisse normalisée à p bits de x. Pour tout réel x, $\mathrm{ulp}(x)$ est toujours positif par définition. On pourra considérer pour un réel x non nul $\mathrm{ulp}(x)$ pour différentes précisions p et on notera $\mathrm{ulp}_p(x)$ pour les distinguer, ou pour indiquer la précision si elle n'est pas évidente suivant le contexte. On attire l'attention du lecteur sur le fait que $\mathrm{ulp}(x)$ et $E(x)$ sont bien définis pour tout réel x non nul, et pas seulement pour x flottant.

Nous pouvons déjà remarquer que les flottants sur p bits ne contiennent qu'un sous-ensemble des rationnels, et que de plus cet ensemble n'est stable pour aucune des opérations de base $\{+, -, \times, \div\}$.

5.1 Flottants IEEE 754

La norme IEEE 754 [22] a proposé un cadre unifié de calcul flottant à une époque où chaque constructeur de matériel utilisait le sien. Elle définit 4 formats de nombres flottants résumés dans le tableau 2.1, ainsi que 4 modes d'arrondi : l'arrondi au plus proche d'un réel x est le nombre

Format	Taille totale	$e_{\min}$	$e_{\max}$	Taille de la mantisse	Taille de l'exposant
simple	32	-126	127	24	8
simple étendu	≥ 43	≤ -1022	≥ 1023	≥ 32	≥ 11
double	64	-1022	1023	53	11
double étendu	≥ 79	≤ -16382	≥ 16383	≥ 79	≥ 15

FIG. 2.1 – Les formats de flottants de la norme IEEE 754 (la plage d'exposant est donnée pour une normalisation $1 \leq |m| < 2$).

flottant le plus proche de x (s'il y a ambiguïté on choisit celui dont la mantisse est paire), l'arrondi vers $-\infty$ est le plus grand nombre flottant inférieur ou égal à x, l'arrondi vers $+\infty$ est le plus petit nombre flottant supérieur ou égal à x, et l'arrondi vers 0 correspond à l'arrondi vers $+\infty$ pour $x \leq 0$ et à l'arrondi vers $-\infty$ pour $x \geq 0$. Dans la suite on s'intéressera essentiellement à l'arrondi au plus proche, que l'on note $\circ(x)$ pour un réel x quelconque, ou $\circ_p(x)$ lorsque l'on voudra indiquer à quelle précision p on arrondit.

Dans la norme, chaque format de flottants détermine la taille dévolue dans la représentation du nombre aussi bien à la mantisse qu'à son exposant, représenté comme un entier avec un biais imposé. On notera que la plage d'exposants accessible $[e_{\min}, e_{\max}]$ ne correspond pas exactement au nombre de bits utilisés pour représenter l'exposant, certaines valeurs de l'exposant sont réservées pour représenter les quantités spéciales ($+\infty$, $-\infty$ et NaN pour *Not a Number* lorsque le résultat d'une opération n'est pas défini) ainsi que les flottants dénormalisés (que nous ne détaillons pas ici).

La condition de normalisation fait économiser un bit dans la représentation de la mantisse (le bit de poids fort que l'on sait égal à 1 n'est pas stocké explicitement).

Enfin une exigence cruciale de la norme est celle de l'arrondi correct :

Définition 4 (Arrondi correct)**.** *On dit que le résultat d'un calcul flottant respecte la règle de l'*arrondi correct *s'il est égal au flottant arrondi à la précision demandée dans le mode d'arrondi courant du même calcul effectué dans* $\mathbb{R}$*, c'est-à-dire sur des réels avec une précision infinie.*

Les opérations a op b où op $\in \{+, -, \times, \div\}$ doivent respecter cette règle, de même que l'opération racine carrée $\sqrt{\cdot}$. Cette propriété est capitale dans la détermination de bornes d'erreur, et permet par exemple d'écrire :

$$|\circ(a+b) - (a+b)| \leq \frac{1}{2}\mathrm{ulp}(\circ(a+b)). \tag{2.1}$$

On montre facilement par l'absurde que si l'inégalité (2.1) n'est pas satisfaite alors l'un des nombres flottants voisins du flottant $\circ(a+b)$ est plus proche de $a+b$, ce qui est absurde pour un arrondi au plus proche.

6 Particularités de MPFR

La bibliothèque MPFR [34] de calcul flottant s'appuie sur le norme IEEE 754 et respecte donc la règle de l'arrondi correct pour chacun des quatre modes d'arrondi. Le choix de la base est $\beta = 2$. Par rapport aux flottants « machine », MPFR étend la norme de deux façons :

- en supportant d'une part la multi-précision (aussi appelée « précision arbitraire »). Chaque nombre MPFR contient l'information de sa précision[2] $p \geq 2$, et les calculs sont arrondis à la précision de la variable résultat,
- en respectant d'autre part la règle de l'arrondi correct pour un ensemble de fonctions plus grand que les opérations de base (par exemple sin, exp, ...).

En pratique la taille des nombres MPFR n'est limitée que par la mémoire du système, et les exposants sont stockés dans un entier machine donc la plage d'exposants accessibles est très grande (on supposera par la suite qu'elle est non bornée).

Les fonctions disponibles dans MPFR sont nombreuses, on peut citer les fonctions trigonométriques (et trigonométriques inverses), l'exponentielle et le logarithme dans les bases 2, 10 et la base naturelle, la fonction ζ de Riemann, la factorielle (parmi d'autres). Certaines constantes comme π et la constante d'Euler sont aussi définies. Toutes ces fonctions et constantes sont calculées en respectant la règle de l'arrondi correct.

D'inspiration voisine, la bibliothèque CRlibm [11] a pour but d'étendre l'arrondi correct aux fonctions disponibles dans les bibliothèques mathématiques `libm` avec une implémentation rapide et prouvée, pour la double précision. L'objectif à terme est d'inclure ces fonctions dans une future révision de la norme IEEE 754.

Dans les algorithmes des chapitres 3 à 6, le modèle de calcul flottant utilisé est celui de MPFR avec l'hypothèse de la plage d'exposant non bornée. Pour que le lecteur n'ait pas à se souvenir de la précision de chaque variable, on indiquera la précision de calcul utilisée dans l'opération d'arrondi, plutôt que de la supposer implicitement indiquée dans la variable de destination. Par exemple une ligne

$$x \leftarrow \circ_p(y + z)$$

signifie que x reçoit l'arrondi exact en arrondi au plus proche sur p bits de l'addition $y + z$, et donc que dans l'implémentation la précision de la variable x aura été implicitement changée au besoin à p bits.

7 Calcul d'ulp

On suppose dans la suite que la plage d'exposant est non bornée. Cela implique notamment qu'il ne se produit ni *overflow* ni *underflow* au cours des calculs.

Lemme 1. *Si $c \neq 0$ et $x \neq 0$ alors $c \cdot \mathrm{ulp}(x) < 2 \cdot \mathrm{ulp}(cx)$.*

Preuve : Si $c < 0$ il n'y a rien à prouver. Par définition de $\mathrm{ulp}(x)$ on a :

$$2^{p-1}\mathrm{ulp}(x) \leq |x|$$

et pour $c > 0$

$$|cx| < 2^p \mathrm{ulp}(cx)$$

donc

$$c \cdot 2^{p-1}\mathrm{ulp}(x) \leq |cx| < 2^p \mathrm{ulp}(cx). \quad \square$$

Lemme 2. *Pour un réel non nul x et dans tous les modes d'arrondi on a : $\mathrm{ulp}(x) \leq \mathrm{ulp}(\circ(x))$, où $\circ(x)$ est l'arrondi de x dans le monde d'arrondi choisi, avec une plage d'exposant non bornée.*

[2] Les calculs effectués en précision 1 bit n'ont que très peu d'intérêt pratique de toute façon.

PREUVE : L'hypothèse de la plage d'exposant non bornée implique notamment qu'il ne se produit pas d'*underflow* (arrondi à zéro). On a $2^{E(x)-1} \leq |x| < 2^{E(x)}$ et $\text{ulp}(x) = 2^{E(x)-p}$. Après arrondi on obtient $2^{E(x)-1} \leq |\circ(x)| \leq 2^{E(x)}$ puisque $2^{E(x)}$ et $2^{E(x)-1}$ sont exactement représentables, et donc $\text{ulp}(\circ(x)) \geq 2^{E(x)-p} \geq \text{ulp}(x)$. □

Lemme 3. *Soit x un réel non nul et $\circ(x)$ l'arrondi au plus proche de x sur p bits. Alors $|x| \leq (1 + 2^{-p})|\circ(x)|$.*

PREUVE : Par définition de l'arrondi au plus proche on a

$$|x - \circ(x)| \leq \frac{1}{2}\text{ulp}(\circ(x)) \leq \frac{1}{2}2^{1-p}|\circ(x)|,$$

$$|x| \leq |\circ(x)| + 2^{-p}|\circ(x)|. \quad \square$$

Lemme 4. *Soient a et b deux nombres flottants non nuls du même signe et de même précision p alors dans tous les modes d'arrondi*

$$\text{ulp}(a) + \text{ulp}(b) \leq \frac{3}{2}\text{ulp}(\circ(a + b)).$$

PREUVE : Il suffit de considérer le cas où a et b sont positifs. La définition de ulp donne :

$$2^{p-1}\text{ulp}(a) \leq a < 2^p\text{ulp}(a),$$

$$2^{p-1}\text{ulp}(b) \leq b < 2^p\text{ulp}(b)$$

donc

$$2^{p-1}[\text{ulp}(a) + \text{ulp}(b)] \leq a + b < 2^p[\text{ulp}(a) + \text{ulp}(b)].$$

Si $\text{ulp}(a) = \text{ulp}(b)$ on obtient

$$2^p\text{ulp}(a) \leq a + b < 2^{p+1}\text{ulp}(a)$$

et donc $\text{ulp}(\circ(a + b)) \geq \text{ulp}(a + b) \geq 2\text{ulp}(a) = \text{ulp}(a) + \text{ulp}(b)$ (Lemme 2) ce qui prouve le résultat.

Sinon on peut supposer sans perte de généralité que $\text{ulp}(a) > \text{ulp}(b)$, c'est-à-dire $\text{ulp}(a) \geq 2 \cdot \text{ulp}(b)$. On en déduit :

$$\text{ulp}(a) + \text{ulp}(b) \leq \frac{3}{2}\text{ulp}(a),$$

et avec $\text{ulp}(\circ(a + b)) \geq \text{ulp}(a + b) \geq \text{ulp}(a)$ (Lemme 2) cela conclut la preuve. □

EXEMPLE : Prenons $p = 4$ et l'arrondi au plus proche : $a = 1,010$, $b = 0,1001$ en format binaire.
$a + b = 1,1101$, $\circ(a + b) = 1,110$,
$\text{ulp}(a) + \text{ulp}(b) = 2^{-3} + 2^{-4} = \frac{3}{2}2^{-3} = \frac{3}{2}\text{ulp}(\circ(a + b))$.

Lemme 5. *Soient a et b deux nombres réels tels que $0 < a \leq b$ et $\widehat{a}$, $\widehat{b}$ leur arrondi respectif au plus proche sur p bits, et $\widehat{b} \neq \widehat{a}$. Alors*

$$b - a \leq \frac{5}{2}\left(\widehat{b} - \widehat{a}\right)$$

PREUVE : On a

$$\begin{aligned} b-a &= \widehat{b}-\widehat{a}+(b-\widehat{b})+(\widehat{a}-a) \\ b-a &\leq \widehat{b}-\widehat{a}+|b-\widehat{b}|+|\widehat{a}-a|. \end{aligned}$$

On se ramène à montrer que

$$|b-\widehat{b}|+|a-\widehat{a}| \leq \frac{3}{2}\left(\widehat{b}-\widehat{a}\right).$$

Par définition de l'arrondi au plus proche on a toujours

$$|\widehat{b}-b| \leq \frac{1}{2}\mathrm{ulp}(\widehat{b}) \tag{2.2}$$

et

$$|\widehat{a}-a| \leq \frac{1}{2}\mathrm{ulp}(\widehat{a}). \tag{2.3}$$

De plus $b > a$ donne $\mathrm{ulp}(\widehat{b}) \geq \mathrm{ulp}(\widehat{a})$. On distingue trois cas, suivant la position respective de $\widehat{b}$ et $\widehat{a}$:

Cas $\mathrm{ulp}(\widehat{b}) = \mathrm{ulp}(\widehat{a})$ **:** Alors $\widehat{a}$ et $\widehat{b}$ s'écrivent

$$\begin{aligned} \widehat{a} &= m_a \mathrm{ulp}(\widehat{b}) \\ \widehat{b} &= m_b \mathrm{ulp}(\widehat{b}) \end{aligned}$$

avec m_a, m_b entiers tels que $2^{p-1} \leq m_a, m_b \leq 2^p - 1$ et $m_b > m_a$. On en déduit

$$\begin{aligned} \widehat{b}-\widehat{a} &= (m_b - m_a)\,\mathrm{ulp}(\widehat{b}) \\ &\geq \mathrm{ulp}(\widehat{b}) \\ &\geq |\widehat{a}-a|+|\widehat{b}-b|. \end{aligned}$$

Cas $\mathrm{ulp}(\widehat{b}) > \mathrm{ulp}(\widehat{a})$ **:** On écrit

$$\mathrm{ulp}(\widehat{b}) = 2^{\alpha}\mathrm{ulp}(\widehat{a}), \quad \alpha \geq 1$$

puis

$$\begin{aligned} \widehat{a} &= m_a \mathrm{ulp}(\widehat{a}) \\ \widehat{b} &= 2^{\alpha} m_b \mathrm{ulp}(\widehat{a}) \end{aligned}$$

avec m_a, m_b entiers tels que $2^{p-1} \leq m_a, m_b \leq 2^p - 1$. On obtient

$$\begin{aligned} \widehat{b}-\widehat{a} &= (2^{\alpha} m_b - m_a)\mathrm{ulp}(\widehat{a}) \\ &\geq (2^{p-1+\alpha} - 2^p + 1)\mathrm{ulp}(\widehat{a}). \end{aligned}$$

Si $\alpha > 1$, alors

$$\begin{aligned}\widehat{b} - \widehat{a} &\geq \left(2^{p+1} - 2^p + 1\right) \mathrm{ulp}(\widehat{a}) \\ &\geq 2^{p-1}\mathrm{ulp}(\widehat{b}) \\ &\geq \mathrm{ulp}(\widehat{b})\end{aligned}$$

et on conclut comme précédemment. Si $\alpha = 1$, alors

$$\widehat{b} - \widehat{a} \geq \mathrm{ulp}(\widehat{a})$$

et $\mathrm{ulp}(\widehat{b}) = 2\mathrm{ulp}(\widehat{a})$ donne

$$\begin{aligned}|\widehat{b} - b| &\leq \mathrm{ulp}(\widehat{a}) \\ |\widehat{a} - a| &\leq \frac{1}{2}\mathrm{ulp}(\widehat{a}) \\ |\widehat{a} - a| + |\widehat{b} - b| &\leq \frac{3}{2}\mathrm{ulp}(\widehat{a}) \\ &\leq \frac{3}{2}\left(\widehat{b} - \widehat{a}\right). \quad \square\end{aligned}$$

Exemple : Prenons $p = 3$ et l'arrondi au plus proche. On choisit $b = \frac{9}{8}$ et $a_k = \frac{13}{16} + 2^{-4-k}$. On obtient $\widehat{b} = 1$, $\widehat{a_k} = \frac{7}{8}$, $b - a_k = \frac{5}{16} - 2^{-4-k}$ et $\frac{5}{2}\left(\widehat{b} - \widehat{a_k}\right) = \frac{5}{16}$. De façon plus visuelle :

$$\begin{aligned}b &= 1,001 \\ \widehat{b} &= 1,00 \\ a &= 0,1101\underbrace{0\ldots0}_{k}1 \\ \widehat{a} &= 0,111 \\ \widehat{b} - \widehat{a} &= 0,001 \\ \frac{5}{2}\left(\widehat{b} - \widehat{a}\right) &= 0,0101 \\ b - a &= 0,0100\underbrace{1\ldots1}_{k+1}\end{aligned}$$

Cet exemple montre en particulier que le facteur $\frac{5}{2}$ dans la borne est optimal.

Lemme 6. *Pour deux nombres réels x et y, en calculant en base 2 en précision p avec arrondi au plus proche et s'il ne se produit ni* overflow *ni* underflow *on a*

$$|\circ(\circ(x) \circ (y)) - xy| \leq \frac{5}{2}\mathrm{ulp}(\circ(\circ(x) \circ (y))).$$

Preuve : soit $u = \circ(x)$, $v = \circ(y)$ et $z = \circ(uv)$. Pour le calcul de l'erreur relative sur la multiplication, les signes de x et de y n'interviennent pas. De même en base 2, multiplier x ou y par une puissance de 2 ne modifie pas la propriété à démontrer puisqu'on traite de l'erreur relative sur z. On peut donc sans perte de généralité supposer

$$1 \leq x \leq y < 2.$$

On pose $e = \frac{1}{2}\text{ulp}(1) = 2^{-p}$. Avec l'arrondi correct au plus proche on peut écrire

$$x = u(1+\theta),\ y = v(1+\theta'),\ uv = z(1+\theta'') \quad \text{où} \quad |\theta|, |\theta'|, |\theta''| \leq e.$$

On distingue deux cas :

Cas $x \leq 1+e$ **:** alors $u = \circ(x) = 1$ et la multiplication $z = \circ(uv) = uv$ est exacte. L'erreur se borne par :

$$\begin{aligned} |z - xy| &= |uv - xy| \\ &\leq |uv| \cdot |1 - (1+\theta)(1+\theta')| \\ &\leq |uv|(2^{1-p} + 2^{-2p}) \\ &\leq |z|(2^{1-p} + 2^{-2p})(1 + 2^{-p}). \end{aligned}$$

Puis $|z| \leq (2^p - 1)\text{ulp}(z)$ d'où l'on déduit $|z - xy| \leq 2 + 2^{-p} - 2^{1-2p} - 2^{-3p} \leq \frac{5}{2}$ car $p \geq 1$.

Cas $x > 1+e$ **:** alors $u = \circ(x) \geq 1 + 2e$. En effet la condition $x > 1+e$ assure que le nombre flottant $1+2e$ est plus près de x que le nombre flottant 1, donc dans tous les cas l'arrondi au plus proche de x sera supérieur ou égal au nombre flottant $1+2e$. On en déduit que

$$\begin{aligned} |u - x| &\leq e \\ &\leq \frac{eu}{1+2e} \\ |\theta| &\leq \frac{e}{1+2e}. \end{aligned}$$

Selon le même argument on a

$$|\theta'| \leq \frac{e}{1+2e}.$$

On borne ensuite l'erreur globale par :

$$\begin{aligned} |z - xy| &= \frac{1}{2}\text{ulp}(z) + |uv - xy| \\ &\leq \frac{1}{2}\text{ulp}(z) + |uv| \cdot |1 - (1+\theta)(1+\theta')| \\ &\leq \frac{1}{2}\text{ulp}(z) + |uv| \left(\left(\frac{e}{1+2e} \right)^2 + \frac{2e}{1+2e} \right) \\ &\leq \left(\frac{1}{2} + \left(\frac{1}{e} - 1 \right) \left(\left(\frac{e}{1+2e} \right)^2 + \frac{2e}{1+2e} \right) \right) \text{ulp}(z) \\ &\leq \left(\frac{1}{2} + \frac{(5e+2)(1-e)}{(1+2e)^2} \right) \text{ulp}(z). \end{aligned}$$

Une simple étude de fonction permet de vérifier que

$$\frac{(5e+2)(1-e)}{(1+2e)^2} \leq 2$$

d'où le résultat. □

3

Intégration de Newton-Cotes

Ce chapitre décrit la méthode d'intégration de Newton-Cotes et son implémentation dans CRQ. Une partie en a été publiée dans [16].

8 Présentation de la méthode

On appelle $f : [a, b] \to \mathbb{R}$ la fonction C^∞ à intégrer sur un segment fini $[a, b]$, et n le nombre de points de la méthode choisie. On note

$$I = \int_a^b f(x)\mathrm{d}x$$

la valeur exacte de l'intégrale que l'on souhaite calculer. La méthode d'intégration de Newton-Cotes utilise des points régulièrement espacés sur $[a, b]$ aussi appelés « abscisses » $x_0, x_1, \ldots, x_{n-1}$ avec

$$x_i = a + ih \text{ pour } 0 \leq i < n$$

et $h = \frac{b-a}{n-1}$. On appelle h le pas de la méthode. Les bornes a et b de l'intervalle sont incluses dans les points de la méthode : on parle de méthode *close*. Il est possible de les omettre pour obtenir ainsi une méthode ouverte. Dans la suite on s'intéresse uniquement aux méthodes closes.

Le principe de la méthode est d'interpoler, par le polynôme d'interpolation de Lagrange P, la fonction f en chacun des n points choisis et de calculer l'intégrale de P sur $[a, b]$:

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b P(x)\mathrm{d}x. \tag{3.1}$$

Dans la suite du chapitre nous montrerons comment calculer efficacement le membre droit de (3.1) et aussi dans quelle mesure l'approximation faite est valide. On appelle $I_n = \int_a^b P(x)\mathrm{d}x$ cette intégrale approchée par la méthode à n points.

Pour $i \in \{0, 1, \ldots, n-1\}$, notons $l_i = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$ et $w_i = \frac{1}{h} \int_a^b l_i(x)\mathrm{d}x$. Alors en particulier

$$P(x) = \sum_{i=0}^{n-1} f(x_i) l_i(x)$$

et

$$\begin{aligned} \int_a^b P(x)\mathrm{d}x &= \int_a^b \left(\sum_{i=0}^{n-1} f(x_i) l_i(x) \right) \mathrm{d}x \\ &= \sum_{i=0}^{n-1} \left(f(x_i) \int_a^b l_i(x)\mathrm{d}x \right) \\ &= h \sum_{i=0}^{n-1} w_i f(x_i). \end{aligned} \tag{3.2}$$

Le calcul approché de l'intégrale de f suivant la méthode de Newton-Cotes est donc ramené au calcul de n coefficients $(w_i)_{0 \leq i < n}$ qui ne dépendent pas de la fonction à intégrer (et qui ne dépendent pas non plus de l'intervalle d'intégration $[a, b]$ comme démontré plus loin). Les coefficients w_i sont aussi appelés « poids » de la méthode.

n	Nom	Formule	c_n
2	règle des trapèzes	$I_2 = h\frac{f_0+f_1}{2}$	$-\frac{1}{12}$
3	règle de Simpson	$I_3 = \frac{h}{3}(f_0 + 4f_1 + f_2)$	$-\frac{1}{90}$
4	règle $\frac{3}{8}$ de Simpson	$I_4 = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3)$	$-\frac{3}{80}$
5	règle de Boole	$I_5 = \frac{2}{45}h(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$	$-\frac{8}{945}$

TAB. 3.1 – Formules d'intégration de Newton-Cotes pour de petits n. Pour simplifier les notations, on définit $f_i = f(x_i)$.

Par exemple, retrouvons les poids de la méthode d'intégration à 4 points aussi appelée « règle $\frac{3}{8}$ de Simpson » sur $[a, b] = [0, 1]$. Les points d'intégration sont :

$$\begin{aligned} x_0 &= 0 \\ x_1 &= \frac{1}{3} \\ x_2 &= \frac{2}{3} \\ x_3 &= 1. \end{aligned}$$

Les polynômes l_i sont :

$$\begin{aligned} l_0(x) &= -\frac{9}{2}\left(x - \frac{1}{3}\right)\left(x - \frac{2}{3}\right)(x - 1) \\ l_1(x) &= \frac{27}{2}x\left(x - \frac{2}{3}\right)(x - 1) \\ l_2(x) &= -\frac{27}{2}x\left(x - \frac{1}{3}\right)(x - 1) \\ l_3(x) &= \frac{9}{2}x\left(x - \frac{1}{3}\right)\left(x - \frac{2}{3}\right) \end{aligned}$$

d'où l'on déduit les poids w_i :

$$\begin{aligned} w_0 &= \int_0^1 l_0(x)\mathrm{d}x = \frac{1}{8} \\ w_1 &= \int_0^1 l_1(x)\mathrm{d}x = \frac{3}{8} \\ w_2 &= \int_0^1 l_2(x)\mathrm{d}x = \frac{3}{8} \\ w_3 &= \int_0^1 l_3(x)\mathrm{d}x = \frac{1}{8}. \end{aligned}$$

On retrouve bien les poids indiqués dans la table 3.1.

L'erreur mathématique $E_n = I - I_n$ de la méthode d'intégration est de la forme $E_n = c_n h^{n+1} f^{(n)}(\zeta)$ pour n pair et $E_n = c_n h^{n+2} f^{(n+1)}(\zeta)$ pour n impair. Les constantes c_n sont

données dans la table 3.1 pour les premières valeurs de n. La formule sera justifiée en section 10.3.

Les algorithmes de calcul des coefficients sont décrits en section 9. L'analyse d'erreur, aussi bien l'erreur mathématique que les erreurs de calcul en nombres flottants, est donnée en section 10 pour aboutir au résultat principal du chapitre, le théorème 4.

9 Algorithmes pour le calcul des coefficients

On distingue dans la méthode d'intégration de Newton-Cotes le calcul des poids par rapport à l'intégration (3.2) elle-même, car les coefficients peuvent être pré-calculés et servir pour plusieurs intégrations différentes. Par exemple le procédé de composition découpe un intervalle d'intégration en plusieurs parties sur lesquelles la même méthode est appliquée séparément.

Dans un premier temps nous montrons que les coefficients ne dépendent pas de l'intervalle d'intégration considéré $[a, b]$. Ce résultat est valide de manière générale pour toutes les méthodes d'intégration de type interpolatoire, à savoir celle obéissant à une formule telle que (3.2). Ce résultat (classique) se montre par un changement de variable simple, mais il n'est explicité que dans le cas particulier de Newton-Cotes, pour mettre en évidence la formule utilisée

Proposition 1. *Les coefficients des méthodes de Newton-Cotes ne dépendent pas de l'intervalle d'intégration, et sont symétriques par rapport au milieu de l'intervalle.*

Preuve : Pour $i \in \{0, \dots, n-1\}$ on transforme l'expression de w_i donnée plus haut :

$$\begin{aligned}
w_i &= \frac{1}{h}\int_a^b l_i(x)\mathrm{d}x \\
&= \int_0^{n-1} l_i(a+xh)\mathrm{d}x \\
&= \int_0^{n-1} \left(\prod_{j\neq i} \frac{a+xh-x_j}{x_i-x_j}\right)\mathrm{d}x \\
&= \int_0^{n-1} \left(\prod_{j\neq i} \frac{a+xh-(a+jh)}{h(i-j)}\right)\mathrm{d}x \\
&= \int_0^{n-1} \left(\prod_{j\neq i} \frac{x-j}{i-j}\right)\mathrm{d}x \\
&= (n-1)\prod_{j<i}\frac{1}{i-j}\prod_{j>i}\frac{1}{i-j}\int_0^{n-1}\prod_{j\neq i}(x-j)\mathrm{d}x \\
&= \frac{(-1)^{n-1-i}}{i!(n-1-i)!}\int_0^{n-1}\prod_{j\neq i}(x-j)\mathrm{d}x.
\end{aligned}$$

Le changement de variable $x \mapsto n-1-x$ montre que $w_{n-1-i} = w_i$. □

9.1 Algorithme naïf

Soit $\delta = \mathrm{ppcm}(2, 3, \dots, n-1)$, $l_i^\star(x) = \delta\prod_{j\neq i}(x-j)$ et L_i la primitive de $l_i^\star$ définie par $L_i(0) = 0$. Soit $u_i = \frac{(-1)^{n-1-i}}{i!(n-1-i)!} = (-1)^{n-1-i}\frac{\binom{n-1}{i}}{(n-1)!}$.

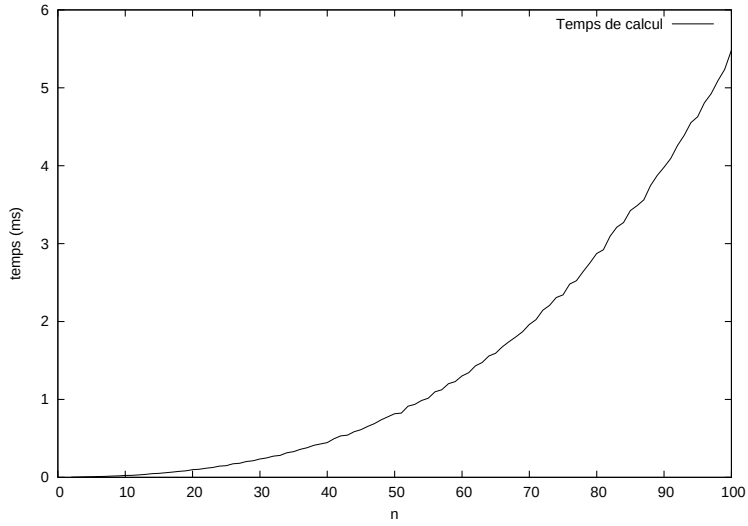

FIG. 3.1 – Temps de calcul des poids de Newton-Cotes pour de petits n avec l'algorithme naïf.

Alors on calcule les poids par

$$w_i = \frac{u_i}{\delta} L_i(n-1). \tag{3.3}$$

Cette formule montre notamment que les poids sont rationnels ; on peut donc les calculer exactement sous la forme $w_i = \frac{v_i}{d}$ par l'algorithme 2 ci-dessous. Le facteur δ dans $l_i^\star$ permet de s'assurer qu'à chaque étape $L_i(n-1)$ est un entier ; en effet $l_i^\star$ est un polynôme à coefficients entiers et on ne divise que par des entiers de 2 à $n-1$ dans l'intégration polynomiale.

L'algorithme retourne la liste des numérateurs $(v_0, v_1, \ldots, v_{\lfloor \frac{n-1}{2} \rfloor})$ ainsi que le dénominateur commun d, sans garantir que la sortie soit réduite.

Algorithme 2 Coefficients de Newton-Cotes, algorithme naïf

ENTRÉE : n nombre de points de la méthode.
SORTIE : $(v_0, v_1, \ldots, v_{\lfloor \frac{n-1}{2} \rfloor}, d)$ tels que $w_i = \frac{v_i}{d}$.
1: $\delta \leftarrow \text{ppcm}(2, 3, \ldots, n)$
2: $l_0^\star \leftarrow (x-1)(x-2)\ldots(x-(n-1))\delta$
3: **for** $i \leftarrow 0$ to $n-1$ **do**
4: $\quad L_i(n-1) \leftarrow \int_0^{n-1} l_i^\star(x)\mathrm{d}x$
5: $\quad l_{i+1}^\star \leftarrow \frac{x-i}{x-(i+1)} l_i^\star$ ▷ mise à jour en place
6: $\quad v_i \leftarrow (-1)^{n-1-i}\binom{n-1}{i} L_i(n-1)$
7: **end for**
8: **return** $(v_0, v_1, \ldots, v_{\lfloor \frac{n-1}{2} \rfloor}, d = \delta \cdot (n-1)!)$

Le résultat de *timings* effectués sur un ordinateur Pentium-4 à 3GHz avec GMP version 4.1.4 est donné en figure 3.1.

La complexité binaire de l'algorithme s'estime ainsi :

ligne 1 la taille de δ est $\mathcal{O}(n)$ [15] et on peut le calculer naïvement par n étapes de l'algorithme d'Euclide entre le plus petit commun multiple (ppcm) courant de taille $\mathcal{O}(n)$ et l'entier suivant de taille $\mathcal{O}(\log n)$. La complexité totale est donc $\mathcal{O}(n^2 \log n)$.

ligne 2 on calcule le produit polynomial étape par étape. On maintient la représentation développée, sous forme d'un tableau de coefficients, du produit des monômes déjà traités. Les coefficients du polynôme de l'étape courante sont de taille $\mathcal{O}(n \log n)$. La complexité est $\mathcal{O}(n^3 \log^2 n)$.

lignes 4 et 5 ces lignes sont exécutées n fois ; à chaque exécution on effectue $\mathcal{O}(n)$ multiplications ou divisions d'entiers de taille $\mathcal{O}(n \log n)$ par des entiers de taille $\mathcal{O}(\log n)$, d'où une complexité totale de $\mathcal{O}(n^3 \log^2 n)$.

ligne 6 la taille de l'entier $L_i(n-1)$ est $\mathcal{O}(n \log^2 n)$, et la taille de $\binom{n-1}{i}$ est $\mathcal{O}(n)$, donc cette étape coûte $\mathcal{O}(n^2 \log^2 n)$. On ne calcule pas $\binom{n-1}{i}$ à chaque étape, mais on se contente de mettre à jour sa valeur à chaque étape. Le coût de la mise à jour est donc négligeable.

La complexité finale en temps pour l'algorithme 2 est donc $\mathcal{O}(n^3 \log^2 n)$.

9.2 Algorithme rapide

En pratique, l'algorithme naïf est suffisant pour calculer des poids de Newton-Cotes jusqu'à un nombre de points de l'ordre de quelques centaines. Pour espérer dépasser ce nombre de points dans un temps raisonnable, un algorithme plus efficace est requis (on entend par là qu'on espère gagner un facteur n dans la complexité de l'algorithme naïf).

L'ingrédient essentiel de l'algorithme amélioré est d'utiliser l'évaluation polynomiale multipoints rapide.

Soit $p(x) = x(x-1)\dots(x-(n-1))$, et on rappelle $q_i(x) = \frac{p(x)}{x-i}$. Les quantités que l'on cherche à calculer sont les

$$\widehat{l_i} = \int_0^{n-1} q_i(x)\mathrm{d}x$$

pour $i \in \{0, \cdots, n-1\}$. On note $p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x$, et une division formelle donne :

$$\begin{aligned} q_i(x) &= x^{n-1} + (i + a_{n-1})x^{n-2} + (i^2 + a_{n-1}i + a_{n-2})x^{n-3} \\ &+ \cdots \\ &+ (i^{n-1} + a_{n-1}i^{n-2} + \cdots + a_2 i + a_1). \end{aligned}$$

Par intégration :

$$\begin{aligned} \widehat{l_i} &= \frac{(n-1)^n}{n} + (i + a_{n-1})\frac{(n-1)^{n-1}}{n-1} \\ &+ (i^2 + a_{n-1}i + a_{n-2})\frac{(n-1)^{n-2}}{n-2} + \cdots \\ &+ (i^{n-1} + a_{n-1}i^{n-2} + \cdots + a_2 i + a_1)(n-1). \end{aligned}$$

On a réussi à exprimer $\widehat{l_i}$ comme un polynôme en i : $\widehat{l_i} = r(i)$ avec

$$\begin{aligned} r(y) &= (n-1)y^{n-1} + [(n-1)a_{n-1} + \frac{(n-1)^2}{2}]y^{n-2} \\ &+ [(n-1)a_{n-2} + \frac{(n-1)^2}{2}a_{n-1} + \frac{(n-1)^3}{6}]y^{n-3} + \cdots \\ &+ [(n-1)a_1 + \frac{(n-1)^2}{2}a_2 + \cdots + \frac{(n-1)^{n-1}}{n-1}a_{n-1} + \frac{(n-1)^n}{n}]. \end{aligned}$$

On constate de plus que $r(x)$ se calcule comme $r(x) = p(x)s(x)/x^n$ (division tronquée) avec

$$s(x) = (n-1)x^{n-1} + \frac{(n-1)^2}{2}x^{n-2} + \cdots + \frac{(n-1)^n}{n}.$$

Algorithme 3 Coefficients de Newton-Cotes, algorithme rapide

ENTRÉE : n nombre de points de la méthode.
SORTIE : $(w_0, w_1, \ldots, w_{n-1})$ les poids de la méthode.

1: $p(x) \leftarrow x(x-1)\ldots(x-(n-1))$ ▷ arbre des produits
2: $s(x) \leftarrow (n-1)x^{n-1} + \frac{(n-1)^2}{2}x^{n-2} + \cdots + \frac{(n-1)^n}{n}$
3: $r(x) \leftarrow \mathrm{quo}(p(x)s(x), x^n)$ ▷ produit haut
4: **return** $(r(0), r(1), \ldots, r(n-1))$ ▷ évaluation multi-points

De ces formules on déduit l'algorithme 3. La formation de l'arbre des produits ainsi que l'évaluation multi-points méritent une description plus détaillée.

Arbre des produits. Pour $(a_1, a_2, \ldots, a_n)$ une liste de points d'évaluations, on définit par induction l'arbre des produits associé appelé $\mathrm{Tree}(a_1, a_2, \ldots, a_n)$ par :

- $\mathrm{Tree}(a) = \mathrm{Feuille}(X - a)$
- $\mathrm{Tree}(a_1, a_2, \ldots, a_n) = \mathrm{Nœud}\left(\mathrm{Tree}(a_1, a_2, \ldots, a_{\lfloor \frac{n}{2} \rfloor}), \prod_{i=1}^{n}(X - a_i), \mathrm{Tree}(a_{\lfloor \frac{n}{2} \rfloor + 1}, \ldots, a_n)\right)$.

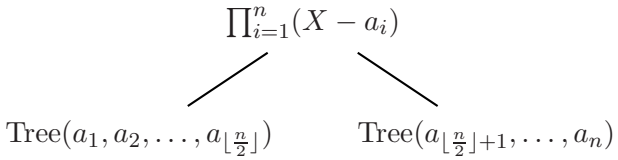

FIG. 3.2 – Définition de $\mathrm{Tree}(a_1, a_2, \ldots, a_n)$.

Le calcul de $\mathrm{Tree}(0, 1, \ldots, n-1)$ se fait naturellement par un parcours en profondeur, chaque nœud étant étiqueté par le produit des polynômes des fils gauche et droit.

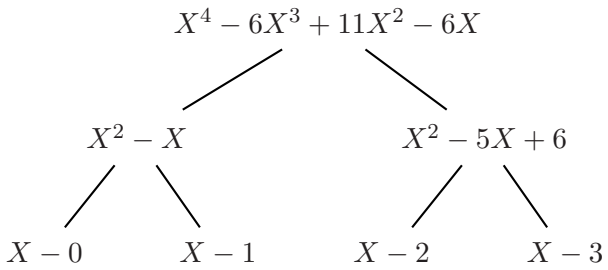

FIG. 3.3 – Calcul de l'arbre des produits $\mathrm{Tree}(0, 1, 2, 3)$.

On note dans la suite $M(d,c)$ le coût de la multiplication de deux polynômes de degré d dont les coefficients sont bornés en taille par c bits, et $M(c)$ le coût de la multiplication de deux entiers de c bits.

Calcul de $s(x)$. Dans le calcul de complexité de Tree$(0,1,\ldots,n-1)$ on majore la taille des coefficients aux feuilles par $\log_2 n$ bits, et on appelle $T(k)$ le coût de construire un sous-arbre de hauteur k, que l'on supposera plein et donc étiqueté par un polynôme unitaire de degré 2^k à coefficients entiers. Pour borner la taille des coefficients d'un tel polynôme, nous utiliserons le lemme suivant :

Lemme 7. *Soient $P(X)=\sum_{i=0}^{d} a_i X^i$ et $Q(X)=\sum_{i=0}^{d} b_i X^i$ deux polynômes unitaires de degré d à coefficients entiers vérifiant*

$$\forall i \in \{0,\ldots,d\}, |a_i|, |b_i| \leq K$$

pour une borne réelle $K>2$. Alors le produit $P(X)Q(X)=\sum_{i=0}^{2d} c_i X^i$ vérifie

$$\forall i \in \{0,\ldots,2d\}, |c_i| \leq dK^2$$

Preuve : On a la formule

$$c_j = \sum_{i=0}^{j} a_i b_{j-i}$$

pour $j \in \{0,\ldots,2d\}$. Si $j<d$, alors les termes apparaissant dans la somme sont tous bornés par K^2 et on obtient $|c_j| \leq (j+1)K^2$, qui est maximal en $j=d-1$.

Pour $j \geq d$ la somme porte sur $2d-j+1$ termes non nuls, parmi lesquels deux font apparaître a_d ou b_d qui sont égaux à 1. On a dans ce cas $|c_j| \leq (2d-j-1)K^2+2K$ qui est maximal en $j=d$.

On conclut par $dK^2 \geq (d-1)K^2+2K$. □

Pour l'analyse de complexité on s'intéresse à la taille binaire des coefficients, que l'on résume par le tableau suivant :

Étage	Degré du polynôme	Borne sur $\log_2$ des coefficients
0	1	$\log_2 n$
1	2	$2\log_2 n$
2	4	$4\log_2 n+1$
3	8	$8\log_2 n+2\cdot 1+2$
k	2^k	$2^k\log_2 n+2^{k-1}\sum_{i=0}^{k-1} i2^{-i}$

On suppose que $n=2^u$ est une puissance de 2, et que la multiplication de deux polynômes de degré d dont les coefficients sont de taille c bits au plus a un coût égal à la multiplication de deux entiers de taille dc : $M(d,c)=M(dc)$. Cette supposition est justifiée dès lors que $\log d = \mathcal{O}(c)$ car en pratique on est ramené à multiplier deux entiers de taille $d(2c+\lceil\log_2 d\rceil)$. La formule de récurrence pour $T(k)$ devient

$$\begin{cases} T(0) &= \log n \\ T(k+1) &= 2T(k)+M(2^{2k}(\log_2 n+\frac{1}{2}\sum_{i=0}^{k-1} i2^{-i})) \end{cases}$$

On rappelle la formule pour $x \neq 1$:

$$\sum_{i=0}^{j} ix^i = \frac{jx^{j+2} - (j+1)x^{j+1} + x}{(x-1)^2}$$

ce qui permet de simplifier

$$T(k+1) = 2T(k) + M(2^{2k}(\log_2 n + \mathcal{O}(1))).$$

La complexité globale pour $T(u)$ est donc

$$\begin{aligned} T(u) &= M\left(\frac{n^2}{4}\log_2 n\right) + 2M\left(\frac{n^2}{16}\log_2 n\right) + \ldots + nM\left(\log_2 n\right) \\ &= \mathcal{O}(M(n^2 \log n)) = \mathcal{O}(n^2 \log^2 n) \end{aligned}$$

en supposant une multiplication entière rapide par la FFT et en négligeant les facteurs $\mathcal{O}(\log\log n)$. Cette complexité est à comparer à la taille binaire du résultat. Notons

$$\begin{aligned} P(X) &= (X-1)(X-2)\ldots(X-n) \\ &= \sum_{i=0}^{n} a_i X^i. \end{aligned}$$

En notant $U = |\prod_{i=0}^{n} a_i|$ la taille binaire de P est $\log_2 U$. Le développement du polynôme P donne :

$$\begin{aligned} |a_{n-i}| &= \sum_{1 \leq j_1 < j2 < \ldots < j_i \leq n} j_1 j_2 \ldots j_i \\ &\geq \binom{n}{i} i! = \frac{n!}{(n-i)!} \end{aligned}$$

car le produit $j_1 j_2 \ldots j_i$ est minimal quand $j_k = k$ pour $1 \leq k \leq i$. Puis :

$$U \geq \frac{n!}{0!}\frac{n!}{1!}\cdots\frac{n!}{(n-1)!}\frac{n!}{n!}$$

qui se simplifie en

$$\begin{aligned} U &\geq \prod_{i=2}^{n} i^i \\ \log U &\geq \sum_{i=2}^{n} i \log i \\ &\geq \sum_{i=2}^{n} \int_{i-1}^{i} x \log x \mathrm{d}x \\ &\geq \int_{1}^{n} x \log x \mathrm{d}x = \frac{1}{2}n^2 \log n - \frac{n^2-1}{4}. \end{aligned}$$

Cela signifie qu'en négligeant les facteurs $\mathcal{O}(\log\log n)$ l'algorithme de calcul de l'arbre des produits ne perd qu'un facteur $\log n$ au plus par rapport à la taille de la sortie, et donc par rapport à l'optimal.

Dans le calcul de $s(x)$, on constate qu'il se produit des divisions par des entiers de 2 à n. On peut envisager plusieurs solutions :

- poursuivre les calculs en rationnel,
- multiplier par un dénominateur commun comme cela est fait dans l'algorithme naïf,
- choisir un entier f suffisamment grand et premier avec $(2, 3, \ldots, n)$, poursuivre les calculs dans $\mathbb{Z}/f\mathbb{Z}$ pour obtenir $p(x)s(x) \bmod f$, et faire une reconstruction rationnelle à la fin.

Dans CRQ, le troisième choix a été fait pour profiter des algorithmes de calculs polynomiaux et d'évaluation multi-points déjà implémentés dans GMP-ECM [38] sur des entiers. Dans la suite de l'algorithme, on calculera donc modulo f où f est de taille $\mathcal{O}(n \log n)$ et tel que toutes les divisions à effectuer sont bien définies. La taille de f sera justifiée plus loin dans l'explication du mécanisme de reconstruction rationnelle.

Le calcul direct de $s(x)$ demande le calcul de n inversions modulo f en $\mathcal{O}(M(n) \log n)$, pour un coût total en $\mathcal{O}(nM(n) \log n)$. L'algorithme 4 dû à Montgomery calcule n inverses via le calcul de $3(n-1)$ multiplications et une inversion et un coût global en $\mathcal{O}(nM(n))$.

Algorithme 4 Calcul simultané d'inverses

ENTRÉE : $x_1, x_2, \ldots, x_n$.
SORTIE : $\frac{1}{x_1}, \frac{1}{x_2}, \ldots, \frac{1}{x_n}$.

1: $m_1 \leftarrow x_1$
2: **for** $i \leftarrow 2$ to n **do**
3: $\quad m_i \leftarrow m_{i-1} x_i$
4: **end for**
5: $z \leftarrow \frac{1}{m_n}$
6: **for** $i \leftarrow n$ to 2 **do**
7: $\quad r_i \leftarrow m_{i-1} z$
8: $\quad z \leftarrow x_i z$
9: **end for**
10: $r_1 \leftarrow z$
11: **return** $(r_1, r_2, \ldots, r_n)$

Évaluation multi-points. L'évaluation multi-points réutilise l'arbre des produits en se basant sur les deux lemmes suivants :

Lemme 8. *Soit P un polynôme à coefficients dans un anneau A, et $a \in A$. Alors $P \bmod X - a = P(a)$.*

Lemme 9. *Soient P, Q, et R trois polynômes à coefficients dans un anneau A avec Q et R unitaires. Alors $P \bmod R = (P \bmod QR) \bmod R$.*

L'idée pour calculer $r(0), r(1), \ldots, r(n-1)$ est donc de calculer $r(x) \bmod p(x)$ dont les racines sont $0, 1, \ldots, n-1$ au sommet de l'arbre, et de parcourir l'arbre en profondeur en calculant à chaque fois le reste du polynôme courant avec le polynôme du nœud visité. À la fin du calcul on lit bien aux feuilles de l'arbre

$$(r(x) \bmod x - 0, r(x) \bmod x - 1, \ldots, r(x) \bmod x - (n-1)) = (r(0), r(1), \ldots, r(n-1)).$$

Pour $n = 4$ on a $r(X) = 3X^3 - \frac{27}{2}X^2 + 15X - \frac{9}{4}$, et les calculs effectués dans l'arbre sont donnés en figure 3.4.

L'analyse du coût du calcul des restes polynomiaux dans l'arbre est analogue à celle du calcul de l'arbre en lui-même, le calcul du reste d'un polynôme de degré $2d$ par un polynôme de degré d coûtant $\mathcal{O}(M(d))$. En négligeant de même les facteurs $\log \log n$ dans la complexité et en supposant une multiplication FFT on obtient un coût en $\mathcal{O}(n^2 \log^2 n)$.

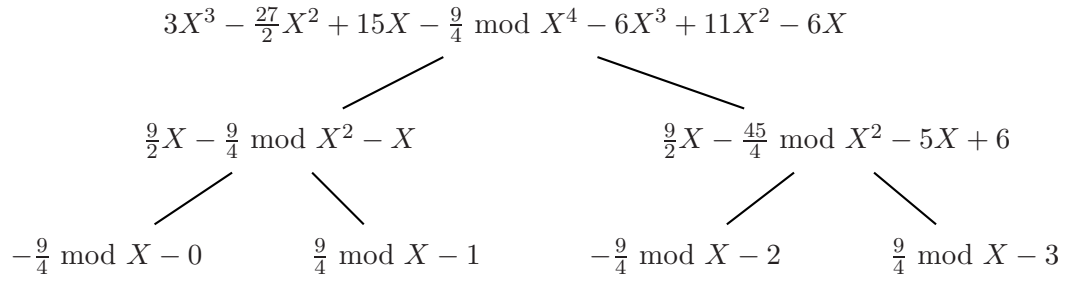

FIG. 3.4 – Évaluation multi-points rapide.

Algorithme 5 Algorithme d'Euclide étendu

ENTRÉE : $f, g \in \mathbb{N}$.
SORTIE : $r, s, t \in \mathbb{Z}$ tels que $sf + tg = r$, et $r = \text{pgcd}(f, g)$.
$r_0 \leftarrow f, s_0 \leftarrow 1, t_0 \leftarrow 0$
$r_1 \leftarrow g, s_1 \leftarrow 0, t_1 \leftarrow 1$
$i \leftarrow 1$
while $r_i \neq 0$ **do**
    $q_i \leftarrow r_{i-1}$ quo r_i
    $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
    $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
    $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
    $i \leftarrow i + 1$
end while
$l \leftarrow i - 1$
return r_l, s_l, t_l

Reconstruction rationnelle. Considérons l'équation

$$g = \frac{r}{t} \bmod f \tag{3.4}$$

où g, r, t, et f sont des entiers. Le principe de la reconstruction rationnelle est de retrouver r et t connaissant g et f. Le lemme suivant, tiré de [37, p. 123], donne la méthode à suivre :

Lemme 10. *Soient $f, g \in \mathbb{N}$ et $r, s, t \in \mathbb{Z}$ tels que $sf + tg = r$, et supposons qu'il existe $k \in \{1, \ldots, f\}$ tel que*

$$|r| < k \text{ et } 0 < t \leq \frac{f}{k}.$$

On définit j le premier indice dans l'algorithme 5 tel que $r_j < k$:

$$r_j < k \leq r_{j-1}.$$

Si j est inférieur à la valeur de l en fin d'algorithme 5 alors on pose $q = \lceil \frac{r_{j-1}-k}{r_j} \rceil$ et $q = 0$ sinon. Alors il existe un entier non nul $\alpha \in \mathbb{Z}$ tel que

$$\begin{cases} (r, s, t) & = & (\alpha r_j, \alpha s_j, \alpha t_j) \\ & \text{ou} & \\ (r, s, t) & = & (\alpha r^\star, \alpha s^\star, \alpha t^\star) \end{cases}$$

où $r^\star = r_{j-1} - qr_j$, $s^\star = s_{j-1} - qs_j$ *et* $t^\star = t_{j-1} - qt_j$.

La preuve, omise ici, est consultable dans [37]. On en déduit la méthode suivante pour résoudre l'équation (3.4) : comme

$$g = \frac{r}{t} \bmod f$$

il existe $s, t \in \mathbb{Z}$ tels que

$$sf + tg = r.$$

Supposons de plus que l'on connaisse des bornes $B_1, B_2 \in \mathbb{N}$ telles que

$$\begin{aligned} |r| &< B_1 \\ 0 < t &\leq B_2 \end{aligned}$$

et

$$B_1 B_2 \leq f.$$

Alors l'algorithme d'Euclide appliqué à g et f et le lemme 10 fournissent une à deux fractions solutions de l'équation (3.4).
Exemple : Prenons $f = 17$ et $g = 12$. L'exécution de l'algorithme 5 pour f et g donne :

i	q_i	r_i	s_i	t_i
0		17	1	0
1	1	12	0	1
2	2	5	1	-1
3	2	2	-2	3
4	2	1	5	-7
5		0	-12	17

Pour $k = 4$, on obtient $j = 3$ et $q = \lceil \frac{r_2 - 4}{r_3} \rceil = 1$. On constate que

$$\frac{r_3}{t_3} = \frac{2}{3} = 12 \bmod 17$$

donc $\frac{2}{3}$ est une solution à l'équation (3.4). De plus $r_2 - qr_3 = 5 - 2 = 3$ et $t_2 - qt_3 = -1 - 3 = -4$ et on constate de même que $-\frac{3}{4} = 12 \bmod 17$ est une solution à l'équation (3.4). On a donc deux solutions avec $B_1 = 4$, $B_2 = 4$. Or dans l'algorithme de calcul des poids de Newton-Cotes l'unicité de la solution de la reconstruction est évidemment requise. Nous avons donc besoin d'une condition plus forte :

Lemme 11. *Pour* $g \in \mathbb{Z}$, $f, B_1, B_2 \in \mathbb{N}^*$, *le système d'équations*

$$\begin{cases} g &= \frac{r}{t} \bmod f \\ |r| &< B_1 \\ 0 < t &< B_2 \end{cases}$$

admet au plus une solution réduite (r, t), $\gcd(r, t) = 1$ *si* $f \geq 2B_1 B_2$.

Preuve : Supposons qu'il existe deux solutions (r_1, t_1) et (r_2, t_2). On obtient

$$\begin{aligned} \frac{r_1}{t_1} &= \frac{r_2}{t_2} \bmod f \\ r_1 t_2 &= r_2 t_1 \bmod f \end{aligned}$$

donc $f \mid r_1t_2 - r_2t_1$. D'autre part

$$|r_1t_2 - r_2t_1| < 2B_1B_2 \leq f$$

donc

$$\begin{array}{rcl} r_1t_2 & = & r_2t_1 \\ \dfrac{r_1}{t_1} & = & \dfrac{r_2}{t_2} \end{array}$$

où la dernière égalité se situe dans $\mathbb{Q}$. □

Il reste finalement à justifier la taille $\mathcal{O}(n \log n)$ annoncée pour f. Pour cela on montre que chacun des poids w_i admet un représentant rationnel dont le numérateur et le dénominateur sont de taille $\mathcal{O}(n \log n)$. On rappelle la formule

$$w_i = \frac{(-1)^{n-1-i}}{i!(n-1-i)!} \int_0^{n-1} q_i(x)\mathrm{d}x.$$

On note

$$q_i(x) = \prod_{\substack{j=0, \\ j \neq i}}^{n-1} (x-j) = \sum_{j=0}^{n-1} b_j x^j$$

où les b_j sont des entiers. Soit $j \in \{0, \ldots, n-1\}$.

Dans le développement du produit de polynômes, le coefficient b_j est la somme de $\binom{n-1}{j}$ produits de $(n-1-j)$ entiers distincts choisis dans $\{1, \ldots, n-1\}$ donc on peut majorer chaque terme par $\frac{(n-1)!}{j!}$ et obtenir

$$|b_j| \leq \binom{n-1}{j} \frac{(n-1)!}{j!}.$$

On note comme précédemment $\delta = \mathrm{ppcm}(2, \ldots, n)$ et alors $\delta \int_0^{n-1} q_i(x)\mathrm{d}x$ est un entier. De plus :

$$\begin{array}{rcl} \left| \delta \displaystyle\int_0^{n-1} q_i(x)\mathrm{d}x \right| & = & \left| \displaystyle\sum_{j=0}^{n-1} b_j (n-1)^{j+1} \frac{\delta}{j+1} \right| \\ & \leq & \delta \displaystyle\sum_{j=0}^{n-1} \frac{(n-1)^{j+1}((n-1)!)^2}{j!(j+1)!(n-1-j)!} \\ & \leq & \delta (n-1)^n ((n-1)!)^2 \displaystyle\sum_{j=0}^{n-1} \frac{1}{j!(j+1)!(n-1-j)!}. \end{array}$$

Posons

$$h(j) = j!(j+1)!(n-1-j)!$$

alors

$$\frac{h(j+1)}{h(j)} = \frac{(j+2)(j+1)}{n-1-j}$$

et la résolution de l'équation

$$x^2 + 4x + 3 - n = 0$$

montre que

$$h(j+1) \geq h(j) \Leftrightarrow j \geq \sqrt{n+1} - 2$$

et pour $n \geq 3$ le minimum de h est atteint en $\lceil\sqrt{n+1}\rceil - 2$. On suppose donc $n \geq 3$, les valeurs des coefficients pour $n = 2$ étant de toute façon pré-calculées, et on obtient

$$\left|\delta \int_0^{n-1} q_i(x)\mathrm{d}x\right| \leq \delta \frac{(n-1)^n (n-1)!n!}{(\lceil\sqrt{n+1}\rceil - 2)!(\lceil\sqrt{n+1}\rceil - 1)!(n+1-\lceil\sqrt{n+1}\rceil)!}.$$

On en conclut que w_i admet un représentant rationnel dont le numérateur est borné par

$$B_{\text{num}} = \delta \frac{(n-1)^n (n-1)!n!}{(\lceil\sqrt{n+1}\rceil - 2)!(\lceil\sqrt{n+1}\rceil - 1)!(n+1-\lceil\sqrt{n+1}\rceil)!}$$

et le dénominateur est borné par

$$B_{\text{denum}} = \delta(n-1)!.$$

La borne $f = 2B_{\text{num}}B_{\text{denum}}$ convient pour assurer l'unicité de la solution lors de la reconstruction rationnelle, et on vérifie facilement que sa taille en bits est $\mathcal{O}(n \log n)$. La reconstruction rationnelle se calcule donc par l'algorithme d'Euclide étendu dont le coût est de $\mathcal{O}(n \log^3 n)$ pour chaque coefficient donc $\mathcal{O}(n^2 \log^3 n)$ en tout.

Pour résumer le coût global de l'algorithme rapide :

Opération	Coût
$p(x) \leftarrow x(x-1)\dots(x-(n-1))$ [arbre des produits]	$\mathcal{O}(n^2 \log^2 n)$
$s(x) \leftarrow (n-1)x^{n-1} + \frac{(n-1)^2}{2}x^{n-2} + \dots + \frac{(n-1)^n}{n} \bmod f$	$\mathcal{O}(n^2 \log^2 n)$
$r(x) \leftarrow \text{quo}(p(x)s(x), x^n) \bmod f$ [produit court haut]	$\mathcal{O}(n^2 \log^2 n)$
$(r(0), r(1), \dots, r(n-1)) \bmod f$ [évaluation multi-points]	$\mathcal{O}(n^2 \log^2 n)$
$(r(0), r(1), \dots, r(n-1)) \in \mathbb{Q}$ [reconstruction rationnelle]	$\mathcal{O}(n^2 \log^3 n)$
Total	$\mathcal{O}(n^2 \log^3 n)$

En calculant les coefficients directement au même dénominateur $\delta \cdot (n-1)!$ qui est justifié dans la section 9.1, il est possible d'éviter complètement la reconstruction rationnelle et de la remplacer par une simple multiplication : si

$$w_i = \frac{v_i}{\delta \cdot (n-1)!} \equiv g_i \bmod f$$

alors

$$v_i \equiv g_i \delta \cdot (n-1)! \bmod f$$

et la réduction $\bmod f$ donne exactement v_i dès lors que $f \geq 2B_{\text{num}}$ car la borne B_{num} calculée pour le numérateur est compatible avec le choix de $\delta \cdot (n-1)!$ comme dénominateur. Le calcul d'un numérateur avec cet algorithme ne coûte plus qu'une multiplication et une réduction modulaire de taille $\mathcal{O}(n \log n)$ soit un coût total de $\mathcal{O}(n^2 \log^2 n)$ pour cette étape. Dans la mesure où il s'agit de l'étape ayant la plus grande complexité dans l'algorithme global de calcul des coefficients, le coût total de l'algorithme rapide passe de $\mathcal{O}(n^2 \log^3 n)$ à $\mathcal{O}(n^2 \log^2 n)$. Il est intéressant de comparer cette complexité à la taille totale $S(n)$ mesurée de la sortie de cet algorithme (figure 3.5) qui semble se comporter expérimentalement comme $\mathcal{O}(n^2 \log^{3/2} n)$ pour $n \leq 700$.

Une autre approche (suggérée par Richard Brent) est possible pour l'évaluation rapide de $r(0), r(1), \dots, r(n-1)$. Plutôt que de conduire tous les calculs modulo f de taille $\mathcal{O}(n \log n)$ bits, on peut choisir $\mathcal{O}(n)$ nombres premiers supérieurs à n et de taille $\mathcal{O}(\log n)$ bits, donc

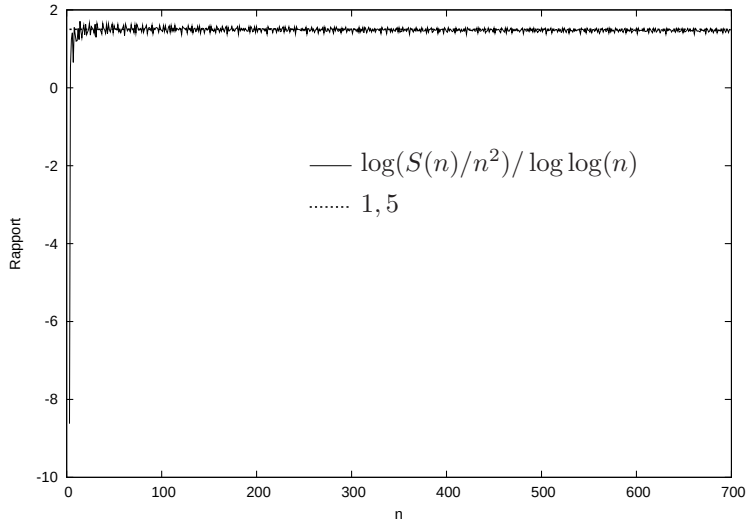

FIG. 3.5 – Le rapport $\log(S(n)/n^2)/\log\log(n)$ semble tendre vers $1,5$, où $S(n)$ est la taille des coefficients de Newton-Cotes en bits pour n points.

petits. Les étapes du calcul de l'arbre des produits de $p(x)$, de $s(x)$, $r(x)$ ainsi que l'évaluation multi-points sont alors effectuées mod $\mathcal{P}_k$ pour chacun des premiers choisis $\mathcal{P}_k$. En vérifiant que $\prod_k \mathcal{P}_k > 2B_{\text{num}}B_{\text{denum}}$ le théorème des restes chinois permet de conclure comme précédemment, en retrouvant pour chaque coefficient sa valeur modulo $\prod_k \mathcal{P}_k$.

Pour un premier $\mathcal{P}_k$ donné, le calcul de l'arbre des produits pour $p(x)$ modulo $\mathcal{P}_k$ se fait dans l'anneau des polynômes à coefficients dans le corps $\mathbb{F}_{\mathcal{P}_k}$, et les opérations arithmétiques dans ce corps coûtent $\mathcal{O}(M(\log n))$. Le coût du calcul de l'arbre modulo $\mathcal{P}_k$ est donc $\mathcal{O}(M(n)M(\log n))$ où le premier $M()$ désigne le coût de l'arithmétique polynomiale et le deuxième celui de l'arithmétique scalaire. En supposant que l'on utilise la FFT dans les deux cas on obtient $\mathcal{O}(n\log^2 n)$ pour un $\mathcal{P}_k$, soit $\mathcal{O}(n^2\log^2 n)$ en tout (en négligeant les facteurs $\log\log n$). On vérifie de même que les autres étapes de l'algorithme ne voient pas leur complexité changer lorsqu'on les effectue modulo tous les $\mathcal{P}_k$. Il reste cependant à calculer la complexité de l'algorithme des restes chinois.

Dans [37, p. 104] l'algorithme 6 est donné avec une complexité $\mathcal{O}(t^2)$ où t désigne la taille en bits du produit $\prod_k \mathcal{P}_k$. L'appliquer naïvement pour chacun des coefficients ne convient pas, on obtiendrait une complexité finale en $\mathcal{O}(n^3\log^2 n)$. Il est heureusement possible de faire mieux. Sans rentrer dans les détails que l'on retrouvera dans [37, p. 302], l'algorithme d'évaluation rapide multi-points avec l'arbre des produits polynomiaux peut d'une part se généraliser dans un anneau autre que celui des polynômes, et d'autre part servir à une interpolation polynomiale de Lagrange rapide lorsqu'on l'exécute à l'envers. L'analogue de l'interpolation polynomiale sur les entiers étant l'algorithme des restes chinois, c'est ainsi qu'on peut calculer rapidement cet algorithme. La complexité obtenue est $\mathcal{O}(M(n\log n)\log\log n)$ pour chacun des n poids à calculer, en négligeant les facteurs plus petits que doublement logarithmiques. On constate donc bien que ce choix alternatif de faire les calculs intermédiaires modulo des petits premiers ne modifie pas la complexité globale de l'algorithme rapide.

Algorithme 6 Algorithme des restes chinois naïf.

ENTRÉE : $(m_1, m_2, \ldots, m_r) \in \mathbb{Z}^r$ deux à deux premiers entre eux et $(v_1, v_2, \ldots, v_r) \in \mathbb{Z}^r$.
SORTIE : $f \in \mathbb{Z}$ tel que $f = v_i \bmod m_i$ pour $1 \leq i \leq r$.

1: $m \leftarrow m_1 m_2 \ldots m_r$
2: $r_1 \leftarrow g, s_1 \leftarrow 0, t_1 \leftarrow 1$
3: **for** $i \leftarrow 1$ to r **do**
4: $\quad u_i \leftarrow m/m_i$
5: $\quad s_i \leftarrow u_i^{-1} \bmod m_i$ ▷ par l'algorithme 5.
6: $\quad c_i \leftarrow s_i v_i \bmod m_i$
7: **end for**
8: **return** $\sum_{i=1}^{r} c_i u_i$

10 Analyse d'erreur

Lors d'un calcul d'intégration numérique par la méthode de Newton-Cotes, il y a deux sources d'erreurs à considérer : l'erreur mathématique qui provient de la méthode elle-même, et l'erreur qui provient des erreurs d'arrondi lors des calculs.

L'erreur mathématique, aussi appelé erreur de troncature, vient de la simplification du problème que l'on fait dans l'équation (3.1) en approchant l'intégrale de la fonction f par celle d'un polynôme interpolateur.

Les erreurs d'arrondi proviennent des calculs avec des nombres à précision limitée. À chaque opération même élémentaire le nombre flottant calculé n'est pas toujours égal au résultat exact de l'opération. L'écart entre la valeur calculée effectivement sur ordinateur et la valeur théorique constitue donc ce qu'on appelle globalement « erreur d'arrondi ».

On donne d'abord des bornes sur l'erreur mathématique à l'aide de preuves élémentaires. Les théorèmes 1 et 2 sont classiques mais rappelés pour référence.

10.1 Bornes sur l'erreur mathématique

Dans la suite, on dit qu'une méthode d'intégration est d'ordre n si elle intègre correctement les polynômes de degré n, et d'ordre *maximal* n si elle est d'ordre n et commet une erreur non nulle sur les polynômes de degré $n+1$.

Théorème 1. *Pour n impair, la méthode d'intégration de Newton-Cotes sur $[a, b]$ avec n points est exacte pour les polynômes de degré $\leq n$. Pour n pair, elle est exacte pour les polynômes de degré $\leq n-1$.*

PREUVE : Pour tout n, la méthode est exacte pour les polynômes de degré au plus $n-1$ car dans ce cas le polynôme d'interpolation de Lagrange sur n points est exactement f.

Supposons donc n impair. Le choix des points d'évaluation pour la méthode donne $x_i +$

$x_{n-1-i} = a + b$. Soit $g(x) = (x - x_0)(x - x_1)\dots(x - x_{n-1})$.

$$\begin{aligned} g\left(\frac{a+b}{2} - x\right) &= \prod_{i=0}^{n-1}\left(\frac{a+b}{2} - x - x_i\right) \\ &= \prod_{i=0}^{n-1}\left(\frac{a+b}{2} - x - (a+b-x_{n-1-i})\right) \\ &= \prod_{i=0}^{n-1}\left(-\frac{a+b}{2} - x + x_{n-1-i}\right) \\ &= (-1)^n \prod_{i=0}^{n-1}\left(\frac{a+b}{2} + x - x_{n-1-i}\right) \\ &= -g\left(\frac{a+b}{2} + x\right) \end{aligned}$$

et alors $\int_a^b g(x)\mathrm{d}x = 0 = \sum_{i=0}^{n-1} w_i g(x_i)$ puisque $g(x_i) = 0$. La méthode étant exacte pour les polynômes de degré au plus $n-1$, et pour g de degré n, est exacte pour tout polynôme de degré au plus n par linéarité. □

10.2 Théorème de Peano

Dans cette section on établit l'expression de l'erreur mathématique telle que donnée dans l'introduction. Le noyau de Peano d'une méthode d'intégration est un outil naturel et puissant pour y parvenir.

Pour une méthode d'intégration $I : C^{\nu+1}([a,b]) \to \mathbb{R}$ l'erreur $E : f \mapsto \int_a^b f(x)\mathrm{d}x - I(f)$ est une fonction linéaire $C^{\nu+1}([a,b]) \to \mathbb{R}$.

Nous avons le résultat suivant :

Théorème 2. *Soient*

$$K_\nu(t) = \frac{1}{\nu!} E[x \mapsto [(x-t)_+]^\nu]$$

et

$$(x-t)_+ = \begin{cases} x - t & \text{si } x > t \\ 0 & \text{sinon.} \end{cases}$$

Si $E[p] = 0$ pour tous les polynômes p de degré au plus ν alors pour $f \in C^{\nu+1}([a,b])$,

$$E[f] = \int_a^b f^{(\nu+1)}(t) K_\nu(t)\mathrm{d}t.$$

K_ν est appelé le noyau de Peano *d'ordre ν de E.*

Preuve : On écrit le développement de Taylor avec reste intégral de f en a, où p_ν désigne la

partie polynomiale du développement :

$$\begin{aligned} f(x) &= p_\nu(x) + \int_a^x \frac{1}{\nu!}(x-t)^\nu f^{(\nu+1)}(t)\mathrm{d}t \\ &= p_\nu(x) + \int_a^b \frac{1}{\nu!}[(x-t)_+]^\nu f^{(\nu+1)}(t)\mathrm{d}t, \\ E[f] &= E\left[\int_a^b \frac{1}{\nu!}[(x-t)_+]^\nu f^{(\nu+1)}(t)\mathrm{d}t\right] \\ &= \int_a^b E\left[\frac{1}{\nu!}[(x-t)_+]^\nu\right] f^{(\nu+1)}(t)\mathrm{d}t. \quad \square \end{aligned}$$

Ce théorème relie l'erreur mathématique avec le degré maximal des polynômes que la méthode intègre exactement (son ordre maximal).

Théorème 3. *Le noyau de Peano des méthodes de Newton-Cotes garde un signe constant sur* $[a, b]$.

Ce résultat est ici admis.

Dans le théorème 1 on montre que l'ordre d'une méthode de Newton-Cotes à n points est au moins $n-1$ pour n pair et n pour n impair. Si on admet que pour ces ordres les noyaux de Peano correspondants ne changent pas de signe sur $[a, b]$, et qu'ils ne sont pas identiquement nuls, alors cela signifie de plus que ces ordres sont maximaux et que l'on dispose d'une méthode pour calculer les coefficients c_n de la table 3.1 (p. 23).

Fixons le nombre de points n de la méthode de Newton-Cotes, et posons $\nu = n-1$ si n est pair, $\nu = n$ sinon. En appliquant le théorème 2 et le théorème des valeurs intermédiaires, il existe $\zeta \in]a, b[$ tel que :

$$E[f] = f^{(\nu+1)}(\zeta) \int_a^b K_\nu(t)\mathrm{d}t \tag{3.5}$$

et alors en identifiant avec

$$E_n = \begin{cases} c_n h^{n+1} f^{(n)}(\zeta) & \text{si } n \text{ est pair,} \\ c_n h^{n+2} f^{(n+1)}(\zeta) & \text{si } n \text{ est impair} \end{cases}$$

on obtient

$$c_n := \begin{cases} \frac{1}{h^{n+1}} \int_a^b K_{n-1}(t)\mathrm{d}t & \text{si } n \text{ est pair,} \\ \frac{1}{h^{n+2}} \int_a^b K_n(t)\mathrm{d}t & \text{si } n \text{ est impair.} \end{cases}$$

Par exemple pour la méthode à 3 points appelée *règle de Simpson*, on obtient

$$E[f] = \int_a^b f(x)\mathrm{d}x - \frac{b-a}{6}\left[f(a) + 4f(\frac{a+b}{2}) + f(b)\right],$$

$$K_3(t) = \frac{1}{6} E[x \mapsto (x-t)_+^3],$$

Pour $t \in [a,b]$:

$$\begin{aligned}
6K_3(t) &= \int_a^b (x-t)_+^3 \mathrm{d}x - \frac{b-a}{6}\left[(a-t)_+^3 + 4(\frac{a+b}{2}-t)_+^3 + (b-t)_+^3\right] \\
&= \int_t^b (x-t)^3 \mathrm{d}x - \frac{b-a}{6}\left[4(\frac{a+b}{2}-t)_+^3 + (b-t)^3\right] \\
&= \begin{cases} \frac{(b-t)^4}{4} - \frac{b-a}{6}\left[4(\frac{a+b}{2}-t)^3 + (b-t)^3\right] & \text{si } t < \frac{a+b}{2} \\ \frac{(b-t)^4}{4} - \frac{b-a}{6}(b-t)^3 & \text{si } t \geq \frac{a+b}{2} \end{cases}, \\
\int_a^b K_3(t)\mathrm{d}t &= \int_a^b \frac{(b-t)^4}{24} - \frac{b-a}{36}(b-t)^3 \mathrm{d}t - \int_a^{\frac{a+b}{2}} \frac{(b-a)}{9}(\frac{a+b}{2}-t)^3 \mathrm{d}t \\
&= \frac{(b-a)^5}{120} - \frac{(b-a)^5}{144} - \frac{(b-a)^5}{576} = -\frac{1}{90}\left(\frac{b-a}{2}\right)^5
\end{aligned}$$

et on retrouve la valeur $c_3 = -\frac{1}{90}$ donnée dans la table 3.1.

10.3 Borne sur les coefficients

Afin de donner une borne absolue sur l'erreur mathématique il est nécessaire de savoir borner les coefficients c_n. On détaille ici les preuves pour n pair, et on donne simplement le résultat pour n impair. Les bornes données sont des constantes, ce qui n'est pas optimal par rapport à la décroissance essentiellement polynomiale de c_n en fonction de n que l'on mesure expérimentalement, mais il s'agit du seul résultat prouvé connu.

Soit donc n pair. La méthode de Newton-Cotes à n points est exacte pour les polynômes de degré inférieur ou égal à $n-1$ (théorème 1).

Prenons pour f dans l'équation (3.5) un polynôme unitaire p de degré n ce qui nous donne :

$$E[p] = p^{(n)}(\zeta) \int_a^b K_{n-1}(t)\mathrm{d}t$$

et alors $c_n = \frac{E[p]}{n!h^{n+1}}$ car $p^{(n)}(\zeta) = n!$.

En particulier pour $p(x) = (x-x_0)(x-x_1)\dots(x-x_{n-1})$, on a

$$E[p] = \int_a^b p(x)\mathrm{d}x - \sum_{i=0}^{n-1} w_i p(x_i) = \int_a^b p(x)\mathrm{d}x$$

donc il suffit de borner $\left|\int_a^b p(x)\mathrm{d}x\right|$ pour borner c_n. On utilisera à plusieurs reprises le lemme suivant :

Lemme 12. *Pour $(u,x,v) \in \mathbb{R}^3$ tels que $u \leq x \leq v$, $|x-u||x-v| \leq \frac{(v-u)^2}{4}$.*

Preuve :

$$\begin{aligned}
|x-u||x-v| &= (x-u)(v-x) \\
&= \left(x - \frac{u+v}{2} + \frac{v-u}{2}\right)\left(\frac{v-u}{2} - (x - \frac{u+v}{2})\right) \\
&= \left(\frac{v-u}{2}\right)^2 - \left(x - \frac{u+v}{v}\right)^2 \\
&\leq \left(\frac{v-u}{2}\right)^2.
\end{aligned}$$

□

Proposition 2. *Pour $p(x) = (x - x_0)(x - x_1)\dots(x - x_{n-1})$, on a*

$$\forall x \in [a, b],\ |p(x)| \leq \frac{h^n (n-1)!}{4}.$$

PREUVE : Soit $x \in [a, b]$ tel que $p(x) \neq 0$. Alors il existe $i_0 \in \{0, \dots, n-2\}$ tels que $x \in]x_{i_0}, x_{i_0+1}[$, et alors

$$|x - x_{i_0}||x - x_{i_0+1}| \leq \frac{h^2}{4}. \quad \text{[Lemme 12]}$$

Alors

$$\begin{aligned} |p(x)| &\leq \prod_{i=0}^{n-1} |x - x_i| \\ &\leq \frac{h^2}{4} \prod_{i \neq \{i_0, i_0+1\}} |x - x_i| \\ &\leq \frac{h^2}{4} \left[\prod_{i > i_0+1} (i - i_0)h\right] \left[\prod_{j < i_0} (i_0 + 1 - j)h\right], \end{aligned}$$

$$\begin{aligned} \left[\prod_{i > i_0+1} (i - i_0)h\right] \left[\prod_{j < i_0} (i_0 + 1 - j)h\right] &\leq (n - 1 - i_0)!(i_0 + 1)!h^{n-2} \\ &\leq (n-1)!h^{n-2}, \end{aligned}$$

$$|p(x)| \leq \frac{h^n (n-1)!}{4}. \quad \square$$

D'autre part nous avons $b - a = (n - 1)h$, ce qui donne

$$|E_n[p]| = \left| \int_a^b p(x) \mathrm{d}x \right| \leq \int_a^b |p(x)| \mathrm{d}x \leq \frac{h^{n+1}(n-1)!(n-1)}{4}.$$

Avec $E_n[p] = c_n h^{n+1} n!$ il vient

$$|c_n| \leq \frac{n-1}{4n} \leq \frac{1}{4}.$$

Pour n impair on prend $p(x) = (x - x_0)(x - x_1)...(x - x_{n-1})(x - \frac{a+b}{2})$; les zéros de p sont les points d'évaluation et le milieu de l'intervalle.

Des calculs similaires montrent que

$$|p(x)| \leq \frac{h^{n+1}(n-1)!(n-1)}{8}$$

puis

$$\left| \int_a^b p(x) \mathrm{d}x \right| \leq \frac{h^{n+2}(n-1)!(n-1)^2}{8}$$

et avec $E_n = c_n h^{n+2} p^{(n+1)}$ on obtient

$$|c_n| \leq \frac{(n-1)^2}{8n(n+1)} \leq \frac{1}{8}.$$

Algorithme 7 Intégration de Newton-Cotes

ENTRÉE : $\widehat{a}$ et $\widehat{b}$ les arrondis au plus proche sur p bits des bornes a et b, f, n le nombre de points de la méthode, $\{v_i\}_{i\in\{0,n-1\}}$ et d tels que $w_i = \frac{v_i}{d}$.

SORTIE : $\widehat{I}$.

1: **for** $i \leftarrow 0$ to $n-1$ **do**
2: $\quad \widehat{t} \leftarrow \circ(\widehat{a} \cdot (n-i-1))$
3: $\quad \widehat{u} \leftarrow \circ(\widehat{b} \cdot i)$
4: $\quad \widehat{v} \leftarrow \circ(\widehat{t} + \widehat{u})$
5: $\quad \widehat{x_i} \leftarrow \circ(\widehat{v}/(n-1))$
6: $\quad \widehat{z_i} \leftarrow \circ(f(\widehat{x_i}))$
7: $\quad \widehat{y_i} \leftarrow \circ(\widehat{z_i} \cdot v_i)$
8: **end for**
9: $\widehat{S} \leftarrow \text{somme}(\widehat{y_i}, i = 0 \dots n-1)$ ▷ avec l'algorithme de Demmel et Hida [13]
10: $q \leftarrow d \cdot (n-1)$
11: $\widehat{U} \leftarrow \circ(\widehat{S}/q)$
12: $\widehat{D} \leftarrow \circ(\widehat{b} - \widehat{a})$
13: **return** $\circ(\widehat{D}\widehat{U}) = \widehat{I}$

10.4 Erreurs d'arrondi

Pour déterminer une borne d'erreur sur le résultat numérique donné par la méthode de Newton-Cotes, nous faisons une étude étape par étape de l'algorithme 7.

Souvent lors de calculs numériques cette étape est négligée, et l'analyse d'erreur s'interrompt juste après l'énoncé des bornes sur l'erreur mathématique. L'expérience montre (figure 3.6) qu'il reste encore beaucoup à faire au delà de l'erreur mathématique pour pouvoir contrôler l'erreur finale sur le résultat. En utilisant la borne sur l'erreur montrée en section 10.1 ainsi que la borne sur les coefficients c_n montrée en section 10.3 et en passant au logarithme, on constate qu'en négligeant toutes les autres erreurs le nombre de bits corrects sur le résultat est en $\mathcal{O}(n \log n)$, ce qui se constate sur la figure 3.6. On constate aussi que sur le simple exemple de la fonction exponentielle en double précision (53 bits) l'erreur mesurée reste inférieure à la borne sur l'erreur mathématique jusqu'à 18 points d'intégration environ. Avec plus de points l'erreur mathématique devient rapidement négligeable par rapport aux autres sources d'erreur et il n'est donc plus du tout licite de ne considérer que l'erreur mathématique.

Dans cette section on dénote par $\widehat{x}$ la valeur effectivement calculée (*i.e.* avec toutes les erreurs d'arrondi) pour une certaine valeur « exacte » x, qui aurait été calculée avec une précision infinie depuis le début de l'algorithme.

Dans l'algorithme 7, on appelle p la précision de calcul exprimée en nombre de bits de la mantisse, les paramètres $\widehat{a}$ et $\widehat{b}$ sont les nombres flottants arrondis au plus proche sur p bits des bornes a et b.

Pour le calcul de la borne d'erreur l'algorithme nécessite des paramètres additionnels fournis par l'utilisateur : un majorant M de $|f^{(n)}|$ sur $[a,b]$ si n est pair, ou un majorant de $|f^{(n+1)}|$ si n est impair ; m un majorant de $|f'|$ sur $[a,b]$.

Dans la suite de cette section nous allons prouver le théorème suivant :

Théorème 4. *Soit* $\delta_{\widehat{y_i}} = 6|n_i| \cdot m \cdot \text{ulp}(\widehat{x}_i) + \frac{5}{2}\text{ulp}(\widehat{y_i})$ *où* $\widehat{x}_i$ *et* $\widehat{y_i}$ *sont définis dans l'algorithme 7. Lors du calcul numérique de l'intégrale de* f *par l'algorithme 7 en précision* p *bits avec* a *et* b

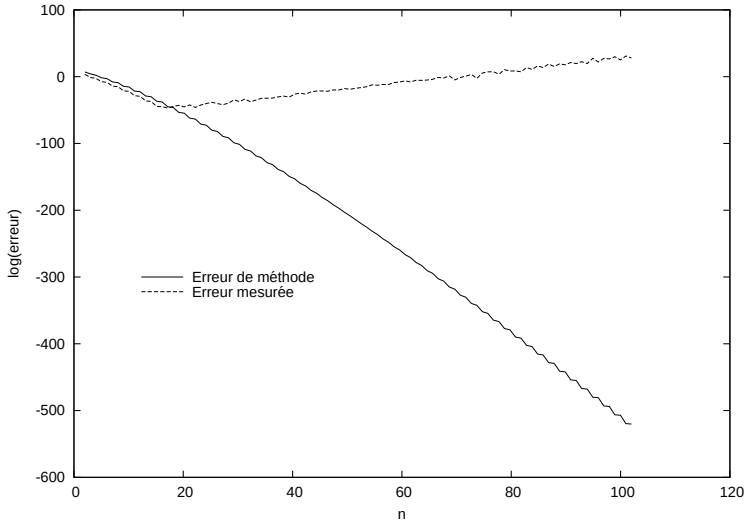

FIG. 3.6 – Un exemple de mesure d'erreur pour la méthode de Newton-Cotes appliquée à $f : x \mapsto e^x$, $[a, b] = [0, 3]$. Les calculs sont faits avec la précision par défaut de 53 bits des flottants double précision, le nombre de points n est indiqué en abscisse, le logarithme en base 2 de l'erreur est en ordonnée.

de même signe l'erreur totale sur le résultat est bornée par :

$$\begin{aligned} B_{totale} &= \left(19 + 35 \cdot 2^{-p-1}\right) \mathrm{ulp}(\widehat{I}) + \frac{1}{2}|\widehat{U}| \left(\mathrm{ulp}(\widehat{a}) + \mathrm{ulp}(\widehat{b})\right) \\ &\quad + 5\frac{(1+2^{-p})n \cdot \widehat{D}}{2d(n-1)} \max(\delta_{\widehat{y}_i}) + \begin{cases} \frac{1}{8}\left(\frac{b-a}{n-1}\right)^{n+2} M & \textit{si } n \textit{ est impair,} \\ \frac{1}{4}\left(\frac{b-a}{n-1}\right)^{n+1} M & \textit{sinon.} \end{cases} \end{aligned}$$

On peut analyser l'algorithme en plusieurs étapes :

1. Le calcul des poids w_i, $i \in [0, n-1]$ de la méthode. Pour Newton-Cotes, ces poids sont rationnels et sont calculés exactement : $w_i = \frac{n_i}{d}$ où $n_i, d \in \mathbb{Z}$, donc il n'y a pas d'arrondi à cette étape.
2. Le calcul de x_i. Il se fait de la ligne 2 à 5 de l'algorithme 7 :

$$\widehat{x}_i = \circ\left(\frac{\circ(\circ((n-1-i)\cdot\widehat{a}) + \circ(i\cdot\widehat{b}))}{n-1}\right).$$

Pour simplifier les notations on écrit $t = (n-i-1)a$, $u = i \cdot b$ et leurs contreparties inexactes $\widehat{t} = \circ((n-i-1)\widehat{a})$, $\widehat{u} = \circ(i\ \widehat{b})$. Si $b = 0$ ou $i = 0$ l'erreur sur $\widehat{u}$ est nulle. Sinon l'estimation d'erreur donne :

$$\begin{aligned} |\circ(i\cdot\widehat{b}) - ib| &\leq \tfrac{1}{2}\mathrm{ulp}(\circ(i\widehat{b})) + \tfrac{i}{2}\mathrm{ulp}(\widehat{b}) \\ &\leq \tfrac{3}{2}\mathrm{ulp}(\circ(i\widehat{b})) = \tfrac{3}{2}\mathrm{ulp}(\widehat{u}). \quad \text{[Lemmes 1 et 2]} \end{aligned}$$

De façon similaire si $a = 0$ ou $n-i-1 = 0$ alors l'erreur sur $\widehat{t}$ est nulle, et sinon on obtient $|\widehat{t} - t| \leq \frac{3}{2}\mathrm{ulp}(\widehat{t})$.

Comme a et b ont le même signe, $\widehat{t}$ et $\widehat{u}$ ont aussi le même signe et on peut appliquer le lemme 4. Cette supposition n'est pas restrictive en pratique, car il est toujours possible de découper l'intervalle d'intégration en une partie positive et une partie négative si $0 \in [a, b]$ et appliquer la borne d'erreur sur chaque partie pour en déduire une borne globale.

On suppose de plus sans perte de généralité que $0 \leq a < b$, et donc que $0 \leq \widehat{a} \leq \widehat{b}$; et alors :

$$\begin{aligned} |\widehat{v} - v| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{v}) + \tfrac{3}{2}(\mathrm{ulp}(\widehat{t}) + \mathrm{ulp}(\widehat{u})) \\ &\leq \tfrac{11}{4}\mathrm{ulp}(\widehat{v}). \quad \text{[Lemme 4]} \end{aligned}$$

En tenant compte de l'erreur provenant de la division par $n-1$ on arrive à :

$$\begin{aligned} \delta_{\widehat{x}_i} = |x_i - \widehat{x}_i| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{x}_i) + \tfrac{11}{4(n-1)}\mathrm{ulp}(\widehat{v}) \\ &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{x}_i) + \tfrac{11}{2}\mathrm{ulp}(\widehat{x}_i) \quad \text{[Lemmes 1 et 2]} \\ &\leq 6 \cdot \mathrm{ulp}(\widehat{x}_i). \end{aligned}$$

3. Le calcul de $f(x_i)$. On suppose que l'on dispose d'une implémentation de f qui calcule un résultat sur lequel l'erreur est bornée par 1 ulp. On parle parfois d'arrondi « fidèle » pour désigner cette propriété, mais l'arrondi fidèle n'étant alors pas unique nous préférons expliciter directement la propriété requise en terme de borne d'erreur. De telles implémentations avec arrondi « fidèle », et même avec arrondi correct de fonctions mathématiques en précision arbitraire sont disponibles par exemple dans MPFR [34], notamment pour des fonctions non triviales comme exp, sin, arctan et beaucoup d'autres. Une présentation rapide de MPFR est donnée dans le chapitre 6. Avec l'estimation de l'erreur sur $\widehat{x}_i$ obtenue jusqu'à présent on a :

$$|f(\widehat{x}_i) - f(x_i)| = |f'(\theta_i)(\widehat{x}_i - x_i)|, \quad \theta_i \in [\min(x_i, \widehat{x}_i), \max(x_i, \widehat{x}_i)]$$

et avec une borne sur $|f'|$ il est possible de majorer cette erreur de façon absolue. Soit $\widehat{z}_i = \circ(f(\widehat{x}_i))$ le nombre flottant calculé et $z_i = f(x_i)$ la valeur exacte. À cette étape nous avons :

$$\begin{aligned} |\widehat{z}_i - z_i| &\leq |f'(\theta_i)(\widehat{x}_i - x_i)| + \mathrm{ulp}(\widehat{z}_i) \\ &\leq 6m \cdot \mathrm{ulp}(\widehat{x}_i) + \mathrm{ulp}(\widehat{z}_i). \end{aligned}$$

4. Le calcul des $y_i = f(x_i) \cdot v_i$. L'erreur accumulée jusqu'ici est :

$$\begin{aligned} |\widehat{y}_i - y_i| &\leq |v_i| \cdot |\widehat{z}_i - z_i| + \tfrac{1}{2}\mathrm{ulp}(\widehat{y}_i) \\ &\leq 6|v_i| \cdot m \cdot \mathrm{ulp}(\widehat{x}_i) + |v_i|\mathrm{ulp}(\widehat{z}_i) + \tfrac{1}{2}\mathrm{ulp}(\widehat{y}_i) \\ &\leq 6|v_i| \cdot m \cdot \mathrm{ulp}(\widehat{x}_i) + \tfrac{5}{2}\mathrm{ulp}(\widehat{y}_i) = \delta_{\widehat{y}_i}. \quad \text{[Lemmes 1 et 2]} \end{aligned}$$

Remarque : dans la majoration de l'erreur sur $\widehat{x}_i$, $\widehat{z}_i$ ainsi que $\widehat{y}_i$, le terme en $\mathrm{ulp}(\widehat{x}_i)$ disparaît si l'erreur sur $\widehat{x}_i$ est nulle. On peut facilement montrer qu'avec notre hypothèse qu'il ne se produit pas d'*underflow*, si $\widehat{x}_i = 0$ alors l'erreur sur $\widehat{x}_i$ est nulle (i.e. $x_i = 0$) et la quantité indéfinie $\mathrm{ulp}(\widehat{x}_i)$ disparaît. Par convention on définit donc $\mathrm{ulp}(0) = 0$. Pour les besoins de la majoration de l'erreur on ne conserve que la valeur de $\max(\delta_{\widehat{y}_i})$.

5. La sommation des y_i : on la fait avec l'algorithme de Demmel et Hida [13], qui garantit une erreur d'au plus $1,5$ ulp sur le résultat final. Cet algorithme utilise une précision de travail plus grande $p' \approx p + \log_2(n)$. Soit $S = \sum_{i=0}^{n-1} y_i$.

$$|\widehat{S} - S| \leq \frac{3}{2}\mathrm{ulp}(\widehat{S}) + n \cdot \max(\delta_{\widehat{y}_i}).$$

6. La division de S par $d(n-1)$: $U = \frac{S}{d(n-1)}$. Le calcul de $d(n-1)$ est fait en arithmétique entière et est par conséquent exact. L'erreur à cette étape est donc :

$$\begin{aligned} |\widehat{U} - U| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{U}) + \tfrac{3}{2d(n-1)}\mathrm{ulp}(\widehat{S}) + \tfrac{n}{d(n-1)}\max(\delta_{\widehat{y}_i}) \\ &\leq \tfrac{7}{2}\mathrm{ulp}(\widehat{U}) + \tfrac{n}{d(n-1)}\max(\delta_{\widehat{y}_i}). \quad \text{[Lemmes 1 et 2]} \end{aligned}$$

7. La multiplication par $b-a$: $I = (b-a)U$. On note $D = b-a$ et $\widehat{D} = \circ(\widehat{b} - \widehat{a})$.

$$|\widehat{D} - D| \leq \tfrac{1}{2}\left[\mathrm{ulp}(\widehat{D}) + \mathrm{ulp}(\widehat{a}) + \mathrm{ulp}(\widehat{b})\right].$$

Dans la suite du calcul il sera nécessaire de majorer $|D|$. Pour cela nous savons que $\widehat{b} \geq \widehat{a}$, et le cas $\widehat{a} = \widehat{b}$ est passé sous silence car il n'a pas d'intérêt pratique : dans ce cas la précision p n'est même pas suffisante pour décider si $a = b$. On peut supposer $\widehat{b} > \widehat{a}$. Les conditions d'application du lemme 5 sont donc réunies :

$$\begin{aligned} |D| = D &\leq \frac{5}{2}\left(\widehat{b} - \widehat{a}\right) \\ &\leq \frac{5}{2}\left(1 + 2^{-p}\right)\widehat{D}. \quad \text{[Lemme 3]} \end{aligned} \tag{3.6}$$

Si nous utilisons tous les résultats et bornes d'erreurs obtenus jusqu'ici, il est possible d'en déduire la borne suivante pour l'erreur finale sur $\widehat{I} = \circ(\widehat{D}\widehat{U})$:

$$\begin{aligned} |\widehat{I} - I| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + |\widehat{D}\widehat{U} - D \cdot U| \\ &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + |\widehat{U}| \cdot |\widehat{D} - D| + |D| \cdot |\widehat{U} - U| \\ &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + |\widehat{U}| \cdot |\widehat{D} - D| + \tfrac{5}{2}(1 + 2^{-p})|\widehat{D}| \cdot |\widehat{U} - U| \quad \text{[Inégalité (3.6)]} \\ &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + |\widehat{U}| \cdot |\widehat{D} - D| + \\ &\quad \tfrac{5}{2}(1 + 2^{-p})|\widehat{D}|\left(\tfrac{7}{2}\mathrm{ulp}(\widehat{U}) + \tfrac{n}{d(n-1)}\max(\delta_{\widehat{y}_i})\right) \\ &\leq (18 + 35 \cdot 2^{-p-1})\mathrm{ulp}(\widehat{I}) + |\widehat{U}| \cdot |\widehat{D} - D| \\ &\quad + 5\tfrac{(1+2^{-p})n \cdot \widehat{D}}{2d(n-1)}\max(\delta_{\widehat{y}_i}) \quad \text{[Lemmes 1 et 2]} \\ &\leq (19 + 35 \cdot 2^{-p-1})\mathrm{ulp}(\widehat{I}) + \tfrac{1}{2}|\widehat{U}|\left(\mathrm{ulp}(\widehat{a}) + \mathrm{ulp}(\widehat{b})\right) \\ &\quad + 5\tfrac{(1+2^{-p})n \cdot \widehat{D}}{2d(n-1)}\max(\delta_{\widehat{y}_i}). \quad \text{[Lemmes 1 et 2]} \end{aligned}$$

Cette borne d'erreur est satisfaisante pour une utilisation dans l'algorithme d'intégration, car elle se déduit de quantités que l'on connaît soit avant l'exécution de l'algorithme (p, n), soit qui sont calculées naturellement lors de son exécution ($\widehat{I}$, $\widehat{U}$, $\widehat{b}$, $\widehat{a}$, $\widehat{D}$, d, $\delta_{\widehat{y}_i}$).

Dans la borne d'erreur finale il faut ajouter la borne sur l'erreur mathématique :

$$B_{\text{math}} \leq \begin{cases} \dfrac{1}{8}\left(\dfrac{b-a}{n-1}\right)^{n+2} M & \text{si } n \text{ est impair,} \\ \dfrac{1}{4}\left(\dfrac{b-a}{n-1}\right)^{n+1} M & \text{sinon} \end{cases} \tag{3.7}$$

qui est aussi facilement calculée. □

Lors du calcul de la borne d'erreur on choisit pour chaque opération le mode d'arrondi dirigé qui assure que la borne calculée sera plus grande que la valeur théorique. Il convient en effet d'être soigneux jusqu'au bout sans quoi l'analyse précédente perd de son intérêt.

11 Expérimentations

L'algorithme 7 a été implémenté en utilisant la bibliothèque MPFR [34]. En plus du résultat de l'intégration, le programme donne la borne d'erreur sur le résultat calculé, sous la forme de deux termes :

1. l'erreur mathématique, dont l'expression est bornée dans l'équation (3.7),
2. les erreurs d'arrondi qui sont bornées par $B_{\text{arrondi}} = B_{\text{totale}} - B_{\text{math}}$.

Pour nos expériences nous avons choisi une fonction et un domaine d'intégration pour lesquels la valeur exacte est connue. Ainsi l'erreur effectivement commise est mesurable précisément (et on la note $E_{\text{mesurée}}$). La figure 3.7 montre les différentes erreurs lors du calcul de l'intégrale $I = \int_0^3 e^x \mathrm{d}x$ avec 113 bits de précision de calcul, et avec un nombre de points d'évaluation n variant de 2 à 30.

L'erreur mathématique décroît rapidement mais il apparaît clairement qu'elle est compensée largement par les erreurs d'arrondi dès que plus de 15 points d'évaluation environ sont utilisés, pour la fonction et les paramètres considérés. Le gain théorique d'accroître l'ordre de la méthode est donc perdu. La figure 3.8 donne la plus petite valeur du nombre de points d'évaluation pour laquelle l'erreur mathématique est inférieure à l'erreur d'arrondi, pour différentes précisions de calcul. Dans un certain sens, cette valeur de n est en général interprétée comme étant optimale : pour de plus grandes valeurs de n, le gain d'une méthode d'ordre supérieur est perdu dans les erreurs d'arrondi, et pour de plus petites valeurs la précision utilisée dans les évaluations de la fonction f est plus grande que nécessaire et ralentit les calculs.

La complexité en temps de l'algorithme naïf de calcul des poids est trop importante pour considérer son utilisation au delà de quelques centaines de points. En pratique dans CRQ cet algorithme est très tôt plus lent que l'algorithme de calcul des poids de la méthode de Gauss-Legendre, qui bénéficie de plus d'une meilleure borne d'erreur mathématique et d'une meilleure stabilité numérique.

La borne sur l'erreur totale donnée par l'algorithme est plutôt proche de l'erreur mesurée. Dans les données expérimentales, on observe un rapport maximal d'environ 46000 — ce qui semble énorme, mais en échelle logarithmique cela signifie que nous ne perdons que 16 bits de précision par notre estimation. En particulier notre algorithme n'est pas trop grossièrement pessimiste.

L'instabilité numérique de la méthode lorsque n croît n'est pas surprenante, ni particulièrement nouveau. Ce fait montré ici s'explique en partie par la présence de coefficients négatifs dans la formule d'intégration dès que $n \geq 8$. En considérant que la fonction choisie pour l'expérience est particulièrement régulière, l'instabilité peut bien être attribuée à la méthode. On peut lire la figure 3.8 dans l'autre sens, et voir que la précision de calcul nécessaire pour des ordres élevés croît rapidement. De petites valeurs de n sont donc *a priori* recommandées pour la méthode d'intégration numérique de Newton-Cotes, à moins d'ajuster la précision de calcul pour compenser ce phénomène.

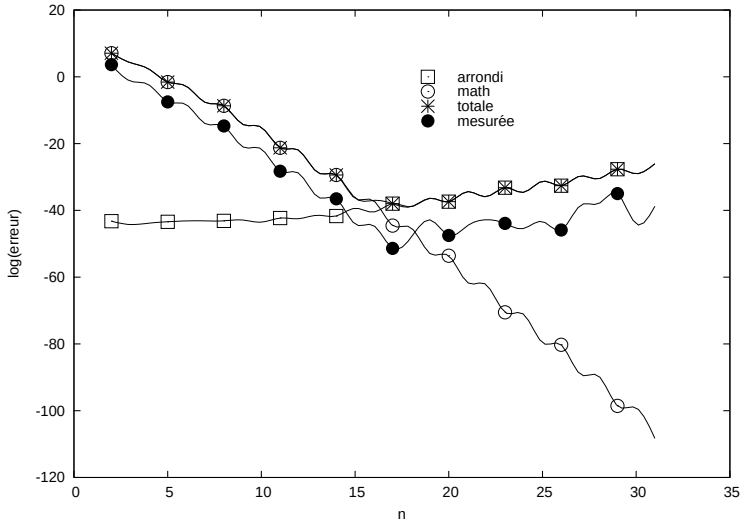

FIG. 3.7 – Les différentes bornes d'erreurs lors du calcul de $\int_0^3 e^x \mathrm{d}x$ avec 113 bits de précision. Seule la courbe « mesurée » correspond directement à une erreur plutôt qu'à une borne sur l'erreur.

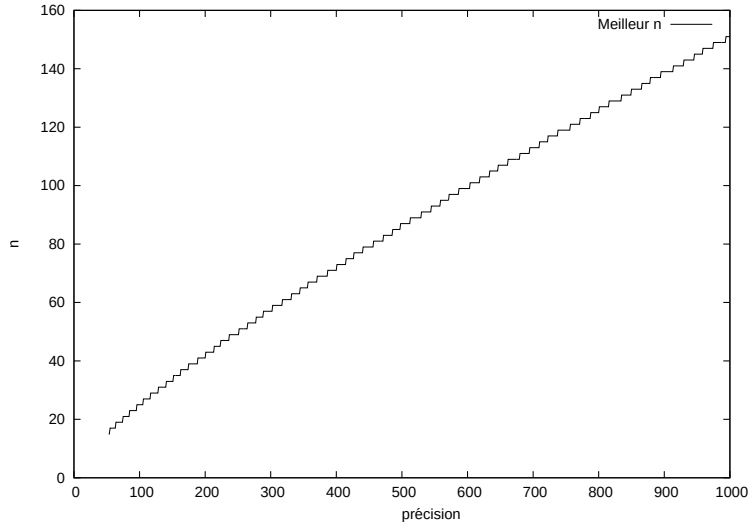

FIG. 3.8 – Les valeurs optimales du nombre n de points d'évaluation pour plusieurs précisions de calcul (données expérimentales issues de l'étude de $\int_0^3 e^x \mathrm{d}x$).

4

Intégration de Gauss-Legendre

12 Présentation de la méthode

Comme la méthode d'intégration de Newton-Cotes vue au chapitre 3, la méthode d'intégration de Gauss-Legendre est une méthode par interpolation. La différence principale réside dans le choix des abscisses.

Alors que pour Newton-Cotes les points d'évaluation sont simplement choisis équidistants dans l'intervalle d'intégration $[a, b]$, pour Gauss-Legendre ce sont les racines du n-ième polynôme d'une famille de polynômes orthogonaux, pour des raisons qui seront expliquées en section 14.1.

Plus précisément, pour deux fonctions C^∞ f et g, et une fonction positive w on définit le produit scalaire :

$$\langle f, g\rangle = \int_a^b w(x)f(x)g(x)\mathrm{d}x.$$

On appelle w la fonction de poids. Ceci définit naturellement une famille de polynômes orthogonaux $(p_i)_{i\geq 0}$ tels que

$$\forall i \in \mathbb{N}, \deg(p_i) = i$$

$$\forall (i,j) \in \mathbb{N}^2, \langle p_i, p_j\rangle = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{sinon.} \end{cases}$$

Pour $n > 0$ fixé, p_n a n racines distinctes dans $]a, b[$ que l'on appelle $x_0 < x_1 < \ldots < x_{n-1}$.

La méthode d'intégration de Gauss associée à la fonction de poids w sur $[a, b]$ est définie de façon unique comme étant la méthode par interpolation aux points d'évaluation $(x_i)_{0\leq i<n}$ telle que

$$\int_a^b w(x)p(x)\mathrm{d}x = \sum_{i=0}^{n-1} w_i p(x_i)$$

soit vrai pour tout polynôme p de degré au plus $n-1$ (cela suffit pour définir les poids w_i, même s'il sera montré que la méthode intègre correctement des polynômes jusqu'au degré $2n-1$ inclus).

La méthode d'intégration de Gauss-Legendre est la méthode de Gauss pour la fonction de poids $w(x) = 1$. Les polynômes de Legendre $(P_n)_{n\geq 0}$ sont naturellement associés à cette méthode : la famille (P_n) est orthogonale pour le produit scalaire défini par la fonction de poids $w(x) = 1$ sur l'intervalle $[-1, 1]$ et normalisée par $\forall n, P_n(1) = 1$. Il suffit d'appliquer une translation pour en déduire la famille des polynômes orthogonaux pour le poids $w(x) = 1$ sur tout intervalle de définition $[a, b]$. Dans la suite on manipulera de préférence les polynômes de Legendre (P_n) (*i. e.* sur $[-1, 1]$) de façon à pouvoir utiliser directement certains résultats connus.

Dans la section 13 sont décrits les algorithmes utilisés pour calculer les polynômes de Legendre, les points d'évaluation et les coefficients de la méthode. Une analyse de l'erreur mathématique est donnée ensuite, et dans la section 14 l'algorithme lui-même est analysé ainsi que son erreur d'arrondi.

13 Algorithmes

Dans la suite P_n est le polynôme de Legendre de degré n défini sur $[-1, 1]$. La méthode d'intégration sur $[a, b]$ est dérivée de la méthode sur $[-1, 1]$ par un décalage et une homothétie sur le polynôme. Si on appelle (V_n) la famille de polynômes définis sur $[a, b]$ nous avons les formules simples

$$V_n(u) = P_n\left(\frac{2u - (b+a)}{b-a}\right) \quad \text{et} \quad P_n(x) = V_n\left(\frac{a+b+x(b-a)}{2}\right).$$

Lors de la translation du domaine de base $[-1,1]$ au domaine de calcul $[a,b]$ les abscisses et les poids subissent une transformation de même nature, dont il sera tenu compte dans le calcul de l'erreur.

13.1 Polynômes de Legendre

Les familles de polynômes orthonormaux pour un certain poids suivent une relation de récurrence déduite directement du procédé d'orthonormalisation de Gram-Schmidt. En tenant compte de la normalisation particulière $P_n(1) = 1$ on obtient pour la famille $(P_n)_{n\geq 0}$:

$$\left\{\begin{array}{rcl} P_0(x) & = & 1 \\ P_1(x) & = & x \\ (n+1)P_{n+1}(x) & = & (2n+1)xP_n(x) - nP_{n-1}(x). \end{array}\right. \tag{4.1}$$

On en déduit que $\deg(P_n) = n$, que P_n n'a que des monômes dont le degré est de même parité que n (par exemple $P_1(x) = x$ n'a que des monômes impairs) et que P_n a des coefficients rationnels. Les polynômes $(P_0, P_1, \ldots, P_{n-1})$ forment donc une base de l'ensemble $\mathcal{P}_{n-1}$ des polynômes à coefficients réels de degré $\leq n-1$ et par conséquent, P_n est orthogonal à $\mathcal{P}_{n-1}$.

On peut être plus précis en utilisant la représentation de Rodrigues :

Théorème 5. *Pour $n \geq 0$,*

$$P_n = \frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left((x^2-1)^n\right).$$

Preuve : On appelle (R_n) la famille de polynômes définie par cette formule, et on montre qu'elle satisfait aux mêmes conditions que la famille (P_n), donc que ces deux familles coïncident. Il est clair que

$$\deg(R_n) = \deg(P_n) = n.$$

Pour calculer $R_n(1)$, remarquons que

$$R_n(x) = \frac{1}{n!2^n}\frac{\mathrm{d}^n}{\mathrm{d}x^n}((x-1)^n(x+1)^n) \tag{4.2}$$

et en appliquant la formule de Leibniz on a

$$\begin{array}{rcl} R_n(x) & = & \dfrac{1}{n!2^n}\displaystyle\sum_{i=0}^{n}\binom{n}{i}\frac{\mathrm{d}^i}{\mathrm{d}x^i}((x-1)^n)\frac{\mathrm{d}^{n-i}}{\mathrm{d}x^{n-i}}((x+1)^n) \\ & = & \dfrac{1}{2^n}\displaystyle\sum_{i=0}^{n}\binom{n}{i}^2(x-1)^{n-i}(x+1)^i \end{array} \tag{4.3}$$

d'où l'on déduit $R_n(1) = 1$. Il reste à montrer que R_n est orthogonal à $\mathcal{P}_{n-1}$, pour cela on montre que

$$s_{n,k,m} = \int_{-1}^{1} x^k \frac{\mathrm{d}^n}{\mathrm{d}x^n}((x^2-1)^m)\mathrm{d}x$$

est nul pour $0 \leq k < n \leq m$, par récurrence sur n. Pour $n = 0$ il n'y a rien à montrer et pour $n = 1$ on vérifie bien que

$$\int_{-1}^{1}\frac{\mathrm{d}}{\mathrm{d}x}((x^2-1)^m)\mathrm{d}x = \left[(x^2-1)^m\right]_{-1}^{1} = 0.$$

Soit $n > k \geq 1$. En intégrant par parties on obtient

$$\begin{aligned} s_{n,k} &= \int_{-1}^{1} x^k \frac{\mathrm{d}^n}{\mathrm{d}x^n}((x^2-1)^m)\mathrm{d}x \\ &= \left[x^k \frac{\mathrm{d}^{n-1}}{\mathrm{d}x^{n-1}}((x^2-1)^m)\right]_{-1}^{1} - k\int_{-1}^{1} x^{k-1}\frac{\mathrm{d}^{n-1}}{\mathrm{d}x^{n-1}}((x^2-1)^m)\mathrm{d}x. \end{aligned}$$

On constate que $\left[x^k \frac{\mathrm{d}^{n-1}}{\mathrm{d}x^{n-1}}((x^2-1)^m)\right]_{-1}^{1} = 0$ en appliquant la formule de Leibniz. Pour $k = 0$ on obtient $s_{n,0,m} = 0$ en intégrant directement. Puis

$$s_{n,k,m} = -k s_{n-1,k-1,m} = 0$$

par hypothèse de récurrence. En particulier $s_{n,k,n} = 0$ pour tout $0 \leq k < n$ et R_n est orthogonal à $\mathcal{P}_{n-1}$, donc la famille (R_k) est bien orthogonale pour le produit scalaire défini par la fonction de poids $w(x) = 1$ sur $[-1, 1]$. □

L'équation (4.3) montre en particulier que le dénominateur commun des coefficients de P_n divise 2^n : P_n peut s'écrire

$$P_n(X) = \begin{cases} 2^{-n}Q_n(X^2) & \text{si } n \text{ est pair} \\ 2^{-n}XQ_n(X^2) & \text{sinon} \end{cases}$$

où Q_n est à coefficients entiers. Le problème du calcul de P_n est donc ramené au calcul de Q_n, dont la procédure est décrite dans l'algorithme 8. La formule en ligne 5 se trouve en exprimant P_n en fonction de Q_n dans la formule (4.1), suivant la parité de n. L'algorithme donnée est celui du déroulement naïf de la formule de récurrence et sa complexité en nombre d'opérations arithmétiques est quadratique (on calcule en effet chacun des polynômes $Q_0, Q_1, \ldots$ jusqu'à Q_n. Il est possible d'obtenir une complexité arithmétique en $\mathcal{O}(M(n)\log n)$ avec la technique des « pas de bébé, pas de géant ».

Algorithme 8 Calcul des polynômes de Legendre

ENTRÉE : $n \geq 2$.
SORTIE : Q_n tel que $P_n(X) = 2^{-n}Q_n(X^2)$ si n est pair, et $P_n(X) = 2^{-n}XQ_n(X^2)$ sinon.

1: $Q_0 \leftarrow 1$
2: $Q_1 \leftarrow 2$
3: $p \leftarrow 0$ ▷ la parité du polynôme calculé
4: **for** $i \leftarrow 2$ to n **do**
5: $\quad Q_p \leftarrow -4(i-1)Q_p + 2(2i-1)X^{1-p}Q_{1-p}$
6: $\quad Q_p \leftarrow \frac{1}{i}Q_p$ ▷ division exacte des coefficients entiers
7: $\quad p \leftarrow 1 - p$
8: **end for**
9: **return** Q_{1-p}

13.2 Calcul des poids

Les poids $(w_i)_{0 \leq i < n}$ de la méthode de Gauss-Legendre vérifient

$$\int_{-1}^{1} q(x)\mathrm{d}x = \sum_{i=0}^{n-1} w_i q(x_i)$$

pour tout polynôme q de degré $\leq 2n-1$, où les $(x_i)_{0\leq i<n}$ sont les racines ordonnées de P_n (cf section 14.1).
Pour $i \in [0, n-1]$ on écrit $L_i(x) = \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}$ et $P_n(x) = k_n(x-x_0)(x-x_1)\dots(x-x_{n-1})$. Le théorème 5 donne $k_n = \frac{1}{2^n}\binom{2n}{n}$.

On vérifie facilement que $L_i(x) = \frac{P_n(x)}{(x-x_i)P_n'(x_i)}$ en constatant que les deux polynômes sont de même degré $n-1$ et coïncident sur les n valeurs $(x_i)_{0\leq i<n}$. L_i' a un degré $n-2$ donc $< L_i', P_n >= 0$.

$$0 = \int_{-1}^{1} P_n(x)L_i'(x)\mathrm{d}x = [P_n(x)L_i(x)]_{-1}^{1} - \int_{-1}^{1} P_n'(x)L_i(x)\mathrm{d}x.$$

$P_n'L_i$ a un degré $2n-2$ donc il est intégré de façon exacte par la méthode (théorème 6) :

$$0 = \frac{P_n^2(1)}{(1-x_i)P_n'(x_i)} - \frac{P_n^2(-1)}{(-1-x_i)P_n'(x_i)} - \sum_{j=0}^{n-1} w_j P_n'(x_j)L_i(x_j).$$

De l'équation (4.1) on voit que $|P_n(\pm 1)| = 1$. De plus $L_i(x_j) = \delta_{i,j}$ (où δ représente la fonction delta de Kronecker) donc

$$0 = \frac{2}{(1-x_i^2)P_n'(x_i)} - w_i P_n'(x_i),$$

$$w_i = \frac{2}{(1-x_i^2)P_n'^2(x_i)}. \tag{4.4}$$

Dans la section 18 (p. 65) on montre comment calculer x_i à une précision arbitraire. Puisque P_n' est connu de façon exacte on peut évaluer $P_n'(x_i)$ avec une borne d'erreur dynamique (aussi appelée *running error* dans [21]), et à partir de là calculer w_i en précision arbitraire.

14 Bornes d'erreur

14.1 Erreur mathématique

Dans cette partie on prouve la borne d'erreur mathématique de la méthode d'intégration de Gauss-Legendre. Pour une preuve plus générale concernant les méthodes de Gauss pour toute fonction de poids w on pourra consulter [12]. Cette preuve est rappelée ici dans le cas particulier de Gauss-Legendre.

Théorème 6. *La méthode de Gauss-Legendre sur $[a, b]$ avec n points est exacte pour tout polynôme de degré $\leq 2n-1$.*

Preuve : par définition la méthode de Gauss-Legendre étant de nature interpolatoire est exacte pour les polynômes de degré $\leq n-1$. Soit f un polynôme de degré $\leq 2n-1$. On écrit la division euclidienne de f par le n-ième polynôme de Legendre P_n :

$$f = q \cdot P_n + r, \quad \text{avec } \deg(q) \leq n-1, \deg(r) \leq n-1.$$

Comme P_n est orthogonal à l'ensemble $\mathcal{P}_{n-1}$ des polynômes de degré $\leq n-1$ on a

$$\int_{-1}^{1} q(x)P_n(x)\mathrm{d}x = 0 \tag{4.5}$$

et

$$\begin{aligned}
\sum_{i=0}^{n-1} w_i f(x_i) &= \sum_{i=0}^{n-1} w_i \left(q(x_i)P_n(x_i) + r(x_i)\right) \\
&= \sum_{i=0}^{n-1} w_i r(x_i) \\
&= \int_{-1}^{1} r(x)\mathrm{d}x \qquad (4.6) \\
&= \int_{-1}^{1} f(x)\mathrm{d}x \qquad (4.7)
\end{aligned}$$

où l'égalité (4.6) vient de ce que la méthode est exacte pour r de degré $n-1$, et l'égalité (4.7) vient de (4.5). Le polynôme f est donc intégré exactement par la méthode. □

Soit $E[f] = \int_{-1}^{1} f(x)\mathrm{d}x - \sum_{i=0}^{n-1} w_i f(x_i)$ l'erreur de la méthode pour la fonction f. Soit h le polynôme de degré $\leq 2n-1$ tel que

$$\forall i \in [0, n-1], f(x_i) = h(x_i) \quad \text{et} \quad f'(x_i) = h'(x_i).$$

Alors le terme reste de l'interpolation de Hermite s'écrit

$$f(x) = h(x) + \frac{f^{(2n)}(\mu(x))}{(2n)!}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2$$

pour $-1 \leq x \leq 1$ et $-1 < \mu(x) < 1$. Du théorème 6, $E[h] = 0$ et donc

$$\begin{aligned}
E[f] &= E\left[\frac{f^{(2n)}(\mu(x))}{(2n!)}(x-x_0)^2(x-x_1)^2\ldots(x-x_{n-1})^2\right] \\
&= \int_{-1}^{1} \frac{f^{(2n)}(\mu(x))}{(2n)!}\frac{P_n^2(x)}{k_n^2}\mathrm{d}x
\end{aligned}$$

où $k_n = \frac{1}{2^n}\binom{2n}{n}$ désigne toujours le coefficient dominant de P_n. Par le théorème des valeurs intermédiaires il existe $\zeta \in]-1,1[$ tel que

$$E[f] = \frac{f^{(2n)}(\zeta)}{k_n^2(2n!)}\int_{-1}^{1} P_n^2(x)\mathrm{d}x.$$

En multipliant l'équation (4.1) par $P_{n-1}(x)$ et en intégrant sur $[-1,1]$ on obtient

$$(2n+1)\int_{-1}^{1} xP_{n-1}(x)P_n(x)\mathrm{d}x = n\int_{-1}^{1} P_{n-1}^2(x)\mathrm{d}x.$$

Soit $t_n = \int_{-1}^{1} P_n^2(x)\mathrm{d}x$. La division euclidienne de $xP_{n-1}(x)$ par P_n donne

$$xP_{n-1}(x) = \frac{k_{n-1}}{k_n}P_n(x) + R(x) \quad \text{où } \deg(R) < n.$$

Donc

$$\begin{aligned}\frac{(2n+1)k_{n-1}}{k_n}\int_{-1}^{1}P_n^2(x)\mathrm{d}x &= n\int_{-1}^{1}P_{n-1}^2(x)\mathrm{d}x\\ \frac{(2n+1)k_{n-1}}{k_n}t_n &= nt_{n-1}\\ t_n &= \left(\frac{n}{2n+1}\right)\frac{k_n}{k_{n-1}}t_{n-1}.\end{aligned}$$

Avec $k_n = \frac{1}{2^n}\binom{2n}{n}$ il vient

$$\begin{aligned}t_n &= \frac{2n-1}{2n+1}t_{n-1},\\ t_n &= \frac{2}{2n+1}.\end{aligned}$$

En conclusion le terme d'erreur est

$$E[f] = \frac{2^{2n+1}(n!)^4}{(2n)!^3(2n+1)}f^{(2n)}(\zeta).$$

En tenant compte de la translation de $[-1,1]$ à $[a,b]$ on a :

$$V_n(u) = P_n\left(\frac{2u-(b+a)}{b-a}\right)$$

donc le coefficient dominant de V_n est $k_n\left(\frac{2}{b-a}\right)^n$. De plus le changement de variable dans l'intégrale donne

$$\int_a^b V_n(u)^2\mathrm{d}u = \frac{2}{b-a}\int_{-1}^{1}P_n(x)^2\mathrm{d}x$$

et il faut tenir compte de ces ajustements dans le calcul de la borne d'erreur de la méthode d'intégration sur $[a,b]$.

Théorème 7. *Soit M une borne de $|f^{(2n)}|$ sur $[a,b]$, alors l'erreur de l'intégration de Gauss-Legendre à n points de f sur $[a,b]$ en précision infinie est bornée par*

$$\frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}M.$$

Corollaire 1. *Lorsque $n \to +\infty$, la borne du Théorème 7 est équivalente à*

$$\frac{1}{2\sqrt{\pi n}}\left(\frac{e}{8n}\right)^{2n}\frac{(b-a)^{2n+1}}{2n+1}M.$$

Preuve : Cet équivalent découle de l'application de la formule de Stirling :

$$n! \sim \left(\frac{n}{e}\right)^n\sqrt{2\pi n}.$$

□

Algorithme 9 Intégration de Gauss-Legendre

ENTRÉES : $\widehat{a}, \widehat{b-a}, (\widehat{w_i}), f, (\widehat{v_i}), n$ ▷ où les (w_i) sont les poids et les (v_i) sont définis dans §14.2.

SORTIE : $\widehat{I}$, une valeur approchée sur p bits de $\int_a^b f(x)\mathrm{d}x$ dont l'erreur est bornée par le théorème 8.

HYPOTHÈSE : les calculs se font en base 2 avec une précision de p bits dans le mode d'arrondi au plus proche, les quantités $\widehat{x}$ dénotent l'arrondi au plus proche de la quantité exacte correspondante x.

1: **for** $i \leftarrow 0$ to $n-1$ **do**
2: $\quad t \leftarrow \circ((\widehat{b-a}) \cdot \widehat{v_i})$
3: $\quad \widehat{x_i} \leftarrow \circ(t + \widehat{a})$
4: $\quad \widehat{f_i} \leftarrow \circ(f(\widehat{x_i}))$
5: $\quad \widehat{y_i} \leftarrow \circ(\widehat{f_i} \cdot \widehat{w_i})$
6: **end for**
7: $\widehat{S} \leftarrow \mathrm{sum}(\widehat{y_i}, i = 0 \dots n-1)$ ▷ avec l'algorithme de Demmel et Hida [13]
8: $\widehat{D} \leftarrow \circ(\widehat{b-a})/2$
9: **return** $\circ(\widehat{D}\widehat{S}) = \widehat{I}$

14.2 Erreurs d'arrondi

Pour l'analyse d'erreur de l'algorithme 9, on reprend les définitions vues dans la section 7. Certains lemmes qui y ont été démontrés seront utiles dans la suite.

On note $\widehat{x}$ la valeur effectivement calculée pour une certaine valeur « exacte » x (telle qu'elle serait calculée si l'on disposait d'une précision infinie).

Pour être capable de fournir une borne d'erreur sur le résultat numérique donné par la méthode de Gauss-Legendre, il est nécessaire de mener une analyse pas à pas de l'algorithme 9.

En plus des paramètres de l'algorithme 9 nous avons besoin d'une borne supérieure M de $|f^{(2n)}|$ sur $[a,b]$; p est la précision de travail en nombre de bits de la mantisse. Les bornes a et b sont données sous la forme d'oracles retournant le nombre flottant arrondi $\widehat{a}$ ou $\widehat{b}$ au plus proche de leur valeur exacte dans la précision demandée ; et m est une borne supérieure de $|f'|$ sur $[a,b]$.

Nous allons maintenant prouver le théorème principal :

Théorème 8. *Soit* $\delta_{\widehat{y_i}} = (\frac{7}{2} + 2^{1-p})\mathrm{ulp}(\widehat{y_i}) + \left(\frac{17}{4} + 17 \cdot 2^{-p-2}\right) m\widehat{w_i}\mathrm{ulp}(\widehat{x_i})$*, où* $\widehat{y_i}$*,* $\widehat{w_i}$ *et* $\widehat{x_i}$ *sont définis dans l'algorithme 9,* p *désigne la précision et* m *une borne sur* $|f'|$*. Lors du calcul numérique de l'intégrale de* f *sur* $[a,b]$ *en utilisant l'algorithme 9 l'erreur totale est majorée par :*

$$B_{totale} = \left(\frac{9}{2} + 3 \cdot 2^{-p}\right)\mathrm{ulp}(\widehat{I}) + n(1+2^{-p})\widehat{D} \cdot \max(\delta_{\widehat{y_i}}) + \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}M.$$

Corollaire 2. *Si l'on suppose que* $p \geq 2$ *on peut écrire plus simplement :*

$$\begin{cases} \delta_{\widehat{y_i}} & \leq & \frac{11}{4}\mathrm{ulp}(\widehat{y_i}) + 6m\widehat{w_i}\mathrm{ulp}(\widehat{x_i}) \\ B_{totale} & \leq & \frac{21}{4}\mathrm{ulp}(\widehat{I}) + \frac{5n}{4}\widehat{D} \cdot \max(\delta_{\widehat{y_i}}) + \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}M. \end{cases}$$

On peut analyser l'algorithme par étapes :

1. Le calcul des poids w_i, $i \in [0, n-1]$ de la méthode. On les calcule en arrondi au plus proche de leur valeur exacte comme expliqué dans la section 13 :

$$|\widehat{w_i} - w_i| \leq \frac{1}{2}\mathrm{ulp}(\widehat{w_i}).$$

2. Le calcul des x_i. Soit x'_i le point d'évaluation correspondant sur $[-1,1]$, et $v_i = \frac{1+x'_i}{2}$. On suppose que v_i est calculé comme l'arrondi au plus proche de sa valeur exacte :

$$|\widehat{v_i} - v_i| \leq \frac{1}{2}\mathrm{ulp}(\widehat{v_i}),$$

$$\widehat{x_i} = \circ(\circ(\widehat{v_i} \cdot \widehat{(b-a)}) + \widehat{a}).$$

Comme a et b sont donnés comme des oracles, on peut supposer que $b-a$ ainsi que a sont calculés comme arrondis au plus proche de leur valeur exacte. L'analyse d'erreur donne :

$$\begin{array}{rcl} |\circ(\widehat{v_i} \cdot \widehat{b-a}) - v_i \cdot (b-a)| & \leq & \frac{5}{2}\mathrm{ulp}(\circ(\widehat{v_i} \cdot \widehat{b-a})) \quad \text{[Lemme 6]} \\ |\widehat{x_i} - x_i| & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{x_i}) + \frac{5}{2}\mathrm{ulp}(\circ(\widehat{v_i} \cdot \widehat{b-a})) + \frac{1}{2}\mathrm{ulp}(\widehat{a}) \\ & \leq & \frac{17}{4}\mathrm{ulp}(\widehat{x_i}). \quad \text{[Lemme 4]} \end{array}$$

3. Le calcul des $f(x_i)$. On suppose que l'on dispose d'une implémentation de f qui calcule un résultat sur lequel l'erreur est bornée par 1 ulp. En tenant compte de la borne déjà calculée pour $\widehat{x_i}$ on a :

$$|f(\widehat{x_i}) - f(x_i)| = |f'(\theta_i)(\widehat{x_i} - x_i)|, \quad \theta_i \in [\min(x_i, \widehat{x_i}), \max(x_i, \widehat{x_i})]$$

et une borne supérieure sur f' permet de borner cette erreur de façon absolue. Soit $\widehat{f_i} = \circ(f(\widehat{x_i}))$ le nombre flottant calculé. À cette étape nous avons :

$$\begin{array}{rcl} \delta_{\widehat{f_i}} = |\widehat{f_i} - f(x_i)| & \leq & |f'(\theta_i)(\widehat{x_i} - x_i)| + \frac{1}{2}\mathrm{ulp}(\widehat{f_i}) \\ & \leq & \frac{17}{4}m \cdot \mathrm{ulp}(\widehat{x_i}) + \frac{1}{2}\mathrm{ulp}(\widehat{f_i}). \end{array}$$

4. Le calcul des $y_i = f(x_i) \cdot w_i$. L'erreur accumulée jusqu'ici est :

$$\begin{array}{rcl} |\widehat{y_i} - y_i| & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{y_i}) + |\widehat{f_i}\widehat{w_i} - f(x_i)w_i| \\ & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{y_i}) + \widehat{f_i}|\widehat{w_i} - w_i| + w_i|\widehat{f_i} - f(x_i)| \\ & \leq & \frac{1}{2}\mathrm{ulp}(\widehat{y_i}) + \frac{1}{2}\widehat{f_i}\mathrm{ulp}(\widehat{w_i}) + w_i\delta_{\widehat{f_i}} \\ & \leq & \frac{3}{2}\mathrm{ulp}(\widehat{y_i}) + w_i\left[\frac{17}{4}m \cdot \mathrm{ulp}(\widehat{x_i}) + \frac{1}{2}\mathrm{ulp}(\widehat{f_i})\right] \quad \text{[Lemmes 1 et 2]} \\ & \leq & \frac{3}{2}\mathrm{ulp}(\widehat{y_i}) + (1+2^{-p})\widehat{w_i}\left[\frac{17}{4}m \cdot \mathrm{ulp}(\widehat{x_i}) + \frac{1}{2}\mathrm{ulp}(\widehat{f_i})\right] \quad \text{[Lemme 3]} \\ & \leq & (\frac{7}{2} + 2^{1-p})\mathrm{ulp}(\widehat{y_i}) + (1+2^{-p})m\widehat{w_i}\frac{17}{4}\mathrm{ulp}(\widehat{x_i}) \quad \text{[Lemmes 1 et 2]} \\ & \leq & (\frac{7}{2} + 2^{1-p})\mathrm{ulp}(\widehat{y_i}) + \left(\frac{17}{4} + 17 \cdot 2^{-p-2}\right)m\widehat{w_i}\mathrm{ulp}(\widehat{x_i}) = \delta_{\widehat{y_i}}. \end{array}$$

Remarque : il n'y a, comme dans l'analyse d'erreur des méthodes de Newton-Cotes, pas de réel problème de définition de $\mathrm{ulp}(\widehat{x_i})$ lorsque $\widehat{x_i} = 0$, car dans ce cas l'hypothèse qu'il ne se produit pas d'*underflow* permet de voir facilement que l'erreur sur $\widehat{x_i}$ est nulle.

Pour calculer la borne d'erreur on ne tient compte que de $\max(\delta_{\widehat{y_i}})$.

5. La sommation des y_i : on le fait avec l'algorithme de Demmel et Hida [13], qui garantit une erreur d'au plus $1,5$ ulp sur le résultat final. Cet algorithme utilise une précision de calcul plus grande $p' \approx p + \log_2(n)$. Soit $S = \sum_{i=0}^{n-1} y_i$.

$$|\widehat{S} - S| \leq \frac{3}{2}\mathrm{ulp}(\widehat{S}) + n \cdot \max(\delta_{\widehat{y_i}}).$$

6. la multiplication par $\frac{b-a}{2}$: $I = \frac{b-a}{2}S$. On note $D = \frac{b-a}{2}$ et on suppose comme précédemment que l'entrée $\widehat{b-a}$ est calculée en arrondi au plus proche de sa valeur exacte. Comme la division par 2 est exacte en binaire on a :

$$\begin{aligned}
|\widehat{D} - D| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{D}) \\
|\widehat{I} - I| &\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + |\widehat{S}\widehat{D} - SD| \\
&\leq \tfrac{1}{2}\mathrm{ulp}(\widehat{I}) + \tfrac{1}{2}|\widehat{S}|\mathrm{ulp}(\widehat{D}) + D|\widehat{S} - S| \\
&\leq \tfrac{3}{2}\mathrm{ulp}(\widehat{I}) + D\left[\tfrac{3}{2}\mathrm{ulp}(\widehat{S}) + n\cdot\max(\delta_{\widehat{y}_i})\right] \quad \text{[Lemmes 1 et 2]} \\
&\leq \tfrac{3}{2}\mathrm{ulp}(\widehat{I}) + (1+2^{-p})\widehat{D}\left[\tfrac{3}{2}\mathrm{ulp}(\widehat{S}) + n\cdot\max(\delta_{\widehat{y}_i})\right] \quad \text{[Lemme 3]} \\
&\leq (\tfrac{9}{2} + 3\cdot 2^{-p})\mathrm{ulp}(\widehat{I}) + n(1+2^{-p})\widehat{D}\cdot\max(\delta_{\widehat{y}_i}). \quad \text{[Lemmes 1 et 2]}
\end{aligned}$$

Corollaire 3. *Si l'on suppose de plus que f ne change pas de signe sur $[a,b]$, alors on peut en déduire la meilleure borne suivante :*

$$\begin{aligned}
B'_{totale} = &\left(\frac{9}{2} + 3\cdot 2^{-p} \quad + (2+2^{1-p})(7+2^{2-p})(1+3\cdot 2^{-p})\right)\mathrm{ulp}(\widehat{I}) \\
&+ nm(1+2^{-p})\left(\frac{17}{4} + 17\cdot 2^{-p-2}\right)\widehat{D}\max(\widehat{w_i}\mathrm{ulp}(\widehat{x_i})) \\
&+ \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}M.
\end{aligned}$$

Preuve : Supposons par exemple que $f \geq 0$, alors sachant que les poids w_i de la méthode de Gauss-Legendre sont positifs

$$\forall i \in [0, n-1], \widehat{y_i} = \circ(\widehat{w_i}\cdot\widehat{f_i}) \geq 0$$

donc

$$\mathrm{ulp}(\widehat{y_i}) \leq 2^{1-p}\widehat{y_i}.$$

Soit $\tilde{S} = \sum_{i=0}^{n-1}\widehat{y_i}$, on sait que

$$\begin{aligned}
|\tilde{S} - \widehat{S}| &\leq \tfrac{3}{2}\mathrm{ulp}(\widehat{S}) \\
\tilde{S} &\leq (1+3\cdot 2^{-p})\widehat{S} \\
L &= \textstyle\sum_{i=0}^{n-1}(\frac{7}{2} + 2^{1-p})\mathrm{ulp}(\widehat{y_i}) \\
&\leq (\tfrac{7}{2} + 2^{1-p})2^{1-p}\textstyle\sum_{i=0}^{n-1}\widehat{y_i} \\
&\leq 2^{1-p}(\tfrac{7}{2} + 2^{1-p})(1+3\cdot 2^{-p})\widehat{S} \\
&\leq (7+2^{2-p})(1+3\cdot 2^{-p})\mathrm{ulp}(\widehat{S}).
\end{aligned}$$

On en déduit alors la borne d'erreur suivante sur $\widehat{S}$:

$$\begin{aligned}
|\widehat{S} - S| \leq &\left(\frac{3}{2} + (7+2^{2-p})(1+3\cdot 2^{-p})\right)\mathrm{ulp}(\widehat{S}) \\
+ \quad &nm\left(\frac{17}{4} + 17\cdot 2^{-p-2}\right)\max(\widehat{w_i}\mathrm{ulp}(\widehat{x_i}))
\end{aligned}$$

ce qui produit bien le résultat annoncé en injectant cette borne dans celle de $|\widehat{I} - I|$ donnée plus haut. □

Le lemme suivant permet de simplifier encore cette borne :

Lemme 13. *Si $p \geq 5$ alors dans l'algorithme 9 on a pour tout $i \in \{0, \ldots, n-1\}$*

$$\text{ulp}(\widehat{x_i}) \leq 2 \max(\text{ulp}(a), \text{ulp}(b)).$$

PREUVE : On a montré que $\forall i \in \{0, \ldots, n-1\}, |x_i - \widehat{x_i}| \leq \frac{17}{4}\text{ulp}(\widehat{x_i})$. On peut sans perte de généralité supposer $|b| \geq |a|$ et $b \geq 0$ et se ramener à montrer dans ce cas que

$$\text{ulp}(\widehat{x_i}) \leq 2 \text{ ulp}(b).$$

Le point x_i est le poids exact associé à $\widehat{x_i}$ et il vérifie en particulier

$$x_i \leq b.$$

On en déduit

$$\begin{aligned} \widehat{x_i} &\leq x_i + \frac{17}{4}\text{ulp}(\widehat{x_i}) \\ &\leq b + \frac{17}{4}\text{ulp}(\widehat{x_i}) \end{aligned}$$

Avec la propriété

$$2^{p-1}\text{ulp}(x) \leq |x| < 2^p\text{ulp}(x) \tag{4.8}$$

pour tout réel x non nul on obtient alors

$$\begin{aligned} \widehat{x_i} &\leq b + 17 \cdot 2^{-p-1}\widehat{x_i} \\ \widehat{x_i}(1 - 17 \cdot 2^{-p-1}) &\leq b \\ \widehat{x_i} &\leq \frac{b}{1 - 17 \cdot 2^{-p-1}} \\ \widehat{x_i} &\leq \frac{64}{47}b \end{aligned}$$

puis l'application de (4.8) donne

$$\begin{aligned} \text{ulp}(\widehat{x_i}) &\leq 2^{1-p}\widehat{x_i} \\ &\leq \frac{64}{47}2^{1-p}b \\ &< 2^{2-p}b \\ \text{ulp}(\widehat{x_i}) &< 4\text{ulp}(b) \\ \text{ulp}(\widehat{x_i}) &\leq 2\text{ulp}(b) \end{aligned}$$

car les ulp sont des puissances de 2. □

Corollaire 4. *Soit m une borne de $|f'|$ et M une borne de $|f^{(2n)}|$ sur $[a, b]$. Si f ne change pas de signe sur $[a, b]$ et $p \geq 5$ alors l'erreur totale du résultat de l'algorithme 9 est borné par*

$$B''_{totale} = 21\text{ulp}(\widehat{I}) + 10nm\widehat{D}\max(\text{ulp}(\widehat{a}), \text{ulp}(\widehat{b})) + \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3}M.$$

PREUVE : On applique le lemme 13 en sachant que $0 \leq \widehat{w_i} \leq 1$ pour tout $i \in \{0, \ldots, n-1\}$. □

15 Expérimentations

L'algorithme 9 a été écrit dans CRQ avec la bibliothèque MPFR [34]. En plus de la valeur numérique de l'intégrale, le programme fournit une borne sur l'erreur totale B_{totale} en la décomposant en deux termes :

1. la borne sur l'erreur mathématique $B_{\text{math}} = \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} M$,
2. une borne sur les erreurs d'arrondi $B_{\text{arrondi}} = B_{\text{totale}} - B_{\text{math}}$.

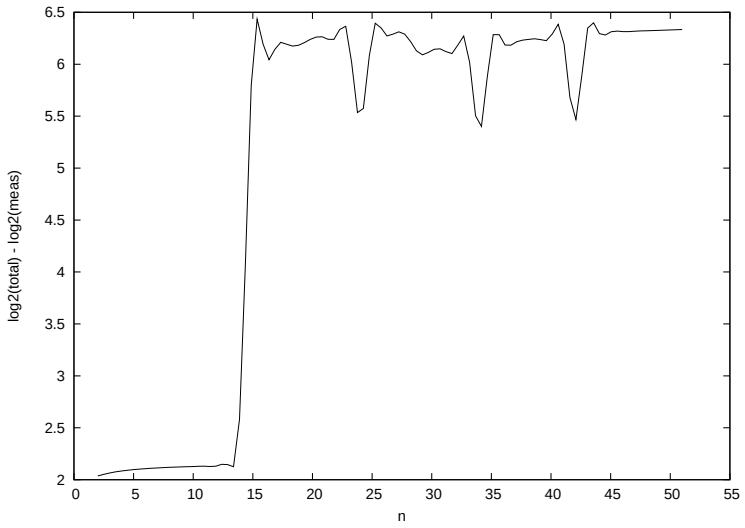

FIG. 4.1 – La surestimation de l'erreur en bits lors du calcul de $\int_0^3 e^x dx$ avec 113 bits de précision et n points d'évaluation.

Pour mener les expérimentations nous avons choisi une fonction et un domaine d'intégration pour lesquelles la valeur exacte de l'intégrale est connue, de telle sorte qu'il soit possible de mesurer précisément l'erreur commise dans le calcul (et on la note $E_{\text{mesurée}}$). La figure 4.1 montre dans quelle mesure la borne d'erreur calculée est pessimiste lors du calcul de l'intégrale $I = \int_0^3 e^x \mathrm{d}x$ avec 113 bits de précision de travail, et un nombre de points d'évaluation qui varie de 2 à 100. On calcule le nombre de bits perdus $(\log_2(B_{\text{totale}}) - \log_2(E_{\text{mesurée}}))$.

Les erreurs dues aux arrondis (B_{arrondi}) sont plutôt stables. C'est un comportement attendu pour la méthode d'intégration de Gauss-Legendre, où tous les poids sont positifs. Cette propriété nous permet d'espérer une amélioration de la précision via l'augmentation du nombre n de points d'évaluation à précision de travail fixée (ce n'est pas le cas pour la méthode de Newton-Cotes par exemple).

On observe que la borne sur l'erreur totale donnée par l'algorithme est proche de l'erreur effectivement mesurée. Dans les données expérimentales on constate une perte d'au maximum 7 bits de précision sur le résultat final à cause d'une borne trop pessimiste.

Cette surestimation de l'erreur est cependant notable seulement pour $n \geq 15$, ce qui correspond au point où l'erreur mathématique devient plus faible que l'erreur d'arrondi. Notre surestimation porte donc essentiellement sur celle-ci, tandis que l'erreur mathématique est bien estimée.

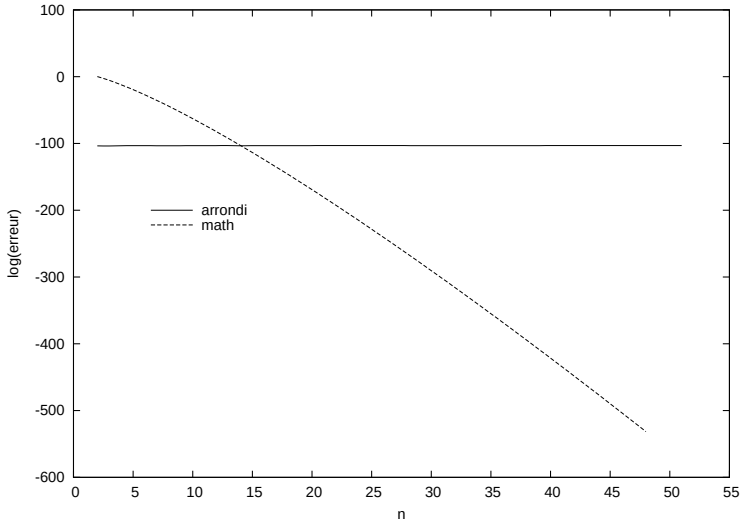

FIG. 4.2 – Les bornes sur les erreurs d'arrondi B_{arrondi} et sur l'erreur mathématique B_{math} lors du calcul de $\int_0^3 e^x \mathrm{d}x$ avec 113 bits de précision et n points d'évaluation.

16 Conclusion

Les méthodes d'intégration de Gauss-Legendre forment une famille d'intégration numérique robuste. L'analyse d'erreur effectuée dans cette section permet d'équiper l'implémentation logicielle d'une borne d'erreur dynamique sur le résultat final. À condition que le résultat de l'intégrale ne soit pas un réel exactement représentable par un nombre flottant, on en déduit un algorithme d'intégration numérique avec arrondi correct.

Le temps de calcul de l'algorithme est dominé par le calcul des poids et des abscisses (figure 4.3). Une discussion plus détaillée des algorithmes mis en jeu se trouve en section 18 ; dans une certaine mesure une façon de contourner ce problème peut être de conserver une table de pré-calcul des racines.

Le choix des paramètres optimaux reste à régler. Le tableau 4.4 donne le plus petit n pour lequel la borne sur l'erreur mathématique B_{math} est plus petite que la borne B_{arrondi} sur l'erreur d'arrondi, pour ce calcul précis de $\int_0^3 \exp(x)\mathrm{d}x$. Le calcul de ce tableau dans le cas général semble un but assez éloigné, mais la figure 4.4 donne une première idée dans cette direction.

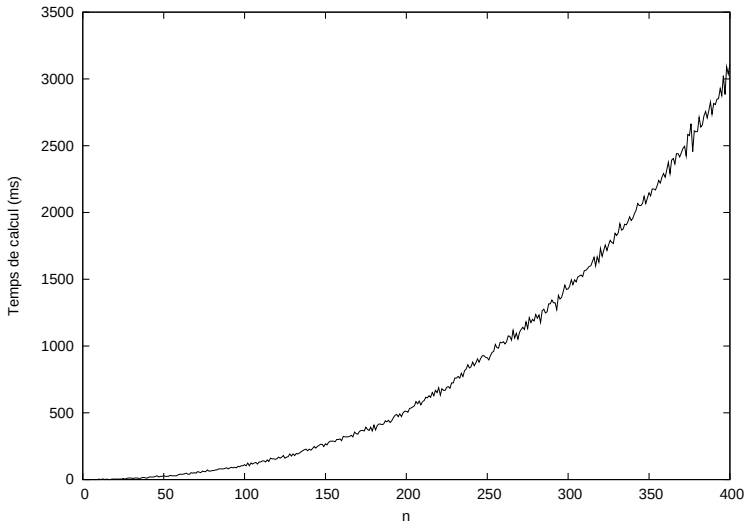

FIG. 4.3 – Temps de calcul des poids et abscisses pour 113 bits de précision et n points d'évaluation. Mesures effectuées sur un Pentium 4 à 3,2GHz.

Précision de calcul en bits	Ordre optimal	Nombre de bits corrects prédits
53	8	47
113	15	108
200	22	194
400	38	395
1000	80	995

FIG. 4.4 – Ordres optimaux pour différentes précisions de calcul, pour le calcul de $\int_0^3 e^x \mathrm{d}x$.

5

Isolation et raffinement de racines

Dans la suite du chapitre, $P(X) = \sum_{i=0}^{n} a_i X^i$ $(a_n \neq 0)$ est un polynôme à coefficients réels dont on cherche à calculer les racines réelles. Plus précisément si P a k racines réelles $u_1 < \ldots < u_k$ on cherche des nombres flottants $\widehat{u_1}, \ldots, \widehat{u_k}$ tels que :

$$\forall i \in \{1, \ldots, k\}, |u_i - \widehat{u_i}| \leq \mathrm{ulp}_p(\widehat{u_i})$$

où p est la précision demandée sur les racines calculées, donnée en paramètre. On demande donc l'arrondi fidèle en précision p bits. La quantité $\mathrm{ulp}_p(0)$ n'est pas définie mais le cas où $P(0) = 0$ se détecte facilement par $a_0 = 0$ et dans ce cas la racine nulle est calculée sans erreur.

17 Isolation de racines

Une étape préliminaire à l'algorithme de raffinement de racines implémenté consiste en l'isolation des racines. Il s'agit de fournir une liste d'intervalles rationnels contenant chacun exactement une racine du polynôme considéré et telle que chaque racine réelle se retrouve dans un intervalle (et un seul). Nous utilisons pour ceci l'algorithme d'Uspensky tel que décrit dans [32] dont nous donnons un rappel ici.

La méthode d'isolation repose sur trois points clefs :

- une borne du nombre de racines réelles positives d'un polynôme déduite simplement des coefficients du polynôme,
- des transformations élémentaires du polynôme efficaces en temps et en mémoire,
- un parcours intelligent de l'espace de recherche.

17.1 Règle des signes de Descartes

La règle des signes de Descartes borne le nombre de racines réelles positives d'un polynôme de la façon suivante :

Théorème 9 (Descartes)**.** *Soit $P = \sum_{i=0}^{n} a_i X^i$ un polynôme à coefficients réels. On note $V(P) = V(a_0, a_1, \ldots, a_n)$ le nombre de changements de signe dans la suite de ses coefficients dans laquelle on a enlevé les coefficients nuls. Alors avec $R(P)$ le nombre de racines réelles positives de P on a $R(P) \leq V(P)$ et $R(P) \equiv V(P) \bmod 2$.*

Cette borne a l'avantage d'être très efficace à calculer ($\mathcal{O}(n)$ comparaisons) et de plus il y a égalité dans le cas où $V(P) \in \{0, 1\}$.

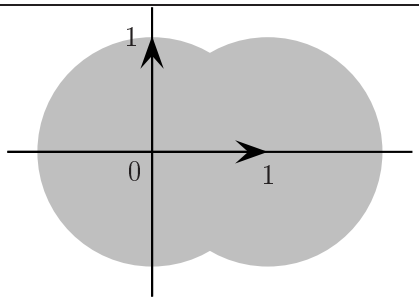

FIG. 5.1 – La zone complexe proscrite pour les racines de P dans le théorème 10.

Trois transformations de polynômes sont utilisées dans l'algorithme d'isolation. Pour c réel on définit la translation T_c :

$$\begin{aligned} T_c : \mathbb{R}[X] &\to \mathbb{R}[X] \\ P(X) &\mapsto P(X+c) \end{aligned}$$

qui a pour effet de déplacer l'origine en c. La transformation R renverse la liste des coefficients d'un polynôme, elle est définie par :

$$\begin{aligned} R : \mathbb{R}[X] &\to \mathbb{R}[X] \\ P &\mapsto X^{\deg(P)} P(1/X). \end{aligned}$$

Enfin pour c réel non nul on appelle H_c l'homothétie de poids c :

$$\begin{aligned} H_c : \mathbb{R}[X] &\to \mathbb{R}[X] \\ P &\mapsto P(cX). \end{aligned}$$

On vérifie que ces trois applications transforment bien un polynôme à coefficients réels en un polynôme à coefficients réels.

Un résultat similaire à la règle des signes de Descartes, dû à Collins et Johnson, permet de se restreindre à l'intervalle $]0,1[$:

Théorème 10. *Soit P un polynôme réel sans facteur carré ayant une racine réelle unique dans $]0,1[$ et aucune racine non réelle dans chacun des deux disques complexes de rayon 1 et centres $(0,0)$ et $(1,0)$ (voir figure 5.1). Alors $V(T_1[R[P]]) = 1$.*

Une « réciproque » au théorème 10 est nécessaire pour avoir un critère effectif sur $[0,1]$, elle est donnée dans [36] :

Théorème 11. *Soit P un polynôme réel sans racine dans $]0,1[$ et sans racine non réelle en dehors du cercle de centre $(1/2,0)$ et de diamètre 1. Alors $V(T_1[R[P]]) = 0$.*

17.2 Algorithmes

On commence par préciser l'implémentation des transformations R, T_c et H_c avant de présenter l'algorithme d'isolation en tant que tel. Ces transformations sont détaillées dans les Algorithmes 10, 11 et 12 où la sortie est un nouveau polynôme ; parfois on voudra au contraire faire la transformation en place (il suffit de quelques modifications mineures dans les algorithmes donnés pour cela). L'algorithme 12 est donné ici dans une version quadratique en le nombre d'opérations

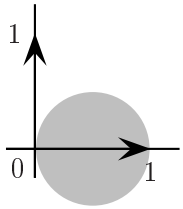

FIG. 5.2 – La zone complexe prescrite pour les racines non réelles de P dans le théorème 11.

arithmétiques ; il est possible de faire linéaire (à des facteurs logarithmiques près) en passant par la représentation de Newton [5].

Du point de vue de la recherche des racines, on peut considérer ces trois opérations comme transformations du domaine de recherche.

Par exemple, le nombre de racines de P dans $\mathbb{R}^+$ est égal au nombre de racines de $R[T_{-1}[P]]$ dans $]0,1[$.

Algorithme 10 Renversement R

ENTRÉE : $[a_0, a_1, \ldots, a_n]$.
SORTIE : $[t_0, t_1, \ldots, t_n]$ tel que $T(X) = \sum_{i=0}^{n} t_i X^i = X^{\deg(P)} P(1/X)$.
1: **for** $i \leftarrow 0$ to n **do**
2: $\quad t_i \leftarrow a_{n-i}$
3: **end for**
4: **return** t

Algorithme 11 Homothétie H_c

ENTRÉE : $[a_0, a_1, \ldots, a_n]$.
SORTIE : $[t_0, t_1, \ldots, t_n]$ tel que $T(X) = \sum_{i=0}^{n} t_i X^i = P(cX)$.
1: $p \leftarrow 1$
2: **for** $i \leftarrow 0$ to n **do**
3: $\quad t_i \leftarrow p a_i$
4: $\quad p \leftarrow c \cdot p$
5: **end for**
6: **return** t

On note $I_{c,k} =]\frac{c}{2^k}, \frac{c+1}{2^k}[$ pour c et k entiers. Le nombre de racines de P dans $I_{c,k}$ est égal au nombre de racines de $P_{c,k}(X) = P(\frac{c+X}{2^k})$ dans $]0,1[$. Remarquons que

$$P_{c,k} = T_c[H_{2^{-k}}[P]] \tag{5.1}$$

donc $P_{c,k}$ peut se calculer directement à partir de P via des transformations élémentaires.

Une première étape préparatoire à l'isolation des racines consiste à ramener l'espace de recherche de $\mathbb{R}$ à $]0,1[$. Si B borne les racines de P en valeur absolue, on isole les racines positives de P via l'isolation des racines de $P' = H_B[P]$ dans $]0,1[$; l'isolation des racines négatives de P étant réalisée par l'isolation des racines de $P'' = H_{-B}[P]$. On suppose donc dans la suite que l'on cherche les racines de P dans $]0,1[$.

Algorithme 12 Translation T_c

ENTRÉE : $[a_0, a_1, \ldots, a_n]$.
SORTIE : $[t_0, t_1, \ldots, t_n]$ tel que $T(X) = \sum_{i=0}^{n} t_i X^i = P(X + c)$.

```
1: for i ← 0 to n do
2:     t_i ← a_i
3: end for
4: for i ← 1 to n do
5:     for j ← n − i to n − 1 do
6:         t_j ← t_j + c · t_{j+1}
7:     end for
8: end for
9: return t
```

L'idée générale de l'isolation consiste à parcourir l'espace des couples $(c, k), c < 2^k$ pour en retenir ceux tels que $I_{c,k}$ contienne exactement une racine de P et qu'aucune racine ne soit oubliée. Le cas des racines exactement représentables en binaire $\alpha = \frac{c}{2^k}$ se traite en calculant le signe de $P_{c,k}(0)$ à chaque étape. On omet ces racines par la suite pour des raisons de simplicité. Il est naturel d'organiser les couples (c, k) en un arbre binaire dont la racine $(c = 0, k = 0)$ représente l'intervalle $]0, 1[$ tout entier, les deux nœuds du niveau suivant $(0, 1), (1, 1)$ représentent $]0, \frac{1}{2}[$ et $]\frac{1}{2}, 1[$ respectivement, et ainsi de suite. Pour chacun de ces couples (c, k) que l'on souhaite tester, on calcule $P_{c,k}$ et on applique le critère du Théorème 9 à $P_{c,k}$ sur $]0, 1[$ pour conclure sur le nombre de racines de P dans $I_{c,k}$. Suivant le nombre de changements de signe dans les coefficients de $P_{c,k}$ on pourra soit oublier le sous-arbre correspondant (pas de racine), soit recopier le nœud dans la sortie de l'algorithme (une racine exactement), ou enfin parcourir le sous-arbre (peut-être plus d'une racine).

Divers algorithmes sont envisageables en fonction du parcours de l'arbre binaire retenu et des compromis temps de calcul / espace mémoire requis. L'équation (5.1) permet de sauter directement de la racine de l'arbre à un nœud arbitraire sans stocker de résultat intermédiaire, mais ce n'est pas forcément la stratégie la plus efficace en temps de calcul.

Ce cadre unifié regroupant les algorithmes d'isolation connus est présenté dans [32] de même qu'un nouvel algorithme s'inscrivant dans ce cadre, optimal en espace mémoire et avec le même nombre d'opérations que les autres algorithmes basés sur la règle des signes de Descartes. C'est cet algorithme qui est utilisé dans CRQ. Il utilise un parcours en profondeur de l'arbre et calcule à chaque étape le polynôme $P_{c,k}$ du nœud courant, sans retenir les polynômes des autres nœuds visités. Remarquons que pour (c, k) et (c', k') deux nœuds de l'arbre on a

$$P_{c',k'} = H_{2^{k-k'}} \left[T_{2^{k-k'}c'-c}[P_{c,k}] \right]$$

ce qui permet de passer d'un nœud quelconque à un autre sans forcément revenir à la racine (une généralisation de l'équation (5.1)), et que de plus lors d'un parcours en profondeur seules des translations de 1 apparaissent :

- pour passer d'un nœud (c, k) à son fils gauche $(2c, k + 1)$ il n'y a pas de translation mais une homothétie $H_{1/2}$,
- pour passer d'un nœud (c, k) à son frère droit $(c + 1, k)$, il y a une translation T_1,
- pour passer d'un nœud (c', k') au frère droit non visité (c, k) de l'ancêtre le plus proche (par *backtracking*) il n'y a aussi qu'une translation de 1 : si n est le nombre de niveaux à

remonter alors on vérifie que $c = 2^n c' - 1$, $k = k' + n$ et alors $2^{k-k'}c' - c = 1$. On effectue n homothéties H_2 (équivalentes à une homothétie H_{2^n}) et une translation T_1.

Il est intéressant de privilégier des translations par des petits entiers, en particulier dans l'Algorithme 12 si $c = 1$ alors il n'y a que des additions et aucune multiplication. La complexité théorique binaire de l'algorithme d'isolation utilisé dans CRQ est $\mathcal{O}(n^6 t^2 \log^2 n)$ en temps où t est une borne sur la taille en bits des coefficients de P pour une occupation mémoire $\mathcal{O}(n^3 t \log n)$ [32] et $n = \deg P$.

L'isolation des racines de P produit donc des intervalles $I_{c,k}$ dont les bornes sont directement des nombres flottants. En effet $x_{\min} = c2^{-k}$ est représentable comme un flottant avec une mantisse de taille $p = 1 + \lfloor \log_2 c \rfloor$ bits, avec $\mathrm{ulp}(x_{\min}) = 2^{-k}$. De même pour $x_{\max} = (c+1)2^{-k}$. En notant u l'unique racine de P dans $I_{c,k}$ on a :

$$\begin{aligned} c2^{-k} < \quad & u & < (c+1)2^{-k} \\ 0 < \quad & u - x_{\min} & < \mathrm{ulp}(x_{\min}) \end{aligned}$$

donc $x_{\min}$ est l'arrondi vers le bas de u en précision p bits (et $x_{\max}$ l'arrondi vers le haut). Du point de vue de la précision relative on peut donc considérer que p bits de la racine sont connus ; l'erreur absolue étant bornée par 2^{-k}.

On constate dans le déroulement de l'algorithme que deux racines sont séparées au mieux par le premier bit différent dans leur développement binaire ; c'est cette distance au sens binaire qui borne la profondeur de l'arbre de recherche. D'autres transformations élémentaires produisent d'autres formes d'intervalles d'isolation. Par exemple les deux transformations

$$P \leftarrow (x+1)^n P\left(\frac{1}{x+1}\right)$$

et

$$P \leftarrow P(x+1)$$

envoient l'intervalle $]0, +\infty[$ sur les intervalles $]0, 1[$ et $]1, +\infty[$ respectivement, ce qui permet d'appliquer récursivement la règle de Descartes simple à P. Il s'agit de l'algorithme original d'Uspensky tel que décrit dans [10]. Les bornes de séparation des intervalles ont alors une forme plus proche d'une réduite dans le développement en fraction continue d'une des racines plutôt que de son développement binaire ; ce qui peut poser un problème en pratique lorsque l'on préfère un tel développement binaire. S'il est possible de produire des polynômes favorisant davantage une approche ou l'autre, l'expérience des polynômes de Legendre montre en pratique que la méthode binaire est suffisante pour notre application[3] ce qui ne nous a pas conduit à chercher une stratégie optimale à ces polynômes.

Dans la suite on parlera par abus de « l'algorithme d'Uspensky » en faisant référence non pas à l'algorithme original mais à la version binaire optimale parmi celles présentées dans [32] qui est celle utilisée dans CRQ (cette version est baptisée « Rel » dans [32]).

18 Raffinement de racines

Le raffinement d'une racine consiste à calculer cette racine à une précision donnée à partir d'un intervalle d'isolation, à savoir un intervalle contenant une et une seule racine de la fonction

[3] Le coût de l'isolation est essentiellement négligeable devant celui du raffinement.

étudiée (dans notre cas il s'agit d'un polynôme). Notons u la racine cherchée et p la précision demandée sur u, il s'agit de calculer $\widehat{u}$ sur p bits tel que

$$|\widehat{u} - u| \leq \mathrm{ulp}_p(\widehat{u})$$

pour un arrondi fidèle ou

$$|\widehat{u} - u| \leq \frac{1}{2}\mathrm{ulp}_p(\widehat{u}) \quad [= \mathrm{ulp}_{p+1}(\widehat{u}) \text{ en binaire}]$$

pour un arrondi au plus proche. À strictement parler l'arrondi au plus proche demande en plus de respecter la règle de l'arrondi pair et impose donc des conditions supplémentaires. On demande ici seulement une borne sur l'erreur relative — ce qui est suffisant dans l'application ultérieure du raffinement de racines — et on ne garantit pas que le flottant calculé correspond exactement à l'arrondi au plus proche au sens de la norme IEEE754. Les problèmes de pire cas [24] sont donc évités.

18.1 Dichotomie

La recherche dichotomique d'une racine est basée sur le lemme élémentaire suivant :

Lemme 14. *Soit $f : [a, b] \to \mathbb{R}$ continue tel que $f(a)f(b) \leq 0$, alors il existe $x \in [a, b]$ tel que $f(x) = 0$.*

On en déduit un algorithme simple de raffinement qui consiste à couper l'intervalle de recherche en deux à chaque étape. La procédure est décrite dans l'algorithme 13. Chaque étape

Algorithme 13 Raffinement dichotomique

ENTRÉE : f, a, b, ε tels que $f(a)f(b) < 0$ et $a < b$.
SORTIE : a', b' tels que $f(a')f(b') \leq 0$, $|b' - a'| \leq \varepsilon$ et $[a', b'] \subseteq [a, b]$.

```
1:  sa ← sign(f(a))                          ▷ La fonction sign est trivaluée.
2:  sb ← sign(f(b))
3:  a' ← a
4:  b' ← b
5:  while b' − a' > ε do
6:      m ← (a'+b')/2
7:      sm ← sign(f(m))
8:      if sa = sm then
9:          a' ← m
10:     else
11:         b' ← m
12:     end if
13: end while
14: return (a', b')
```

divise l'intervalle de recherche de la racine en deux et permet donc de gagner un bit de précision sur la racine en question. On aura donc $\log_2(\frac{b-a}{\varepsilon})$ étapes de boucle pour raffiner avec précision absolue ε.

Le calcul du signe de $f(m)$ à chaque étape est la partie coûteuse de l'algorithme. Dans le cas qui nous intéresse des polynômes de Legendre, on peut se ramener à des coefficients entiers. Le calcul du signe a été envisagé dans notre implémentation de deux manières : une méthode « naïve » en précision maximale, et une méthode tronquée.

Dichotomie naïve

Dans l'algorithme 13 il est nécessaire de calculer le signe de P en divers nombres flottants que l'on peut écrire sous la forme $\frac{c}{2^k}$. Lorsque les coefficients de P sont entiers il est possible de calculer le signe de $P(\frac{c}{2^k})$ en ne faisant que des calculs entiers, donc exacts.

En effet,

$$\begin{aligned}
\operatorname{sign}\left(P\left(\frac{c}{2^k}\right)\right) &= \operatorname{sign}\left(2^{kn}P\left(\frac{c}{2^k}\right)\right) \\
&= \operatorname{sign}\left(2^{kn}\sum_{i=0}^{n} a_i\left(\frac{c}{2^k}\right)^i\right) \\
&= \operatorname{sign}\left(\sum_{i=0}^{n} a_i c^i 2^{(n-i)k}\right).
\end{aligned}$$

En notant $Q(X) = \sum_{i=0}^{n} a_i 2^{k(n-i)} X^i$ on évalue le signe de $P(\frac{c}{2^k})$ comme le signe de $Q(c)$, où Q pourra être évalué par la méthode d'évaluation de polynôme de son choix, par exemple Horner comme cela est détaillé dans l'algorithme 14.

Algorithme 14 Calcul naïf du signe d'un polynôme P à coefficients entiers

ENTRÉE : $[a_0, a_1, \ldots, a_n]$ où $P(X) = \sum_{i=0}^{n} a_i X^i$, c, k.
SORTIE : $\operatorname{sign}(P(\frac{c}{2^k}))$.

```
1: r ← a_n
2: for i ← 1 to n do
3:     t ← a_{n−i} 2^{ik}
4:     r ← c · r + t
5: end for
6: return sign(r)
```

On note $K = \log_2(\max |a_i|)$ une borne sur la taille binaire des coefficients de P. On rappelle que par hypothèse $c < 2^k$. À l'étape i de la boucle dans l'algorithme 14 on a $|t| \leq 2^{K+ik}$ et en entrée de boucle $|r| \leq (i+1)2^{K+ik}$. Le coût dominant est celui de la multiplication ligne 4 à savoir $M(k, (K+ik)\log_2(i+1))$. Pour un polynôme P fixé, on s'intéresse essentiellement au coût de la dichotomie et donc de l'algorithme 14 en fonction de la précision k. Pour le calcul de complexité on peut supposer $ik \geq K$ et le coût de cette multiplication est donc $M(k, ik)$ en négligeant les facteurs logarithmiques.

Le coût total de l'algorithme 14 est donc $\mathcal{O}(n^2 M(k))$, à comparer à la taille du numérateur r calculé qui est de $\mathcal{O}(kn \log n)$ bits. Si l'algorithme 14 de détermination du signe est choisi dans l'algorithme 13 de raffinement de racine, en notant $p_{\text{fin}} = -\log_2 \varepsilon$ la valeur finale pour k alors le coût total du raffinement est $\mathcal{O}(n^2 p_{\text{fin}} M(p_{\text{fin}}))$.

Remarquons que l'algorithme 14 calcule non seulement le signe de $P\left(\frac{c}{2^k}\right)$ mais aussi la valeur de $2^{kn}P\left(\frac{c}{2^k}\right)$. En ajustant l'exposant on a donc calculé la valeur flottante exacte de $P\left(\frac{c}{2^k}\right)$, que l'on peut ensuite arrondir à une précision inférieure au besoin. En exploitant cette propriété il est possible d'appliquer le paradigme « diviser pour régner » à l'algorithme 14. En effet, décomposons P en sa partie haute et sa partie basse : $P(X) = X^{\lceil \frac{n}{2} \rceil} P_h(X) + P_l(X)$ où $\deg P_l = l = \lceil \frac{n}{2} \rceil$ et

Algorithme 15 Évaluation d'un polynôme P à coefficients entiers – dpr_eval

ENTRÉE : $[a_0, a_1, \ldots, a_n]$ où $P(X) = \sum_{i=0}^{n} a_i X^i$, c, k.
SORTIE : $2^{kn} P(\frac{c}{2^k})$.

```
1: if n = 0 then
2:     return a_0
3: end if
4: l ← ⌈n/2⌉
5: h ← n + 1 − l
6: lv ← dpr_eval([a_0, a_1, ..., a_{l−1}], c, k)
7: hv ← dpr_eval([a_l, a_{l+1}, ..., a_n], c, k)
8: t ← c^l
9: return hv · t + lv
```

deg $P_h = h = n - l$. Alors

$$\begin{aligned} 2^{kn} P\left(\frac{c}{2^k}\right) &= 2^{kn}\left(\left(\frac{c}{2^k}\right)^l P_h\left(\frac{c}{2^k}\right) + P_l\left(\frac{c}{2^k}\right)\right) \\ &= c^l 2^{kh} P_h\left(\frac{c}{2^k}\right) + 2^{kl} P_l\left(\frac{c}{2^k}\right). \end{aligned}$$

On en déduit directement l'algorithme récursif 15, dont la complexité est $\mathcal{O}(M(kn))$ en négligeant les facteurs logarithmiques. La complexité totale pour le raffinement en utilisant l'algorithme 15 serait alors $\mathcal{O}(p_{\text{fin}} M(np_{\text{fin}}))$. À n fixé cela n'est pas asymptotiquement meilleur que la version naïve mais les expériences montrent que cela est plus performant en pratique (voir chapitre 6).

Dichotomie tronquée

Algorithme 16 Évaluation de Horner avec borne sur l'erreur – Horner$_p$

ENTRÉE : $P = [a_0, a_1, \ldots, a_n]$, un flottant x, une précision de calcul p.
SORTIE : $\widehat{y_0}, \mu$ tel que $|P(x) - \widehat{y_0}| \leq \mu$.

```
1: ŷ_n ← ∘_p(a_n)
2: μ ← |ŷ_n|/2
3: for i ← n − 1 downto 0 do
4:     t̂_i ← ∘_p(x ŷ_{i+1})
5:     ŷ_i ← ∘(t̂_i + a_i)
6:     μ ← |x|μ + |ŷ_i|
7: end for
8: μ ← 2^{−p}(2μ − |ŷ_0|)
```

Dans la dichotomie naïve le coût de l'évaluation du signe du polynôme à chaque étape est trop élevé. Le signe est calculé en évaluant le polynôme en arithmétique entière ce qui induit une grande précision de calcul, alors que seul un bit du résultat est nécessaire (son signe). L'idée de la dichotomie tronquée est de calculer $P\left(\frac{c}{2^k}\right)$ avec une précision moindre mais suffisante pour pouvoir déterminer son signe.

L'algorithme 16 décrit l'évaluation polynomiale de Horner avec calcul d'une borne d'erreur sur le résultat final. Si la borne d'erreur est plus petite (en valeur absolue) que ce résultat, alors en particulier on a déterminé le signe de $P(x)$ avec certitude.

On dénote par y_i la valeur de $\widehat{y_i}$ si les calculs dans l'algorithme 16 se faisaient en précision infinie, sans erreur d'arrondi, et de même pour les autres quantités.

La justification du calcul de la borne d'erreur μ se trouve dans [21, p. 95], nous la reproduisons ici en l'adaptant à nos notations. On définit la suite $(\pi_i)_{0\leq i\leq n}$ par

$$\begin{aligned}\pi_n &= 0\\ \pi_i &= |x|\pi_{i+1} + |x|\cdot|\widehat{y}_{i+1}| + |\widehat{y}_i|\end{aligned}$$

L'utilisation d'une arithmétique flottante avec arrondi correct donne :

$$\begin{aligned}\widehat{t_i} &= (1+\theta_i)x\widehat{y_{i+1}}\\ \widehat{y_i} &= (1+\theta'_i)(\widehat{t_i}+a_i)\\ &= (1+\theta'_i)a_i + (1+\theta'_i)(1+\theta_i)x\widehat{y_{i+1}} \qquad (5.2)\\ (1+\epsilon_i)\widehat{y_i} &= x\widehat{y_{i+1}}(1+\theta_i) + a_i \qquad (5.3)\end{aligned}$$

où $|\theta_i|, |\theta'_i|, |\epsilon_i| \leq 2^{-p}$.

L'objectif est de borner $|\widehat{y_0} - y_0|$. On pose donc $f_i = \widehat{y_i} - y_i$ pour obtenir de l'équation (5.2) :

$$\begin{aligned}y_i + f_i + \epsilon_i\widehat{y_i} &= x(y_{i+1}+f_{i+1}) + x\widehat{y_{i+1}}\theta_i + a_i\\ f_i &= xf_{i+1} + x\widehat{y_{i+1}}\theta_i - \epsilon_i\widehat{y_i}\\ f_n &= 0\\ |f_i| &\leq |x|\cdot|f_{i+1}| + 2^{-p}(|x|\cdot|\widehat{y_{i+1}}| + |\widehat{y_i}|)\end{aligned}$$

d'où l'on conclut par récurrence que $|f_i| \leq 2^{-p}\pi_i$. Plutôt que de calculer π_i dans l'algorithme, on définit la suite μ_i par

$$\mu_i = \frac{1}{2}(\pi_i + |\widehat{y_i}|)$$

qui satisfait les équations suivantes :

$$\begin{aligned}\mu_n &= \frac{1}{2}|\widehat{y_n}|\\ \mu_i &= |x|\mu_{i+1} + |\widehat{y_i}|.\end{aligned}$$

Dans l'algorithme 16 on effectue les calculs concernant μ (lignes 2, 6 et 8) avec le mode d'arrondi vers le haut, pour que la borne calculée soit valide.

Lorsque cette borne μ sur l'erreur est plus grande en valeur absolue que le résultat retourné $\widehat{y_0}$ il n'est pas possible de déterminer le signe de P au point considéré et il est nécessaire d'augmenter la précision de calcul utilisée. Dans l'algorithme de recherche dichotomique 17, on choisit d'augmenter la précision de $\log_2|\frac{\mu}{\widehat{y_0}}| + 1$ pour espérer compenser les erreurs d'arrondi à la tentative suivante (ce choix correspond à exactement le nombre de « bits manquants » à l'évaluation qui a échoué).

Nous poursuivons l'analyse de l'erreur de l'évaluation de $P(x)$ par Horner_p pour déterminer comment choisir la précision de calcul lors du raffinement dichotomique.

En développant l'équation (5.3) pour $\widehat{y_0}$:

$$\begin{aligned}\widehat{y_0} &= (1+\theta'_0)a_0 + (1+\theta'_0)(1+\theta_0)x\widehat{y_1}\\ &= \sum_{i=0}^{n} a_i x^i \left(\prod_{j=0}^{i}(1+\theta'_j)\prod_{j=0}^{i-1}(1+\theta_j)\right).\end{aligned}$$

Un lemme intermédiaire est utile pour simplifier les calculs, il est extrait de [21] :

Lemme 15. *Soient u, n et $\theta_1, \theta_2, \ldots, \theta_n$ tels que $|\theta_i| \leq u$ pour $i \in \{1, \ldots, n\}$ et tels que $nu < 0,01$. Alors*

$$\prod_{i=1}^{n}(1+\theta_i) = 1+\delta$$

avec $|\delta| \leq 1,01nu$.

PREUVE : $\delta = \prod_{i=1}^{n}(1+\theta_i) - 1$. On encadre δ par

$$(1-u)^n - 1 \leq \delta \leq (1+u)^n - 1$$

et donc

$$|\delta| \leq \max\left\{(1+u)^n - 1, 1-(1-u)^n\right\}.$$

On vérifie que $(1+u)^n - 1 \geq 1-(1-u)^n$:

$$\begin{aligned}
(1+u)^n - 1 - [1-(1-u)^n] &= (1+u)^n + (1-u)^n - 2 \\
&= \sum_{i=0}^{n}\binom{n}{i}u^i + \sum_{i=0}^{n}\binom{n}{i}(-u)^i - 2 \\
&= 2\sum_{\substack{i=0,\\ i \text{ pair}}}^{n}\binom{n}{i}u^i - 2 \\
&= 2\sum_{\substack{i=2,\\ i \text{ pair}}}^{n}\binom{n}{i}u^i \\
&\geq 0.
\end{aligned}$$

Avec $1+u \leq \exp(u)$ on obtient

$$(1+u)^n - 1 \leq \exp(nu) - 1$$

que l'on développe en

$$\begin{aligned}
(1+u)^n - 1 &\leq nu + \frac{(nu)^2}{2} + \frac{(nu)^3}{3!} + \frac{(nu)^4}{4!} + \ldots \\
&\leq nu\left(1 + \frac{nu}{2} + \frac{(nu)^2}{3!} + \frac{(nu)^3}{4!} + \ldots\right) \\
&\leq nu\left(1 + \frac{nu}{2} + \left(\frac{nu}{2}\right)^2 + \left(\frac{nu}{2}\right)^3 + \ldots\right) \\
&\leq nu\frac{1}{1-\frac{nu}{2}} \\
&\leq 1,01nu.
\end{aligned}$$

□

On suppose que $(2n+1)2^{-p} \leq 0,01$ et on utilise le lemme 15 pour obtenir

$$\widehat{y_0} = \sum_{i=0}^{n}(1+\lambda_i)a_i x^i$$

où $|\lambda_i| \leq 1,01(2i+1)2^{-p}$. On peut considérer que l'évaluation en flottants du polynôme P tronqué est équivalent à l'évaluation exacte d'un polynôme $\widehat{P}$ dont les coefficients sont perturbés :

$$\widehat{P}(X) = \sum_{i=0}^{n}(1+\lambda_i)a_i X^i.$$

Sur l'intervalle $[a,b]$ d'isolation de départ les fonctions $(x \mapsto x^i)_{i=0,\ldots n}$ sont bornées en valeur absolue par une certaine constante C. De même les coefficients $a_0, a_1, \ldots, a_n$ de P sont majorés en valeur absolue par une certaine constante A. L'erreur globale sur l'évaluation tronquée de P peut donc se majorer ainsi :

$$\begin{aligned}
|P(x) - \widehat{y_0}| &= |P(x) - \widehat{P}(x)| \\
&= \left|\sum_{i=0}^{n} a_i\lambda_i x^i\right| \\
&\leq C \cdot A \sum_{i=0}^{n} |\lambda_i| \\
&\leq 1,01C \cdot A 2^{-p} \sum_{i=0}^{n}(2i+1) \\
&\leq 1,01C \cdot A(n+1)^2 2^{-p}.
\end{aligned}$$

D'autre part au voisinage de u racine simple :

$$P(x) = (x-u)P'(u) + \mathcal{O}((x-u)^2)$$

ce qui montre que l'ordre de grandeur de $|P(x)|$ est essentiellement le même que celui de $|x-u|$ à un facteur multiplicatif inconnu près.

Pour pouvoir continuer à déterminer avec certitude le signe de $P(x)$ il est nécessaire de faire décroître l'erreur globale due aux arrondis au moins aussi vite que la valeur de $P(x)$ décroît ; il est donc légitime de supposer que la précision flottante requise dans le calcul de $P(x)$ est du type

$$C_{P,u} - \log_2 |u-x|$$

où $C_{P,u}$ est une constante dépendant uniquement de P et de la racine considérée. On augmente donc la précision de calcul dans l'algorithme 16 d'un bit à chaque étape de la dichotomie de l'algorithme 17.

On rappelle que l'algorithme 17 est exécuté avec pour paramètres initiaux $a = \frac{c}{2^k}$ et $b = \frac{c+1}{2^k}$ donnés tous les deux en nombres flottants avec une précision suffisante pour être exacts. Au cours de l'algorithme, p désigne la précision en bit déjà connue sur la racine recherchée, et G le nombre de bits de garde utilisés pour compenser les erreurs d'arrondi.

Le coût de l'évaluation de P par la méthode de Horner en précision $p+G$ est de n multiplications de flottants de taille $p+G$ par des entiers de taille au plus k bits (les additions sont négligeables) avec un résultat flottant de taille $p+G$ bits. En moyenne il suffit de calculer cette multiplication avec une taille à peine plus grande que $p+G$ bits pour pouvoir arrondir ensuite ; sur des entrées aléatoires la probabilité que la multiplication nécessite une taille plus grande (mais toujours bornée par la taille du produit complet en $p+G+k$ bits) décroît exponentiellement et on peut considérer qu'en moyenne le coût total de l'algorithme 16 est $\mathcal{O}(nM(p+G))$.

La stratégie d'augmentation du nombre G de bits de garde est choisie de telle sorte que l'algorithme 16 ne devrait échouer qu'une seule fois de suite. Ce comportement est vérifié en

Algorithme 17 Raffinement avec dichotomie tronquée

ENTRÉES : f, a, b tels que $f(a)f(b) \leq 0$, ε.
SORTIE : a', b' tels que $f(a')f(b') \leq 0$, $|b' - a'| \leq \varepsilon$ et $[a', b'] \subseteq [a, b]$.

$sa \leftarrow \text{sign}(f(a))$
$sb \leftarrow \text{sign}(f(b))$
$a' \leftarrow a$
$b' \leftarrow b$
$p \leftarrow -\log_2(b - a)$
$G \leftarrow 1$
while $b' - a' > \varepsilon$ **do**
    $m \leftarrow \frac{a'+b'}{2}$ ▷ Sans erreur d'arrondi en ajustant la précision de m.
    $y, \mu \leftarrow \text{Horner}_{p+G}(P, m)$
    if $\mu \geq |y|$ **then**
        $G \leftarrow G + \log_2 |\frac{\mu}{y}| + 1$ ▷ Précision de calcul insuffisante.
        next
    end if
    if $sa = \text{sign}(y)$ **then**
        $a' \leftarrow m$
    else
        $b' \leftarrow m$
    end if
    $p \leftarrow p + 1$
end while
return (a', b')

pratique, mais nous l'admettons ici sans preuve pour l'étude de la complexité. On peut donc supposer que tout l'algorithme 17 se déroule avec une valeur de G constante et égale à sa valeur finale dans l'exécution réelle : on ne néglige ainsi qu'un facteur constant.

Avec $\log_2(b-a) - \log_2(\varepsilon)$ passages dans la boucle `while` la complexité est

$$\mathcal{O}\left(\sum_{p=-\log_2 b-a}^{-\log_2 \varepsilon} nM(p+G)\right)$$

d'où l'on déduit une complexité globale pour la dichotomie en

$$\mathcal{O}\left(-n\log\varepsilon M(G-\log\varepsilon)\right).$$

18.2 Newton

L'itération de Newton est une méthode de calcul de racine fréquemment utilisée, aussi bien pour le raffinement de racine polynomiale que pour la résolution d'équations plus générales. On parle d'itération car la méthode calcule une suite de valeurs approchant toujours mieux la solution ciblée.

Elle se base sur le développement de Taylor de la fonction f dont on cherche une racine simple u au voisinage du point courant de l'itération x_n :

$$0 = f(u) = f(x_n) + (u - x_n)f'(x_n) + \mathcal{O}((u-x_n)^2). \tag{5.4}$$

Si $|u - x_n|$ est suffisamment petit, on dit qu'on se trouve dans la « zone de convergence » et alors la quantité reste est supposée négligeable dans l'équation (5.4). On obtient alors :

$$u \approx x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.5}$$

et on prend $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ comme nouvelle approximation de u, à condition que $f'(x_n) \neq 0$.

La convergence de la méthode de Newton est quadratique comme montré plus bas. Il est classique de voir la méthode de Newton comme le calcul d'une suite récurrente dont on cherche le point fixe. En effet notons

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

alors

$$\begin{aligned} x_{n+1} &= \varphi(x_n) \\ \varphi(u) &= u \\ \varphi'(u) &= 0 \\ \varphi''(u) &= \frac{f''(u)}{f'(u)}. \end{aligned}$$

En développant φ au voisinage de u on obtient

$$\varphi(x) = \varphi(u) + (x-u)\varphi'(u) + \frac{(x-u)^2}{2}\varphi''(u) + \mathcal{O}((x-u)^3)$$

et en spécialisant en x_n :

$$x_{n+1} - u = \frac{(x_n-u)^2}{2}\frac{f''(u)}{f'(u)} + \mathcal{O}((x_n-u)^3). \tag{5.6}$$

En se plaçant dans un intervalle suffisamment proche de u pour que le terme reste soit négligeable et tel que f' ne s'annule pas, on observe bien la convergence quadratique. Deux implémentations de la méthode sont utilisés dans CRQ : une version scalaire, et une version par intervalles.

Newton scalaire

L'implémentation de la méthode de Newton décrite ci-dessus conduit naturellement à l'algorithme 18. On l'appelle « scalaire » car chaque étape de l'itération manipule une approximation flottante de la racine ; ceci pour la distinguer de la méthode de Newton par intervalles présentée p. 76.

Algorithme 18 Itération de Newton scalaire

ENTRÉE : $x_0, f, f', p, p_{\text{fin}}$ tel que $|x_0 - u| \leq 2^{-p}|x_0|$ où u est la racine recherchée.
SORTIE : x.

1: $x \leftarrow \circ_p(x_0)$
2: **while** $p < p_{\text{fin}}$ **do**
3: $\quad y \leftarrow \circ_p(f(x))$
4: $\quad z \leftarrow \circ_p(f'(x))$
5: $\quad d \leftarrow \circ_p(\frac{y}{z})$
6: $\quad x \leftarrow \circ_{2p}(x - d)$
7: $\quad p \leftarrow 2p$
8: **end while**
9: **return** x

Dans l'algorithme 18 on appelle d la correction apportée à la valeur approchée courante de la racine cible, et on estime que la précision relative sur u double à chaque étape. On s'arrête lorsque la précision de calcul courante dépasse la précision cible p_{fin}.

Une telle implémentation naïve a malheureusement peu de chance d'être correcte car nombre de détails sont passés sous silence. Selon la valeur initiale x_0 de l'itération il n'est pas garanti qu'on se trouve dans la zone de convergence de l'algorithme, ni même qu'on y reste à chaque itération. Il est tout à fait possible de diverger à l'infini ou de converger vers une autre racine de f, ce que l'on observe en pratique dans une telle implémentation naïve.

Ensuite il n'est pas évident *a priori* que le critère d'arrêt soit correct et garantisse une borne sur l'erreur finale commise $|x - u|$; c'est quelque chose à justifier.

Enfin chaque opération se fait en calcul flottant multi-précision, il faut donc gérer les erreurs d'arrondi et décider de la précision à laquelle effectuer les calculs à chaque étape. Ce choix de précision de travail s'avère être crucial en pratique pour bénéficier effectivement du caractère quadratique de la convergence de la méthode.

Analyse d'erreur – erreurs d'arrondi

Dans le cas que l'on traite, $f = P$ est un polynôme évalué par la méthode de Horner décrite dans l'algorithme 16 dont on peut contrôler dynamiquement l'erreur absolue, de même pour f'.

On appelle $\widehat{y}$ la valeur effectivement calculée pour y par l'algorithme (de même pour z et d)

et on peut écrire :

$$\begin{aligned} \widehat{y} &= (1+\theta_y)y \\ \widehat{z} &= (1+\theta_z)z \\ \widehat{d} &= (1+\theta_d)\frac{\widehat{y}}{\widehat{z}} \end{aligned}$$

où des bornes sur $|\theta_y|$ et $|\theta_z|$ sont données par la méthode de Horner. Une borne sur $|\theta_d|$ se déduit de la précision de calcul courante p : $|\theta_d| \leq 2^{-p}$.

En combinant ces trois équations on a

$$\widehat{d} = (1+\theta_d)\frac{1+\theta_y}{1+\theta_z}d = (1+\theta')d$$

où une borne sur $|\theta'|$ se calcule dynamiquement en fonction des bornes sur $|\theta_d|$, $|\theta_y|$ et $|\theta_z|$. L'équation (5.6) nous permet de déterminer les précisions attendues pour chacun des termes. Notons $\lambda = |\frac{f''(u)}{2f'(u)}|$. En passant au logarithme :

$$-\log_2|x_{n+1}-u| \approx -2\log_2|x_n-u| - \log_2\lambda. \tag{5.7}$$

Le nombre (inconnu) de bits corrects après la virgule dans x_n comme valeur approchée de u est donné par $-\log_2|x_n-u|$. En notant $p_n = E(x_n) - \log_2|x_n-u|$ la précision relative en bits sur x_n et sachant qu'au voisinage de u on a $E(x_n) = E(u)$ l'équation (5.7) devient :

$$p_{n+1} + \mu \approx 2(p_n + \mu) \tag{5.8}$$

où $\mu = -\log_2\lambda - E(u)$ est une constante qui ne dépend que de f et de u. La figure 5.3 montre

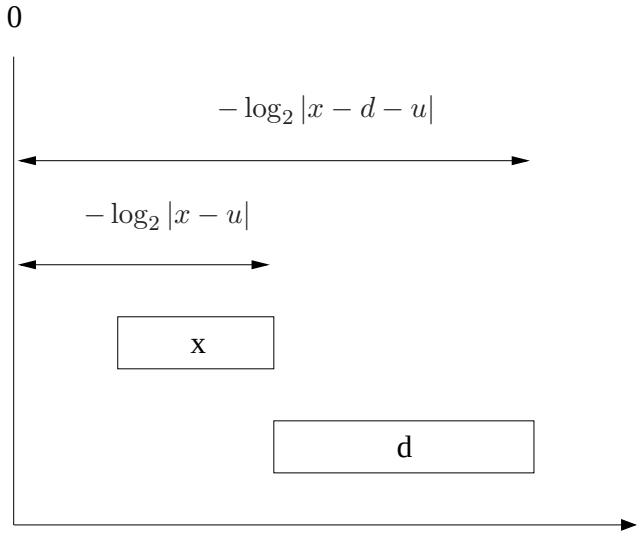

FIG. 5.3 – Une étape de l'itération de Newton dans une situation idéale, où le nombre de bits corrects double à chaque étape et où l'ajustement d de l'étape courante est calculé avec la précision optimale.

la situation lors d'une étape de l'itération de Newton dans un cas de figure idéal où :

1. les bits de la mantisse de x sont tous corrects,
2. l'ajustement d est calculé avec la précision optimale, à savoir que tous les bits de la mantisse de d sont significatifs et contribuent à améliorer la précision de x,
3. il n'y a aucun chevauchement des mantisses de x et de d et le nombre de bits corrects double à cette étape (c'est une conséquence de 1).

Toutes ces hypothèses favorables ne sont pas vérifiées en général, aussi pour écrire une méthode de Newton correcte et efficace il faut rajouter le nécessaire pour en tenir compte, essentiellement par le biais d'heuristiques.

Algorithme 19 Itération de Newton – avec heuristiques explicites

ENTRÉE : $a, b, f, f', p_{\text{fin}}$.
SORTIE : x.

$x_0 \leftarrow \frac{a+b}{2}$
$p_0 \leftarrow \log_2 |x_0| - \log_2(b-a)$
$n \leftarrow 0$
$\mu \leftarrow 0$
while $p_n < p_{\text{fin}}$ **do**
  $y \leftarrow \circ_{p_n+\mu}(f(x_n))$
  $z \leftarrow \circ_{p_n+\mu}(f'(x_n))$
  $d \leftarrow \circ_{p_n+\mu}(\frac{y}{z})$
  $c \leftarrow p_n - (\log_2 |x_n| - \log_2 |d|)$ ▷ Écart par rapport à la précision estimée de x_n.
  $\mu \leftarrow \mu + c$
  $p_n \leftarrow p_n + c$
  $p_{n+1} \leftarrow 2p_n + \mu$
  $x_{n+1} \leftarrow \circ_{p_{n+1}}(x_n - d)$
  $n \leftarrow n + 1$
end while
return x

L'équation (5.5) montre que dans la zone de convergence la correction d apportée à x_n est une bonne estimation de l'erreur $|x_n - u|$. On considère donc que l'écart d'exposant $E(x_n) - E(d)$ donne le nombre de bits corrects dans la mantisse de x_n ou de manière équivalente que l'erreur sur x_n est de $2^{E(d)}$. Cela nous permet de comparer la valeur de p_n mesurée avec la valeur de p_n calculée par application de l'équation (5.8) ; l'écart observé est utilisé pour corriger μ et mieux prédire p_{n+1}. Par le même argument on s'attend à ce qu'au plus $p_n + \mu$ bits de d soient utiles, c'est donc la précision relative à laquelle on calcule $f(x_n)$ et $f'(x_n)$.

Ces choix dans les précisions de calcul visent à faire le moins de travail à chaque étape, tout en conservant la vitesse de convergence prédite par la méthode de Newton. L'algorithme 19 détaille la mise en œuvre de ces heuristiques.

Newton par intervalles

Une grande partie de cette sous-section est extraite de [30].

La méthode « générique » de raffinement de Newton par intervalles opère à chaque étape sur une liste $\mathcal{L}$ d'intervalles contenant potentiellement des racines de f. À partir d'une initialisation adéquate de $\mathcal{L}$ la méthode renvoie en sortie une nouvelle liste d'intervalles contenant tous au plus une racine et dont le diamètre est plus petit qu'un paramètre choisi à l'avance ; modulo des

intervalles « faux positifs » apparaissant du fait de l'évaluation par intervalles de f l'algorithme réalise donc à la fois l'isolation et le raffinement des racines. Dans notre cas des polynômes de Legendre l'isolation est réalisée par l'algorithme d'isolation décrit en section 17 (*a priori* plus efficace) et on s'intéresse donc dans la suite uniquement à l'aspect raffinement de la méthode de Newton par intervalles.

Appelons X_k l'intervalle d'isolation de l'étape courante. On calcule un intervalle $[\alpha, \beta]$ contenant $f'(X_k) = \{f'(x) \mid x \in X_k\}$. On choisit $x_k \in X_k$. Si

$$\forall y \in X_k, \alpha \leq f'(y) \leq \beta$$

alors d'une part pour $y \in X_k, y \leq x_k$:

$$\begin{aligned} \alpha(x_k - y) \leq & \int_y^{x_k} f'(t)dt & \leq \beta(x_k - y) \\ f(x_k) + \beta(y - x_k) \leq & \quad f(y) & \leq f(x_k) + \alpha(y - x_k) \end{aligned}$$

et d'autre part pour $y \in X_k, y \geq x_k$:

$$\begin{aligned} \alpha(y - x_k) \leq & \int_{x_k}^{y} f'(t)dt & \leq \beta(y - x_k) \\ f(x_k) + \alpha(y - x_k) \leq & \quad f(y) & \leq f(x_k) + \beta(y - x_k). \end{aligned}$$

Si on suppose que $u \leq x_k$ avec $f(u) = 0$ on obtient à la racine

$$f(x_k) + \beta(u - x_k) \leq 0$$

et

$$f(x_k) + \alpha(u - x_k) \geq 0.$$

Supposons que α et β sont de même signe, par exemple positif et alors

$$u \in \left[x_k - \frac{f(x_k)}{\alpha}, x_k - \frac{f(x_k)}{\beta}\right]$$

ce qui s'écrit de façon plus concise par intervalles :

$$u \in x_k - \frac{f(x_k)}{f'(X_k)}. \tag{5.9}$$

On vérifie facilement que l'équation (5.9) reste valide dans tous les cas possibles pour les signes de α et β et aussi si $u > x_k$; en particulier lorsque $f'(X_k) \ni 0$ l'ensemble calculé est constitué de deux intervalles infinis. Si $\alpha < 0 < \beta$ on utilise la division étendue par intervalles

$$\frac{1}{[\alpha, \beta]} = \left]-\infty, \frac{1}{\alpha}\right] \bigcup \left[\frac{1}{\beta}, +\infty\right[.$$

De façon plus imagée la fonction f est bornée par un cône défini par les tangentes de pentes extrêmes de f sur X_k, et la racine u de f est à chercher dans l'intersection de ce cône avec l'axe des abscisses (cf. figure[4] 5.4).

[4] Généreusement fournie par Nathalie Revol.

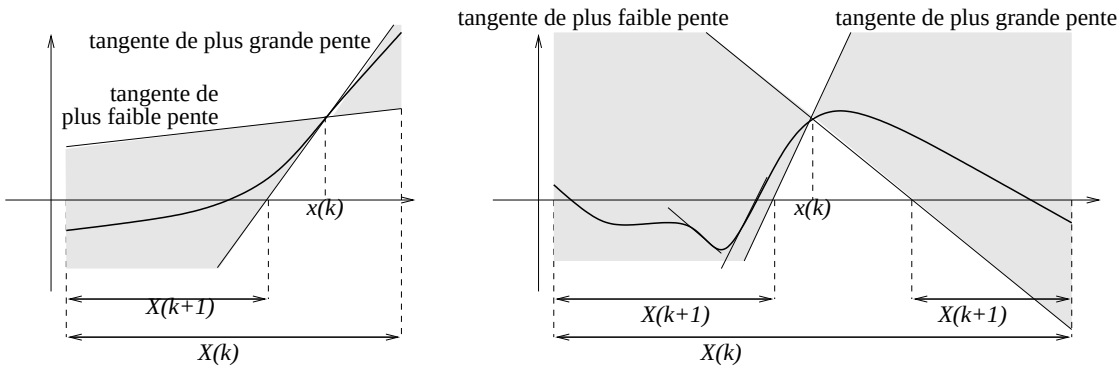

FIG. 5.4 – Une interprétation graphique de la méthode de Newton par intervalles.

La nouvelle valeur X_{k+1} de l'itération est donc définie par

$$X_{k+1} = X_k \bigcap \left(x_k - \frac{f(x_k)}{f'(X_k)} \right) \tag{5.10}$$

où $f(x_k)$ est représenté par un intervalle dont le centre est la valeur flottante calculée pour f en x_k et dont le rayon est la borne sur l'erreur associée. L'algorithme s'arrête lorsque l'intervalle de recherche est plus petit que la précision demandée par l'utilisateur (en toute généralité on devrait considérer une liste d'intervalles, mais comme expliqué plus bas on peut se ramener à un seul intervalle à chaque étape).

Si l'itération ainsi décrite est implémentée de façon naïve il n'est pas garanti qu'elle termine, ni qu'elle converge avec la vitesse attendue. Il se peut que $x_k - \frac{f(x_k)}{f'(X_k)} \supseteq X_k$ à cause des erreurs d'arrondis qui produisent des intervalles trop grands pour $f(x_k)$. Quelques explications à propos de l'algorithme sont requises. Lorsque $f'(X_k) \ni 0$, les deux intervalles produits à cette étape sont de part et d'autre de x_k et le signe de $f(x_k)$ permet de décider lequel conserver (on sait que l'un des deux uniquement convient car on a isolé les racines). Puisque l'on peut prendre x_k arbitraire dans X_k on choisit de prendre le milieu de X_k pour diviser l'intervalle de recherche par deux au moins dans ce cas.

On rappelle qu'on note $w(X)$ la largeur d'un segment X de $\mathbb{R}$. Pour garantir la terminaison de l'algorithme, on modifie l'algorithme naïf de la façon suivante :

- si l'intervalle calculé pour $f(x_k)$ contient 0 alors la précision de calcul courante est insuffisante et on l'augmente. Ce processus termine dans la mesure où l'on finirait par calculer $f(x_k)$ exactement : on rappelle que f est un polynôme dans notre cas ;
- si $w(X_{k+1}) \geq \frac{1}{2} w(X_k)$ on remplace X_{k+1} tel que calculé par l'équation (5.10) par l'application d'une étape de dichotomie déterminée par le signe de $f(x_k)$ et par le signe (connu) des bornes.

Lorsqu'on se trouve suffisamment près de la racine cherchée, on sait que f' ne s'annule pas. En représentant un intervalle X_k par son centre et une borne sur l'erreur relative on constate que l'utilisation d'une méthode par intervalles ne change pas significativement les propriétés de convergence de la méthode de Newton ; aussi les heuristiques en termes de précision de calcul et précision attendue sur X_{k+1} sont les mêmes que dans la version de l'itération de Newton sans intervalle. Une différence appréciable par rapport à la méthode « scalaire » est que le nombre e_n de bits corrects à l'étape n n'est pas estimé *a posteriori* mais connu directement comme $-\log_2(w(X_n))$. Ces idées sont mises en œuvre dans l'algorithme 20.

Algorithme 20 Itération de Newton par intervalles

ENTRÉE : $[a, b], f, f', \varepsilon$.
SORTIE : x tel que $|x - u| \leq \varepsilon$.

$X \leftarrow [a, b]$
$\mu \leftarrow 0$
$g \leftarrow 1$
while $w(X) > \varepsilon$ **do**
  $c \leftarrow -\log_2(w(X))$ ▷ Nombre de bits corrects pour X_k
  $e \leftarrow 2c - \mu$ ▷ Nombre espéré de bits corrects pour X_{k+1}
  $x \leftarrow \text{mid}(X)$
  $Y \leftarrow \circ_{e+g}(f(x))$
  $Z \leftarrow \circ_{e+g}(f'(x))$
  if $Y \ni 0$ **then**
    $g \leftarrow 2g$ ▷ On augmente le nombre de bits de garde.
    next
  end if
  $D \leftarrow \circ_{e+g}(x - Y/Z)$ ▷ Si $Z \ni 0$ on choisit le bon intervalle pour D.
  $U \leftarrow \circ_{e+g}(X \bigcap D)$
  if $w(U) \geq \frac{1}{2}w(X)$ **then**
    $X \leftarrow \text{dicho}(X)$
    next
  end if
  $X \leftarrow \circ_e(U)$
  $d \leftarrow -\log_2(w(X))$
  $\mu \leftarrow 2d - c$
end while
return x

18.3 Méthode de la sécante

La méthode de raffinement dite « de la sécante » procède par itération comme la méthode de Newton, mais d'ordre deux. Une description peut s'en trouver par exemple dans [27, §9-3].

Si x_n et x_{n+1} sont les deux dernières valeurs calculées par la méthode on définit le terme suivant x_{n+2} par :

$$x_{n+2} = \frac{x_{n+1}f(x_n) - x_n f(x_{n+1})}{f(x_n) - f(x_{n+1})}. \tag{5.11}$$

Ce terme est bien défini lorsque $f(x_{n+1}) \neq f(x_n)$ et s'interprète comme le point d'intersection entre la droite passant par les points $(x_n, f(x_n))$ et $(x_{n+1}, f(x_{n+1}))$, et l'axe des abscisses.

Une étape de l'itération est illustrée en figure 5.7.

Pour l'analyse de la méthode on se place dans un voisinage de u où f' ne s'annule pas ce qui permet d'inverser localement la fonction : soit $\varphi(y) = f^{-1}(y) - u$, et notons $y_n = f(x_n)$. On s'intéresse à la quantité $x_n - u$ pour estimer la vitesse de convergence ; l'équation (5.11) nous donne :

$$\begin{aligned} x_{n+2} - u &= \frac{\varphi(y_{n+1})y_n - \varphi(y_n)y_{n+1}}{y_n - y_{n+1}} \\ &= -y_n y_{n+1} \frac{\frac{\varphi(y_{n+1})}{y_{n+1}} - \frac{\varphi(y_n)}{y_n}}{y_{n+1} - y_n}. \end{aligned}$$

Posons $g(y) = \frac{\varphi(y)}{y}$, alors

$$\begin{aligned} \frac{g(y_{n+1}) - g(y_n)}{y_{n+1} - y_n} &= \frac{g(y_{n+1}) - g(0)}{y_{n+1}} \frac{y_{n+1}}{y_{n+1} - y_n} - \frac{g(y_n) - g(0)}{y_n} \frac{y_n}{y_{n+1} - y_n} \\ &= \big(g'(0) + \mathcal{O}(y_{n+1})\big) \frac{y_{n+1}}{y_{n+1} - y_n} + \big(g'(0) + \mathcal{O}(y_n)\big) \frac{y_n}{y_{n+1} - y_n} \\ &= g'(0) + \mathcal{O}\left(\frac{\max(y_n^2, y_{n+1}^2)}{y_{n+1} - y_n}\right). \end{aligned}$$

La quantité $\mathcal{O}\left(\frac{\max(y_n^2, y_{n+1}^2)}{y_{n+1} - y_n}\right)$ est effectivement négligeable si l'on suppose que les deux valeurs calculées précédentes x_n et x_{n+1} ne sont pas beaucoup plus proches l'une de l'autre qu'elles ne sont proches de la racine u cherchée ; cela revient à supposer

$$|x_{n+1} - x_n| \gg \max((x_n - u)^2, (x_{n+1} - u)^2).$$

En développant aux ordres qui conviennent on trouve

$$\begin{aligned} g'(y) &= \frac{\varphi'(y)y - \varphi(y)}{y^2} \\ &= \frac{y(\varphi'(0) + y\varphi''(0)) - (y\varphi'(0) + \frac{y^2}{2}\varphi''(0)) + \mathcal{O}(y^3)}{y^2} \\ &= \frac{1}{2}\varphi''(0) + \mathcal{O}(y) \end{aligned}$$

ce qui donne

$$\begin{aligned} x_{n+2} - u &= -\frac{\varphi''(0)}{2} y_n y_{n+1} + \mathcal{O}\left(\frac{\max(y_n^2, y_{n+1}^2)}{y_{n+1} - y_n}\right) \\ y_n &= f'(u)(x_n - u) + \mathcal{O}(x_n - u) \\ y_{n+1}y_n &= f'^2(u)(x_n - u)(x_{n+1} - u) + \mathcal{O}((x_n - u)(x_{n+1} - u)). \end{aligned}$$

En développant $\varphi(y_n) = x_n - u = \varphi(0) + \mathcal{O}(y_n) = \mathcal{O}(y_n)$ on constate que le terme $\mathcal{O}((x_n - u)(x_{n+1} - u))$ est absorbé dans $\mathcal{O}(\max(y_n^2, y_{n+1}^2))$ donc il est *a fortiori* absorbé dans la quantité $\mathcal{O}\left(\frac{\max(y_n^2, y_{n+1}^2)}{y_{n+1} - y_n}\right)$. On obtient

$$x_{n+2} - u = -\frac{\varphi''(0)}{2} f'^2(u)(x_n - u)(x_{n+1} - u) + \mathcal{O}\left(\frac{\max(y_n^2, y_{n+1}^2)}{y_{n+1} - y_n}\right).$$

La définition de φ permet d'écrire

$$\begin{aligned} \varphi'(y) &= \frac{1}{f'(\varphi(y) + u)} \\ \varphi''(y) &= \frac{-f''(\varphi(y) + u)}{f'^2(\varphi(y) + u)} \varphi'(y) \\ &= -\frac{f''(\varphi(y) + u)}{f'^3(\varphi(y) + u)}. \end{aligned}$$

Au final l'équation clef pour étudier le comportement de la méthode de la sécante est

$$x_{n+2} - u \approx \frac{f''(u)}{2f'(u)}(x_{n+1} - u)(x_n - u). \tag{5.12}$$

En passant au logarithme,

$$-\log_2 |x_{n+2} - u| \approx -\log_2 |x_{n+1} - u| - \log_2 |x_n - u| - \log_2 \left|\frac{f''(u)}{2f'(u)}\right| \tag{5.13}$$

montre que la suite $(-\log_2 \left|\frac{f''(u)}{2f'(u)}\right| - \log_2 |x_n - u|)_{n \geq 0}$ se comporte asymptotiquement comme la suite

$$v_{n+2} = v_{n+1} + v_n$$

dont le terme générique est donné par

$$v_n = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^n + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^n.$$

On pose pour la suite $C = -\log_2 \left|\frac{f''(u)}{2f'(u)}\right|$. Sauf dans le cas de conditions initiales défavorables qui ne se produisent pas lorsque x_n et x_{n+1} sont assez proches de u, $\alpha \neq 0$ et le terme $\left(\frac{1+\sqrt{5}}{2}\right)^n$ domine asymptotiquement. À une constante près le nombre de bits corrects de x_n est donc multiplié par $\frac{1+\sqrt{5}}{2}$ à chaque étape, au lieu de 2 dans le cas de l'itération de Newton. Cette propriété de convergence *a priori* plus faible est compensée par un moindre coût de calcul à chaque étape (une évaluation de $f(x_n)$ au lieu d'une évaluation de $f(x_n)$ et de $f'(x_n)$). Nous donnons en fin de chapitre une comparaison théorique des complexités de ces deux méthodes de raffinement.

Il semble naturel de vouloir améliorer l'itération de la sécante en la modifiant comme suit :

- chaque étape de l'itération porte sur deux valeurs (a_n, b_n) au lieu d'une,
- l'initialisation se fait avec (a_0, b_0) tel que $f(a_0)f(b_0) < 0$,
- on applique une étape de la méthode de la sécante à (a_n, b_n) pour obtenir une nouvelle valeur m,

– le couple (a_{n+1}, b_{n+1}) est constitué de la nouvelle valeur m et de celle des deux valeurs précédentes choisie telle que $f(a_{n+1})f(b_{n+1}) < 0$,

l'idée étant de s'assurer d'une part que le calcul du prochain point est bien défini, et d'autre part de conserver un intervalle d'isolation de la racine permettant de conclure sur l'erreur de troncature finale. Cette méthode est aussi appelée « méthode de la fausse position » [14], [9, Ch 3].

L'initialisation ne pose pas de problème car l'algorithme d'Uspensky fournit un intervalle d'isolation qui vérifie bien la propriété recherchée. Un problème de cette approche est qu'elle peut ne pas converger. Dans le cas d'une fonction convexe par exemple on se retrouve dans la situation où l'une des bornes de l'intervalle d'isolation courant converge bien vers la racine recherchée alors que l'autre borne ne change jamais (voir figure 5.5). La convergence au sens de l'intervalle d'isolation n'est donc pas assurée.

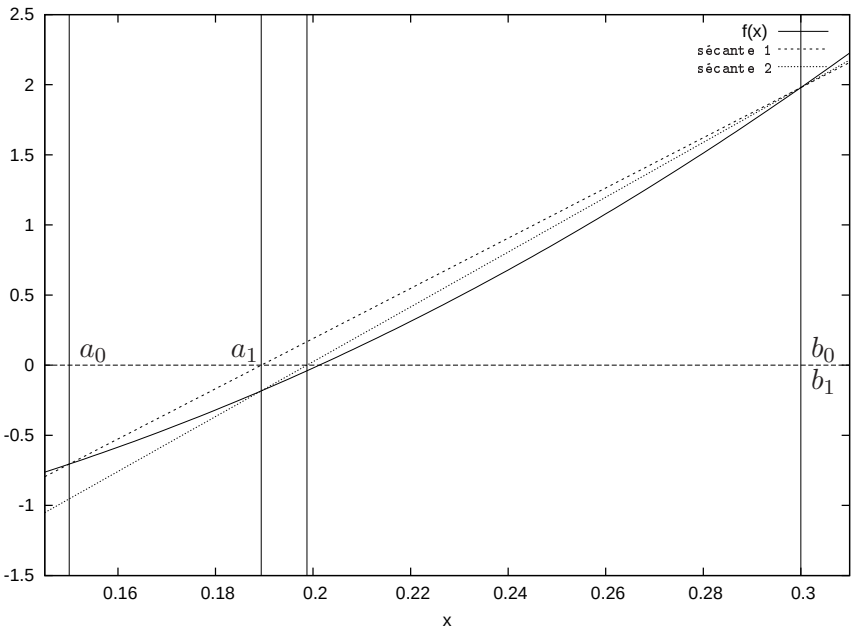

FIG. 5.5 – Une situation où la méthode de la « fausse position » ne converge pas au sens des intervalles : la borne inférieure de l'intervalle converge vers la racine mais la borne supérieure est constante. La taille de l'intervalle d'isolation est donc minorée.

Néanmoins il est possible d'utiliser la méthode de la sécante sans cette modification en calculant malgré tout un intervalle isolant, donc en ayant une garantie sur l'erreur finale commise sur la racine. À chaque étape, on calcule $f(x_n)$ en prenant soin de décider rigoureusement de son signe, ce qui permet de déterminer le signe de $x_n - u$ à partir du signe de f aux bornes de l'intervalle d'isolation initial $[a, b]$. En supposant que la méthode de raffinement atteigne la zone de convergence dans laquelle les hypothèses permettant d'établir (5.12) sont justifiées à partir d'un certain rang n_0, et qu'à cette étape particulière les signes de $x_{n_0} - u$ et $x_{n_0+1} - u$ sont équiprobables, alors le tableau 5.6 montre que dans 3/4 des cas les positions des points x_n et x_{n+1} par rapport à u décrivent un cycle de longueur 3, et que dans ce cycle 2 étapes sur 3 fournissent effectivement un intervalle d'isolation de u.

Un cas où $x_n - u$ est d'un signe constant à partir d'un certain rang est illustré en Figure 5.7 et s'explique par un argument de convexité. Il n'est pas non plus suffisant d'initialiser la méthode

$\frac{f''(u)}{f'(u)}$	$x_n - u$	$x_{n+1} - u$	$x_{n+2} - u$
$+$	$-$	$+$	$-$
$+$	$+$	$-$	$-$
$+$	$-$	$-$	$+$
$+$	$+$	$+$	$+$
$-$	$-$	$+$	$+$
$-$	$+$	$+$	$-$
$-$	$+$	$-$	$+$
$-$	$-$	$-$	$-$

FIG. 5.6 – L'évolution de la position de x_n par rapport à la racine u dans le domaine de convergence, découpée en cycles disjoints.

de la sécante avec un intervalle d'isolation $[x_0, x_1]$ pour espérer rester dans un des deux cycles favorables décrits dans le tableau 5.6. Prenons $\alpha > 0$, $f(x) = x^3 - \alpha^3$, $x_0 = -2$, $x_1 = 1$. On cherche à raffiner la racine $u = \alpha$. La formule (5.11) donne

$$x_2 = \frac{-8 - \alpha^3 - (-2)(1 - \alpha^3)}{-8 - 1} = \frac{-6 - 3\alpha^3}{-9} = \frac{2 + \alpha^3}{3}.$$

Pour $\alpha = 0,5$ on a bien $x_0 < \alpha < x_2 < x_1$ et un argument de convexité sur $[\alpha, x_2]$ montre qu'alors $x_n > \alpha$ pour $n \geq 1$ et la méthode ne produira donc pas un intervalle d'isolation convergeant vers la racine α.

Supposons que $f'f''$ ne change pas de signe sur $[x_n, x_{n+1}]$. On peut supposer sans perte de généralité que $f' > 0$ et $f'' > 0$. Ces hypothèses nous donnent :

$$f(x_n) < 0, x_n < u,$$

$$f(x_{n+1}) > 0, x_{n+1} > u.$$

Puisque f est convexe on sait que pour $t \in [0, 1]$, $f(t \cdot x_n + (1-t)x_{n+1}) \leq t \cdot f(x_n) + (1-t)f(x_{n+1})$. En particulier pour $x_{n+2} = \frac{x_{n+1}f(x_n) - x_n f(x_{n+1})}{f(x_n) - f(x_{n+1})}$ on a

$$\begin{aligned} f(x_{n+2}) &= f\left(\frac{x_{n+1}f(x_n) - x_n f(x_{n+1})}{f(x_n) - f(x_{n+1})}\right) \\ &\leq \frac{f(x_n)}{f(x_n) - f(x_{n+1})} f(x_{n+1}) - \frac{f(x_{n+1})}{f(x_n) - f(x_{n+1})} f(x_n) \\ &\leq 0 \end{aligned}$$

donc $x_{n+2} \leq u$. On a donc prouvé que la première ligne du tableau 5.6 est valide sous l'hypothèse que $f'f''$ garde un signe constant sur l'intervalle $[x_n, x_{n+1}]$. On prouve la troisième ligne en utilisant le fait que la fonction convexe f est supérieure à la corde définie par x_n et x_{n+1} en dehors du segment $[x_n, x_{n+1}]$. Le reste du tableau se prouve par des arguments similaires.

Théorème 12. *Soit $[x_0, x_1]$ un intervalle d'isolation d'une racine u de f, avec $f(x_0)f(x_1) < 0$. Si les points $(x_n)_{n \geq 2}$ calculés par la méthode de la sécante initialisée par x_0 et x_1 vérifient*

$$\forall n, x_i \in [x_0, x_1]$$

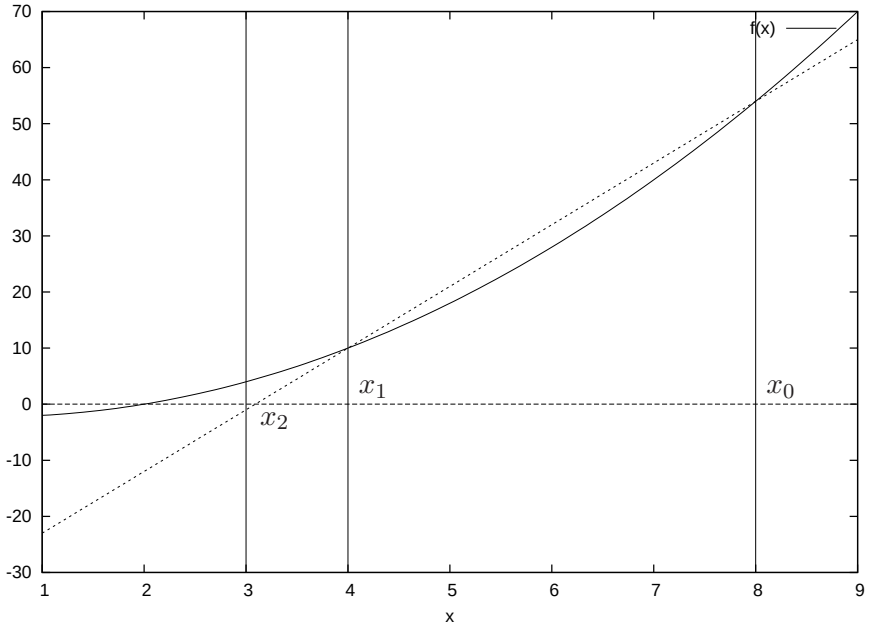

FIG. 5.7 – Une étape de l'itération de la sécante avec $f(x) = x^2 - x - 2$, $x_0 = 8$, $x_1 = 4$ et $u = 2$. Les points initiaux sont choisis loin de la racine pour améliorer la visibilité mais cela ne change pas le résultat.

et que $f''(x)f'(x)$ *ne change pas de signe sur* $[x_0, x_1]$*, alors il existe* $s \in \{1, 2\}$ *tel que*

$$(x_n - u)(x_{n+1} - u) \leq 0$$

pour tout $n \not\equiv s \bmod 3$*. De plus* $s = 1$ *si* $x_0 - u$ *a le même signe que* $f'(x)f''(x)$*, et* $s = 2$ *sinon.*

PREUVE : la preuve découle de la lecture du tableau 5.6, qui reste correct même si on ne se trouve pas *a priori* dans la zone de convergence. □

Pour utiliser cette propriété on peut isoler simultanément les racines de f, f' et f''. On obtient alors des intervalles d'isolation des racines de f sur lesquels f' et f'' n'ont pas de racine (donc gardent un signe constant).

Une autre solution est de constater que si au cours de l'algorithme le signe de $f(x_n)$ est le même pendant trois étapes consécutives, alors soit le domaine de convergence n'est pas encore atteint, soit il est atteint et on est rentré dans un des deux cycles défavorables. Dans les deux cas on peut abandonner la méthode de la sécante et faire une étape de dichotomie pour améliorer l'intervalle d'isolation et se rapprocher du domaine de convergence, tout en conservant un intervalle d'isolation de u. On garantit ainsi la convergence de la méthode, en espérant que la sécante ne sera pas trop souvent dégradée en une dichotomie. Si cela ne se produit pas trop souvent, cette stratégie peut être moins coûteuse que l'isolation des racines de f, f' et f''. C'est cette stratégie qui est utilisée dans CRQ.

Analyse d'erreur

On note comme pour la méthode de Newton $p_n = E(x_n) - \log_2 |x_n - u|$ la précision relative en bits sur x_n. Au voisinage de u on a toujours $E(x_n) = E(u)$, et l'équation (5.13) s'écrit

$$p_{n+2} + C' \approx (p_{n+1} + C') + (p_n + C') \tag{5.14}$$

où $C' = -\log_2 \left| \frac{f''(u)}{2f'(u)} \right| - E(u)$ est une constante qui ne dépend que de f et de u. On simplifie la suite de l'analyse en supposant $C = 0$ (le code de CRQ cependant ne fait pas cette hypothèse), ce qui reviendrait à considérer $q_n = p_n + C'$ dans la suite. L'équation (5.14) permet d'anticiper le nombre de bits corrects à attendre pour x_{n+2}, et donc d'ajuster la précision de calcul utilisée.

Récrivons l'équation (5.11) en

$$\begin{aligned} x_{n+2} &= x_{n+1} - \delta_n \\ \delta_n &= f(x_{n+1}) \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)} \end{aligned}$$

pour faire apparaître comme dans l'itération de Newton l'approximation précédente x_{n+1} et un terme correctif δ_n. Si x_{n+1} (resp. x_n) est connu avec p_{n+1} (resp. p_n) bits significatifs alors d'après (5.13) et (5.14) il n'est pas raisonnable d'espérer pour x_{n+2} plus de $p_{n+1} + p_n$ bits significatifs ; on s'attend donc à ce que p_n bits du terme correctif soient utiles dans le calcul.

Le choix de la précision de travail pour le calcul de δ_n dépend de la précision supposée sur les opérandes et de la précision recherchée dans le résultat ; il faut aussi tenir compte des annulations.

Richard Brent discute dans [6] des précisions nécessaires dans le calcul de δ_n pour assurer une convergence d'un ordre o donné, et ce pour diverses méthodes de sécante (pour la méthode de la sécante décrite ici, $o = \frac{1+\sqrt{5}}{2}$). Nous reprenons le raisonnement ici ; adapté au cas présent.

Par hypothèse

$$x_{n+1} - u = \mathcal{O}(2^{-p_{n+1}})u$$
$$x_{n+2} - u = \mathcal{O}(2^{-p_{n+2}})u$$

et

$$f(x_{n+1}) \approx f'(u)(x_{n+1} - u) = \mathcal{O}(2^{-p_{n+1}})u.$$

Dans un voisinage suffisamment petit de la racine cherchée, la fraction $\frac{x_{n+1}-x_n}{f(x_{n+1})-f(x_n)}$ est essentiellement une approximation de $\frac{1}{f'(u)}$, donc $\mathcal{O}(1)$.

On pose

$$r = \frac{x_{n+1} - x_n}{f(x_{n+1}) - f(x_n)}$$

et $\widehat{r}$ la quantité effectivement calculée avec $\widehat{r} = (1 + \mu)r$, de même on note $\widehat{f(x_{n+1})}$ la quantité calculée pour $f(x_{n+1})$, avec

$$\widehat{f(x_{n+1})} = f(x_{n+1}) + \theta.$$

L'erreur relative sur r est donc μ, et θ est l'erreur absolue sur $f(x_{n+1})$. Développons :

$$\begin{aligned} \widehat{f(x_{n+1})}\widehat{r} &= (f(x_{n+1}) + \theta)(1 + \mu)\, r \\ &= r f(x_{n+1}) + r\mu f(x_{n+1}) + r\theta + r\theta\mu \\ |f(x_{n+1})r - \widehat{f(x_{n+1})}\widehat{r}| &= |r\theta + r\mu f(x_{n+1}) + r\theta\mu| \\ &\leq |r\theta| + |r\mu f(x_{n+1})| + |r\theta\mu|. \end{aligned}$$

L'ordre de grandeur recherché pour l'erreur globale sur $f(x_{n+1})r$ est $\mathcal{O}\left(2^{-(p_n+p_{n+1})}\right)u$ ce qui impose déjà $\theta = \mathcal{O}\left(2^{-(p_{n+1}+p_n)}\right)u$ en tenant compte des ordres de grandeur évoqués plus haut, et en regardant le terme $r\theta$. Le terme $|r\mu f(x_{n+1})|$ donne $\mu = \mathcal{O}\left(2^{-p_n}\right)$ et le dernier terme $r\mu\theta$ est alors effectivement négligeable.

La signification de ces termes s'apprécie visuellement sur la Figure 5.8 :

- $p_n + p_{n+1}$ est le nombre de bits corrects espérés pour x_{n+2} ;
- p_n est le nombre de bits jugés significatifs dans le terme correctif δ_n.

Avec $x_{n+1} - x_n = \mathcal{O}(2^{-p_n})u$ et $f(x_{n+1}) - f(x_n) = \mathcal{O}(2^{-p_n})u$, il faut calculer $f(x_n)$ avec une erreur absolue $\mathcal{O}(2^{-(2p_n)})u$ soit avec environ p_n bits de précision. Pour cette étape le même raisonnement suggère pour le calcul de $f(x_{n+1})$ une précision moindre de $2p_n - p_{n+1}$ bits mais pour pouvoir réutiliser à l'étape suivante (sauf à la fin) les valeurs calculées et ne faire qu'une seule évaluation de polynôme à chaque étape il est naturel de calculer $f(x_{n+1})$ avec une précision finale de p_{n+1} bits.

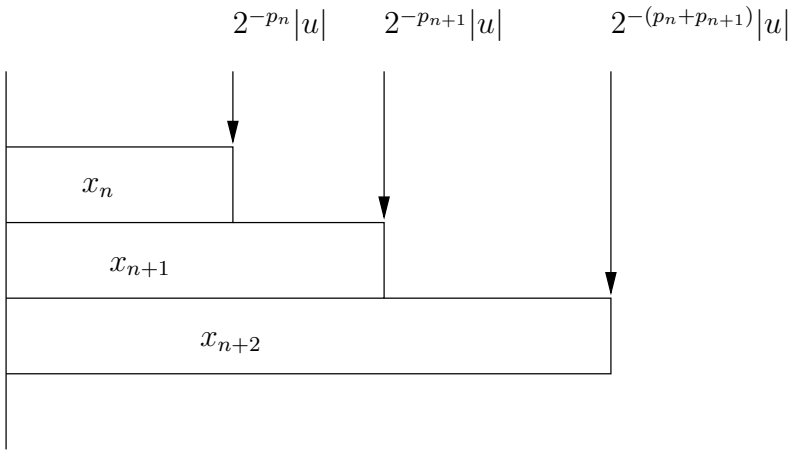

FIG. 5.8 – Une étape de l'itération de la sécante dans le domaine de convergence, indiquant les erreurs estimées sur chaque terme.

L'algorithme utilisé pour l'itération de la sécante emprunte des idées déjà mises en œuvre dans l'itération de Newton. Une estimation p_i du nombre de bits corrects dans x_i est calculée (seules les deux dernières valeurs sont utiles) et avec une estimation de la constante C sert à déterminer les précisions de calcul nécessaires pour obtenir le bon ordre de convergence. Le calcul de δ_n donne ensuite une information sur p_{n+1} qui permet d'ajuster l'estimation de C : on espère ainsi faire toujours des calculs avec la précision optimale, à savoir une précision assez grande pour que les erreurs d'arrondis n'entachent pas la qualité des résultats et donc la vitesse de convergence de façon appréciable, et en même temps une précision suffisamment petite pour que le temps ne soit pas perdu dans le calcul de bits non significatifs mathématiquement.

L'itération de la sécante telle qu'elle est implémentée dans CRQ est décrite dans l'Algorithme 21. Pour simplifier la lecture du code les améliorations décrites plus haut conservant un intervalle d'isolation et avec d'éventuelles étapes de dichotomie sont omises.

18.4 Comparaison de la méthode de la sécante et de l'itération de Newton

Comme expliqué plus haut, l'ordre de convergence de la méthode de la sécante est avec $\phi = \frac{1+\sqrt{5}}{2} \approx 1,618$ plus faible que celui de l'itération de Newton, qui est de 2. Richard Brent

Algorithme 21 Itération de la sécante

ENTRÉE : a, b, f, p_{fin} tels qu'il existe une racine $u \in [a, b]$ de f unique.
SORTIE : x tel que $|x - u| \leq \text{ulp}_{p_{\text{fin}}}(x)$.

$x_0 \leftarrow a$
$x_1 \leftarrow b$
$p_0 \leftarrow \log_2 |x_0| - \log_2(b - a)$
$p_1 \leftarrow p_0$
$n \leftarrow 1$
$f_0 \leftarrow \circ_{p_0}(f(x_0))$
while $p_n < p_{\text{fin}}$ **do**
    $f_n \leftarrow \circ_{p_n}(f(x_n))$
    $z \leftarrow \circ_{p_n}(x_n - x_{n-1})$
    $d \leftarrow \circ_{p_n}(f_n - f_{n-1})$
    $r \leftarrow \circ_{p_n}(z/d)$
    $r \leftarrow \circ_{p_n}(r \cdot f_n)$
    $x_{n+1} \leftarrow \circ_{p_n + p_{n+1}}(x_n - r)$
    $n \leftarrow n + 1$
end while
return x

donne dans [7] des arguments indiquant que la méthode de la sécante est asymptotiquement meilleure que l'itération de Newton sous certaines hypothèses. Une de ces hypothèses est que le coût d'une multiplication de taille p bits est asymptotiquement négligeable devant le coût d'évaluation de la fonction f dont on cherche une racine. Dans le cas où f est un polynôme de degré n le coût d'évaluation de f à précision p est essentiellement $nM(p)$, cette hypothèse est donc vérifiée à condition de faire aussi tendre n vers $+\infty$. Nous ne faisons pas cette hypothèse ici, le but est de comparer les deux méthodes de raffinement lorsque la précision devient grande mais que le polynôme dont on cherche une racine est fixé. On rappelle que $M(p)$ désigne le coût de la multiplication de deux nombres entiers (ou flottants) de taille p bits. Dans le but de comparer les complexités théoriques des deux méthodes de raffinement on se place sous les hypothèses suivantes :

1. la précision initialement connue sur la racine est p_{ini} pour chaque méthode,
2. $M(p) = p^\alpha$ pour une certaine valeur de α vérifiant $1 \leq \alpha \leq 2$,
3. les précisions de calcul dans l'itération de Newton forment une suite $(p_0 = p_{\text{ini}}, p_2, \ldots, p_c = p_{\text{fin}})$ optimale dans le sens où
$$p_{i+1} = 2p_i,$$
4. les précisions de calcul dans la méthode de la sécante forment une suite $(r_0 = p_{\text{ini}}, r_2, \ldots, r_d = p_{\text{fin}})$ optimale dans le sens où
$$r_{i+1} = \phi r_i,$$
5. $n \geq 2$ sans quoi l'utilisation d'une méthode de raffinement n'a pas d'intérêt par rapport à la résolution directe de l'équation linéaire $P(x) = 0$.

Les hypothèses 3 et 4 sont incompatibles (on n'a jamais une puissance entière de ϕ égale à une puissance entière de 2) mais cela ne gêne pas l'étude asymptotique. On néglige dans la suite le coût des additions. Pour l'étape i de l'itération de Newton on évalue P de degré n et P' de degré

$n-1$ en précision p_i. On considère que le coût de la division de taille p_i est $\gamma M(p_i)$ pour une certaine constante γ à préciser. Le coût de l'étape i est

$$(2n-1+\gamma)M(p_i) = (2n-1+\gamma)p_i^\alpha.$$

Pour la méthode de la sécante l'étape i coûte une évaluation de P de degré n et une division, le tout en précision r_i pour une complexité

$$(n+\gamma)M(r_i) = (n+\gamma){r_i}^\alpha.$$

Il s'agit donc de comparer

$$T_{\text{newton}} = (2n-1+\gamma)\sum_{i=0}^{c-1} p_i^\alpha$$

avec

$$T_{\text{sécante}} = (n+\gamma)\sum_{i=0}^{d-1} r_i^\alpha.$$

On pose $\tau = \frac{p_{\text{fin}}}{p_{\text{ini}}}$. Les hypothèses faites nous permettent d'écrire

$$\begin{aligned}
c &= \frac{\log\tau}{\log 2},\\
d &= \frac{\log\tau}{\log\phi},\\
T_{\text{newton}} &= (2n-1+\gamma)p_{\text{ini}}^\alpha\sum_{i=0}^{c-1}2^{i\alpha}\\
&= (2n-1+\gamma)p_{\text{ini}}^\alpha\frac{2^{c\alpha}-1}{2^\alpha-1}\\
&= (2n-1+\gamma)p_{\text{ini}}^\alpha\frac{\tau^\alpha-1}{2^\alpha-1},\\
T_{\text{sécante}} &= (n+\gamma)p_{\text{ini}}^\alpha\sum_{i=0}^{d-1}\phi^{i\alpha}\\
&= (n+\gamma)p_{\text{ini}}^\alpha\frac{\phi^{d\alpha}-1}{\phi^\alpha-1}\\
&= (n+\gamma)p_{\text{ini}}^\alpha\frac{\tau^\alpha-1}{\phi^\alpha-1}.
\end{aligned}$$

Il suffit donc de trouver le signe de $D = \frac{(2^\alpha-1)(\phi^\alpha-1)}{(\tau^\alpha-1){p_{\text{ini}}}^\alpha}\left(T_{\text{newton}} - T_{\text{sécante}}\right)$:

$$\begin{aligned}
D &= (2n-1-\gamma)(\phi^\alpha-1) - (n+\gamma)(2^\alpha-1)\\
&= \left[2\phi^\alpha - 2^\alpha - 1\right]n + (\phi^\alpha-2^\alpha)\gamma - \phi^\alpha + 1.
\end{aligned}$$

On peut vérifier que pour $1 \le \alpha \le 2$ on a $2\phi^\alpha - 2^\alpha - 1 \ge 0$ et donc la méthode de la sécante est asymptotiquement meilleure (quand $\tau \to +\infty$) dès que

$$n \ge \frac{(\phi^\alpha-2^\alpha)\gamma - \phi^\alpha + 1}{2\phi^\alpha - 2^\alpha - 1}.$$

De plus $1 - \phi^\alpha \leq 0$, $\gamma > 0$ et $\phi^\alpha - 2^\alpha < 0$ donc la méthode de la sécante est toujours asymptotiquement meilleure que l'itération de Newton lorsque $\tau \to +\infty$, indépendamment du degré n du polynôme.

Cela n'enlève pas l'intérêt d'une implémentation efficace de l'itération de Newton pour des petites précisions.

En pratique dans CRQ l'algorithme de raffinement utilisé par défaut est l'itération de Newton scalaire, suite à des expériences montrant qu'elle est plus efficace que les autres méthodes. Cependant la vitesse de convergence de l'implémentation de la méthode de la sécante est inférieure à la progression géométrique de raison ϕ que prédit la théorie et ceci est dû à un bug qui n'a pas encore été trouvé. Une version ultérieure de CRQ devrait corriger ce bug et, le cas échéant, utiliser alors la méthode de la sécante comme algorithme de raffinement par défaut.

Précision dégradée

Nous présentons une idée supplémentaire (et classique) pour améliorer le temps de calcul du raffinement pour la méthode de la sécante et l'itération de Newton. Cette amélioration n'est pas encore présente dans CRQ.

Pour atteindre une précision cible donnée sur la racine recherchée il est parfois profitable de perturber la progression géométrique des précisions atteintes à chaque étape. En guise d'exemple concret, supposons que l'on utilise l'itération de Newton avec une précision connue initialement de 1 bit pour une précision cible de 257 bits. Avec un doublement à chaque étape de la précision connue la séquence des précisions sera

$$p_{\text{ini}} = 1; 2; 4; 8; 16; 32; 64; 128; 256; 512 = p_{\text{fin}}.$$

La précision de 256 atteinte à l'avant-dernière étape étant insuffisante, une étape supplémentaire avec une précision presque deux fois trop grande est nécessaire. En dégradant les précisions, on pourrait avoir la séquence suivante :

$$1; 2; 3; 5; 9; 17; 33; 65; 129; 257.$$

On obtient le même nombre d'étapes effectuées avec des précisions de travail inférieure, donc un temps de calcul plus petit.

Si l'on suppose que le coût total de la chaîne de précisions optimale est croissant en fonction de la précision cible, alors la stratégie optimale se trouve en partant de la précision cible et en appliquant la transformation $x \mapsto \lceil \frac{x}{2} \rceil$ jusqu'à obtenir une précision inférieure à la précision de départ.

La même idée s'applique à la méthode de la sécante en ajustant l'ordre de convergence.

6

Résultats expérimentaux

Les algorithmes décrits dans les chapitres précédents ont été implémentés sous la forme d'une bibliothèque d'intégration numérique appelée CRQ. On donne dans ce chapitre une description plus précise et technique de la bibliothèque, aussi bien au niveau des choix de développement que du point de vue de l'utilisateur. On donne des résultats expérimentaux de la bibliothèque sur quelques intégrales choisies, en la comparant avec d'autres logiciels fournissant une opération d'intégration numérique.

19 Description de CRQ

L'acronyme CRQ signifie *Correctly Rounded Quadrature*. Le double objectif de la bibliothèque est en effet de maîtriser l'erreur commise dans les calculs, et d'effectuer rapidement des intégrations numériques. Techniquement, la bibliothèque est écrite en langage `C` et se compose d'un peu plus de 5000 lignes de code source réparties sur vingt fichiers (à la version `0.0.20060823`). Pour atteindre ces objectifs, la bibliothèque utilise GMP pour les calculs en nombres entiers et MPFR pour les calculs en nombres flottants en précision arbitraire.

En étendant les principes de la norme IEEE754 aux nombres flottants en précision arbitraire, MPFR remplit la condition de fiabilité nécessaire au développement de CRQ. La garantie d'arrondi correct sur chaque opération de base est ce qui permet de construire les lemmes présents dans les chapitres 3 et 4 pour obtenir finalement les théorèmes 4 et 8. De plus la bibliothèque MPFR fournit une riche panoplie d'implémentations avec arrondi correct de fonctions élémentaires, permettant aux utilisateurs de programmer une vaste classe d'intégrandes et d'utiliser CRQ pour calculer leurs intégrales. Dans le cas où cela ne serait pas possible, par exemple si la fonction à intégrer a déjà été programmée en utilisant une autre bibliothèque de calcul flottant, la conversion vers le format de flottants MPFR est à la charge de l'utilisateur (il ne s'agit *a priori* pas d'un handicap majeur). Récemment, MPFR a été intégrée dans la collection de compilateurs `GCC`. Plus spécifiquement, le compilateur Fortran utilise MPFR pour calculer correctement les expressions arithmétiques au moment de la compilation. Il ne s'agit là que d'un exemple parmi d'autres[5] d'utilisations de MPFR dans des logiciels libres ou non, et illustre la diffusion de MPFR en particulier et plus généralement de l'arithmétique en précision arbitraire et du souci de la gestion d'erreur dans le monde du calcul numérique.

GMP est sans doute la bibliothèque de calcul en nombres entiers en précision arbitraire la plus utilisée et, à en croire sa page d'accueil, aussi la plus rapide. Un certain nombre de logiciels de calcul mathématique utilisent GMP à la place de leur propre implémentation de nombres entiers de taille arbitraire (c'est le cas de Pari/GP et de Maple à partir de la version 9). Initialement distribué avec GMP, MPFR est depuis la version 2.0.2 (novembre 2003) distribué séparément mais se base depuis toujours sur GMP en interne pour les calculs en précision arbitraire.

Dans GMP, le type correspondant à un entier signé est `mpz_t`, et les flottants MPFR ont pour type `mpfr_t`.

19.1 Interface et structure interne

Les méthodes d'intégration de Newton-Cotes (vues dans le chapitre 3) et de Gauss-Legendre (vues dans le chapitre 4) sont implémentées dans CRQ. On peut distinguer dans CRQ deux niveaux d'interface pour les fonctions d'intégration : un ensemble de fonctions « internes » qui offrent le plus de contrôle à l'utilisateur, et des fonctions de plus haut niveau avec une certaine part d'automatisation.

[5] Que l'on peut consulter sur la page web du projet `http://mpfr.org/`

Niveau interne

Au niveau le plus bas, seule l'intégration à proprement parler est calculée. Les coefficients nécessaires à la méthode doivent avoir été calculés au préalable, et la composition n'est pas disponible. Il s'agit d'une traduction quasi littérale en `C` de l'algorithme 7 pour la méthode de Newton-Cotes, et de l'algorithme 9 pour la méthode de Gauss-Legendre. Les prototypes de ces fonctions sont

```
void gen_nc_closed (mpfr_t res, mpfr_t errb, mpfr_t a, mpfr_t b,
                    unsigned long n, mpz_t *coeffs, mpz_t d, aqfunc_t f,
                    mpfr_t m, mp_prec_t wp, mpfr_t *fxs[])
```

pour Newton-Cotes et

```
void gen_gl_closed (mpfr_t res, mpfr_t errb, mpfr_t a, mpfr_t b,
                    unsigned long n, mpfr_t *coeffs, mpfr_t *roots,
                    aqfunc_t f, mpfr_t m, mp_prec_t wp, mpfr_t *fxs[]);
```

pour Gauss-Legendre. Les prototypes sont identiques à un argument près (`roots`) pour la méthode de Gauss-Legendre qui donne les points d'évaluation (ils sont triviaux pour Newton-Cotes). Ces fonctions calculent à la fois le résultat de l'intégration (`mpfr_t res`) ainsi qu'une borne sur les erreurs d'arrondi (`mpfr_t errb`). Le type `aqfunct_t` est le type désigné des fonctions que CRQ intègre ; il s'agit d'un alias pour les fonctions ayant deux arguments de type `mpfr_t` : un pour le point où la fonction doit être évaluée, et un pour le résultat. Comme indiqué dans les chapitres précédents, l'intégrande est vu par CRQ comme un oracle retournant la valeur de la fonction en un point donné, avec la précision requise et une erreur d'au plus un ulp. La précision demandée pour le résultat (`res`) n'a pas besoin d'être précisée explicitement, c'est celle de l'objet `res`. Un dernier argument qui mérite d'être expliqué est le tableau `mpfr_t *fxs[]` qui doit être alloué par l'appelant et qui sert à stocker les $f(x_i)$ puis les $w_i f(x_i)$ avant de les sommer. L'idée est de ne pas perdre du temps à initialiser plusieurs fois un tel tableau lorsque la fonction d'intégration est appelée à plusieurs reprises avec les mêmes paramètres, comme c'est le cas lors d'une composition par exemple. Ce tableau doit être de taille `n` et ses entrées doivent avoir une mantisse de taille `wp` bits.

Les fonctions de calcul des coefficients des méthodes d'intégration se situent techniquement à ce même niveau le plus bas dans l'interface, puisqu'elles les complètent naturellement. Pour Newton-Cotes, la fonction de calcul des coefficients a pour prototype :

```
void compute_nc_coeffs (mpz_t *coeffs, mpz_t d, unsigned long n, int algo);
```

Pour un nombre donné de points d'évaluation n cette fonction retourne les $\lfloor \frac{n}{2} \rfloor$ premiers poids de la méthode sous la forme d'un tableau des numérateurs `coeffs` et d'un dénominateur commun `d`. Le paramètre supplémentaire `algo` détermine l'algorithme de calcul des poids effectivement utilisé : pour `algo=0` on utilise l'algorithme naïf (p. 25) et pour `algo=1` on utilise l'algorithme rapide (p. 27). Pour Gauss-Legendre, le prototype est plus complexe :

```
void compute_gl_coeffs (mpfr_t *coeffs, mpfr_t *roots, unsigned long n,
                        struct crq_gl_opts options, struct crq_stats *timings);
```

Contrairement à la méthode de Newton-Cotes, les points d'évaluation ne sont plus triviaux et sont calculés en même temps que les poids. De même les poids et points d'évaluation ne sont pas exactement représentables en nombres rationnels, on les calcule donc en nombres flottants à la précision demandée en sortie (cette précision est celle des éléments du tableau `roots`). Comme

cela a été vu dans la section 18, le calcul des racines du polynôme de Legendre, nécessaire au calcul des points d'évaluation de la méthode de Gauss-Legendre, peut se faire avec plusieurs choix d'algorithmes pour le raffinement. De plus il est possible d'accélérer ces calculs en réutilisant des quantités calculées lors d'une exécution précédente, à savoir le polynôme de Legendre et pour chaque racine un intervalle d'isolation plus ou moins fin. On peut ainsi commencer les méthodes de raffinement (dichotomie, itération de Newton, itération de la sécante) avec une très bonne approximation de la racine cherchée. Ces quantités sont alors lues depuis un fichier.

Toutes ces options sont contrôlées via le paramètre `options`. Pour donner à l'utilisateur les moyens de faire des tests pour déterminer les meilleurs paramètres, la fonction retourne dans la variable `stats` le temps processeur passé dans chaque étape du calcul : isolation, raffinement, calcul des poids post-raffinement. Un tel paramètre recueillant les statistiques n'existe pas pour le calcul des coefficients de Newton-Cotes car il n'y a pas la possibilité d'influer sur les étapes de l'algorithme ; le temps de calcul est monolithique. L'ajout relativement récent de l'algorithme rapide et sa variation de calcul modulo des petits premiers (non encore implémentée) peuvent remettre ce choix en question.

Haut niveau

Les méthodes d'intégration de Gauss-Legendre et de Newton-Cotes possèdent chacune une interface de plus haut niveau, il s'agit de

```
void compose_nc (mpfr_t res, mpfr_t errb, mpfr_t a, mpfr_t b, unsigned long m,
                 unsigned long n, aqfunc_t f, mpfr_t m1, mp_prec_t wp)
```

pour Newton-Cotes et de

```
void compose_gl_stats (mpfr_t res, mpfr_t errb, mpfr_t a, mpfr_t b,
                       unsigned long m, unsigned long n, aqfunc_t f,
                       mpfr_t m1, mp_prec_t wp, struct crq_gl_opts options,
                       struct crq_stats *timings)
```

pour Gauss-Legendre. Pour ces deux fonctions le résultat de l'intégration est renvoyé dans le paramètre `res`, la borne calculée sur les erreurs d'arrondi est donnée dans `errb` (le calcul de la borne sur l'erreur mathématique est effectué par les fonctions `gl_error` pour Gauss-Legendre et `nc_error` pour Newton-Cotes pour donner la possibilité à l'utilisateur de distinguer les deux au lieu de n'avoir qu'une seule borne consolidée). Les autres arguments de ces fonctions sont :

- `a`, `b` : bornes inférieure et supérieure de l'intervalle d'intégration,
- `m` : degré de composition de la méthode,
- `n` : nombre de points de la méthode,
- `f` : fonction à intégrer,
- `m1` : borne sur $|f'|$,
- `wp` : précision de travail.

En plus de cela la fonction d'intégration pour la méthode de Gauss-Legendre demande des options pour les algorithmes d'isolation et de raffinement à utiliser, et une variable recueillant des *timings* de chacune de ces étapes. Il existe une fonction `compose_gl` encapsulant `compose_gl_stats` pour les utilisateurs qui ne sont pas intéressés par les *timings*.

Ces fonctions sont considérées de plus haut niveau dans la mesure où :

- elles calculent automatiquement les paramètres (poids, points d'évaluation) associés à la méthode,
- elles supportent directement la composition de la méthode.

Si la borne sur la dérivée première de f n'est pas renseignée de façon utile (en l'initialisant à `NaN` par exemple) alors la borne d'erreur renvoyée n'aura pas de sens, mais cela n'empêche évidemment pas d'approcher l'intégrale.

Pour les utilisateurs qui ne peuvent pas donner de borne sur les dérivées de la fonction à intégrer, un mode d'intégration adaptatif est aussi programmé dans CRQ. Il s'agit de l'algorithme 1 vu dans le chapitre 1. Dans CRQ son prototype est le suivant :

```
void adaptive_quad (mpfr_t res, mpfr_t a, mpfr_t b, aqfunc_t f,
                    mp_prec_t prec, void (*quadf)(mpfr_t, mpfr_t,
                    mpfr_t, aqfunc_t, void *), void *params)
```

Les paramètres `a`, `b` et `f` ont le même sens maintenant usuel que pour les autres fonctions d'intégration. L'argument `prec` ne désigne plus une précision de travail mais un but à atteindre par l'algorithme en terme de précision relative, exprimée en nombre de bits. L'idée de l'algorithme est de détecter les portions de l'intervalle d'intégration où la fonction n'est pas assez régulière pour la méthode d'intégration utilisée, en bissectant récursivement jusqu'à ce qu'une bissection n'apporte plus de différence significative dans le résultat global. Plus précisément on détermine qu'il n'est plus utile de découper un intervalle lorsque la différence entre l'intégration avec et sans bissection est inférieure à une borne dépendant du but de précision `prec` et d'une estimation grossière de la valeur de l'intégrale globale.

L'erreur de méthode est donc estimée de façon empirique, et il n'y a plus de borne d'erreur garantie par la fonction d'intégration. Du fait des erreurs d'arrondi provenant à la fois des appels à la fonction d'intégration et des additions des résultats intermédiaires, la précision effectivement atteinte peut être loin de la précision cible qui n'est à considérer par l'utilisateur que comme une indication donnée à la fonction d'intégration. L'intégration adaptative est une stratégie générique qui ne dépend pas de la méthode sous-jacente ; cette méthode est donc passée en argument (`quadf`). L'argument supplémentaire `void *params` constitue les paramètres à fournir à la méthode d'intégration (par exemple : le nombre de points d'évaluation et les poids correspondants pour la méthode de Newton-Cotes). Le langage `C` n'étant malheureusement pas fonctionnel, il n'est pas possible d'encapsuler à l'exécution ces paramètres directement dans une fonction créée à la volée, d'où la nécessité d'ajouter cet argument pour permettre un peu de flexibilité. Techniquement le profil d'une fonction de type `quadf` est :

```
void quadf (mpfr_t res, mpfr_t a, mpfr_t b, aqfunc_t f, void *params)
```

où outre les arguments usuels, les paramètres propres à la fonction d'intégration sont donnés dans `params` sous la forme d'un pointeur opaque et dont l'interprétation est laissée libre à la fonction d'intégration (pour simuler un comportement « fonctionnel » comme expliqué plus haut). Un exemple de fonction glu (*wrapper*) permettant d'utiliser la fonction d'intégration de Gauss-Legendre dans la stratégie adaptative est donné dans la distribution CRQ.

19.2 Exemple d'utilisation de CRQ

Dans la distribution de CRQ, le fichier `check.c` a évolué d'un simple jeu de tests pour valider les méthodes d'intégration au cours de leur écriture, à un programme d'expérimentations plus générales et de mesures de performances. Il constitue un bon point de départ pour un nouvel utilisateur de CRQ, mais en montrant trop de choses en même temps ce fichier risque de laisser le nouveau venu confus. Nous proposons donc ici un exemple minimal complet et commenté d'utilisation de CRQ.

On choisit pour cela de montrer comment calculer

$$I = \int_0^1 \sin(\sin(x))\mathrm{d}x$$

avec la méthode de Gauss-Legendre avec $n = 6$ points. On pose $f = x \mapsto \sin(\sin(x))$.

On s'intéresse dans un premier temps à programmer la fonction f elle-même en répondant aux conditions requises par la bibliothèque pour que le calcul de la borne d'erreur soit correct. Pour rappel, la condition est de calculer f avec une erreur inférieure à 1 ulp sur le résultat rendu. On appelle p la précision demandée en sortie. La fonction f est explicitée dans l'algorithme 22 pour fixer les notations. Nous allons prouver que la condition sur la borne d'erreur est satisfaite. On distingue la fonction sinus au sens mathématique notée sin de son implémentation dans MPFR notée fsin.

Algorithme 22 La fonction $x \mapsto \sin(\sin(x))$.

ENTRÉE : $x \in [0, 1]$, une précision demandée p.
SORTIE : un flottant z en précision p tel que $|\sin(\sin(x)) - z| \leq \mathrm{ulp}_p(z)$.
1: $y \leftarrow \circ_{p+2}(\mathrm{fsin}(x))$
2: $z \leftarrow \circ_p(\mathrm{fsin}(y))$
3: **return** z

On rappelle que les notations $E(y)$, $\mathrm{ulp}_p(y)$ et $\circ_p(y)$ sont définies au chapitre 2 (p. 14).
La notation indicée pour $\mathrm{ulp}_p(y)$ et $\circ_p(y)$ désigne la précision de travail p considérée.
La fonction fsin utilisée est supposée calculer avec arrondi correct au plus proche, on a donc

$$\begin{aligned} |y - \sin(x)| &\leq \frac{1}{2}\mathrm{ulp}_{p+2}(y), \\ |z - \sin(y)| &\leq \frac{1}{2}\mathrm{ulp}_p(z) \end{aligned}$$

ce qui donne

$$\begin{aligned} |z - \sin(\sin(x))| &\leq |z - \sin(y)| + |\sin(y) - \sin(\sin(x))| \\ &\leq \frac{1}{2}\mathrm{ulp}_p(z) + \max_{u\in[0,1]}|\cos(u)| \cdot |y - \sin(x)| \\ &\leq \frac{1}{2}\left(\mathrm{ulp}_p(z) + \mathrm{ulp}_{p+2}(y)\right). \end{aligned}$$

Il reste à comparer $\mathrm{ulp}_p(z)$ et $\mathrm{ulp}_{p+2}(y)$. Remarquons que sin est croissante concave sur $[0, \frac{\pi}{2}]$. On a

$$\begin{array}{ccccc} 0 & \leq & x & \leq & 1 \leq \frac{\pi}{2} \\ 0 & \leq & \sin(x) & \leq & \sin(1). \end{array}$$

Après arrondi

$$0 \leq y \leq \sin(1) + \frac{1}{2}\mathrm{ulp}_{p+2}(\circ_{p+2}(\sin(1))).$$

Si $x = 1$ alors

$$|y - \sin(1)| \leq \frac{1}{2}\mathrm{ulp}_{p+2}(y).$$

Sachant que $\sin(1) = 0,1101\ldots$ en format binaire, $E(y) = E(\sin(1)) = E(\frac{1}{2})$ dès que $p \geq 2$. On suppose donc $p \geq 2$ et on peut écrire

$$0 \leq y \leq \sin(1) + 2^{-p-3}.$$

On vérifie qu'alors $y \leq 1$. Par concavité, la fonction sin est supérieure à sa corde sur $[0, 1]$:

$$\sin(y) \geq y \sin(1).$$

Après arrondi

$$z = (1 + \theta) \sin(y)$$

avec $|\theta| \leq 2^{-p}$. On continue à minorer z par

$$z \geq (1 - 2^{-p}) \sin(1) y$$

puis le lemme 1 donne

$$\mathrm{ulp}_p(z) \geq \frac{1 - 2^{-p}}{2} \sin(1) \mathrm{ulp}_p(y).$$

Nous rappelons que $\mathrm{ulp}_p(y) = 4\ \mathrm{ulp}_{p+2}(y)$, et nous utilisons le fait que $\sin(1) \geq \frac{13}{16}$ et $p \geq 2$ pour majorer plus simplement

$$\mathrm{ulp}_p(z) \geq \frac{39}{32} \mathrm{ulp}_{p+2}(y)$$

et alors

$$\begin{aligned} |z - \sin(\sin(x))| &\leq \frac{1}{2}\left(\mathrm{ulp}_p(z) + \frac{32}{39}\mathrm{ulp}_p(z)\right) \\ &\leq \frac{71}{78}\mathrm{ulp}_p(z) \\ &\leq \mathrm{ulp}_p(z) \end{aligned}$$

ce qui achève de prouver qu'il suffit de calculer la fonction fsin intermédiaire avec 2 bits de précision supplémentaire pour satisfaire la condition d'erreur inférieure à 1 ulp sur le résultat de f. On en déduit l'écriture en `C` donnée lignes 6 à 13 dans le listing 6.1.

Le calcul de la borne sur f' demande moins d'efforts :

$$\begin{aligned} f'(x) &= \cos(\sin(x)) \cos(x) \\ |f'(x)| &\leq 1. \end{aligned}$$

On a choisi arbitrairement $n = 6$, et la méthode de Gauss-Legendre demande une borne sur la dérivée $(2n)$-ième de f. On utilise pour cela Maple, qui ne donne pas de borne satisfaisante sur $[0, 1]$ pour $f^{(12)}$. En revanche, on obtient par `extrema(diff(sin(sin(x)), x$13), {}, x)` que

$$|f^{(13)}(x)| \leq 990784$$

et donc avec $f^{(12)}(0) = 0$ cette borne est aussi valide sur $[0, 1]$ pour $f^{(12)}$. Le listing 6.1 donne le programme finalement obtenu pour calculer I. Après l'inclusion des fichiers d'en-tête standard `C` ainsi que celui de la bibliothèque CRQ elle-même, on définit la fonction `sinsin` avec une précision de 2 bits supplémentaires par rapport à la précision demandée en sortie. Les paramètres du

nombre de points de la méthode, du degré de composition et la précision de travail sont fixés en ligne 23. On choisit ici de ne pas composer la méthode ($m = 1$). Pour la méthode de Gauss-Legendre, on doit choisir les algorithmes utilisés lors du calcul des poids : on choisit ici de ne partir que d'une isolation des racines de P (et non pas simultanément de P et de P'), de raffiner avec l'itération de Newton et lorsque celle-ci échoue, d'utiliser le raffinement par une dichotomie avec coefficients tronqués (ce qui dégrade les performances par rapport à Newton). Ces algorithmes sont détaillés dans la section 18. En initialisant le paramètre `options.file` à `NULL` on s'interdit de réutiliser des données pré-calculées.

Les lignes 28 à 42 du listing 6.1 se composent de l'initialisation des bornes de l'intervalle d'intégration, des bornes sur les dérivées première et $(2n)$-ième de f et en l'appel à la fonction d'intégration. On termine par le calcul d'une borne sur l'erreur de méthode avec `gl_error` et par l'affichage des quantités calculées. L'exécution affiche ceci :

```
val = 4.30606103107249495263571458360626049260026548902068242014591 13e-1
1.8636086852782783e-10 [method error]
1.1547928308686448e-59 [roundoff error]
1.8636086852782785e-10 [total error]
4ms [Total CPU time]
```

La comparaison des bornes d'erreur d'arrondi et de méthode suggère que l'ordre de méthode utilisé est insuffisant par rapport à la précision utilisée, la borne sur les erreurs d'arrondi étant négligeable devant celle sur l'erreur de méthode. Le temps de calcul est mesuré en millisecondes sur un processeur G4 à 1,2GHz. Il est bon de préciser que la borne d'erreur totale affichée concerne la représentation en flottant MPFR de la valeur de l'intégrale calculée ; la valeur affichée par le programme souffre d'une erreur d'arrondi supplémentaire due à la conversion en base 10.

20 Résultats expérimentaux

20.1 Calcul de poids

Nous donnons dans la figure 6.1 une comparaison des performances des deux algorithmes de calcul des poids de Newton-Cotes pour un nombre de points variant de 2 à 2048. Pour les nombres de points utilisés en pratique l'algorithme asymptotiquement meilleur ne confirme hélas pas son avantage sur la méthode naïve.

20.2 Intégration de fonctions

Pour mesurer les performances de CRQ nous avons choisi de calculer un certain nombre d'intégrales, de paramètres et de précision de calcul. Le but est de mesurer les performances de CRQ en explorant un ensemble supposé représentatif de paramètres, et aussi de les comparer à d'autres bibliothèques.

Plus précisément, nous avons choisi les intégrales suivantes pour les expérimentations :

Listing 6.1 – Un exemple d'utilisation de CRQ.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "crq.h"
4  #include "utils.h" /* pour cputime */
5
6  void sinsin (mpfr_t res, mpfr_t x)
7  {
8      mpfr_t tmp;
9      mpfr_init2 (tmp, mpfr_get_prec (res) + 2);
10     mpfr_sin (tmp, x, GMP_RNDN);
11     mpfr_sin (res, tmp, GMP_RNDN);
12     mpfr_clear (tmp);
13 }
14
15 int main (int argc, char *argv[])
16 {
17     unsigned int n, m;
18     mp_prec_t wp;
19     mpfr_t a, b, res, total_error, math_error, rounderr;
20     mpfr_t diffbound, ndiffbound;
21     struct crq_gl_opts options;
22     int total_time;
23     m = 1; n = 6; wp = 200;
24     options.iso = SIMPLE;
25     options.ref = NEWTON;
26     options.dicho = TRUNC;
27     options.file = NULL;
28     mpfr_init2 (a, wp);
29     mpfr_init2 (b, wp);
30     mpfr_init2 (res, wp);
31     mpfr_init (math_error);
32     mpfr_init2 (diffbound, wp);
33     mpfr_init2 (ndiffbound, wp);
34     mpfr_init (roundoff_error);
35     mpfr_init (total_error);
36     mpfr_set_ui (a, 0, GMP_RNDN);
37     mpfr_set_ui (b, 1, GMP_RNDN);
38     mpfr_set_ui (diffbound, 1, GMP_RNDN);
39     mpfr_set_ui (ndiffbound, 990784, GMP_RNDN);
40     total_time = cputime ();
41     compose_gl (res, rounderr, a, b, m, n, sinsin, diffbound, wp, options);
42     total_time = cputime () - total_time;
43     gl_error (math_error, a, b, m, n, ndiffbound);
44     printf ("val = ");
45     mpfr_out_str (NULL, 10, 0, res, GMP_RNDN); printf ("\n");
46     mpfr_out_str (NULL, 10, 0, math_error, GMP_RNDN);
47     printf (" [method error]\n");
48     mpfr_out_str (NULL, 10, 0, rounderr, GMP_RNDN);
49     printf (" [roundoff error]\n");
50     mpfr_add (total_error, rounderr, math_error, GMP_RNDU);
51     mpfr_out_str (NULL, 10, 0, total_error, GMP_RNDN);
52     printf (" [total error]\n");
53     printf ("%d [Total CPU time]\n", total_time);
54     mpfr_clears (a, b, res, math_error, diffbound, ndiffbound, NULL);
55     return 0;
56 }
```

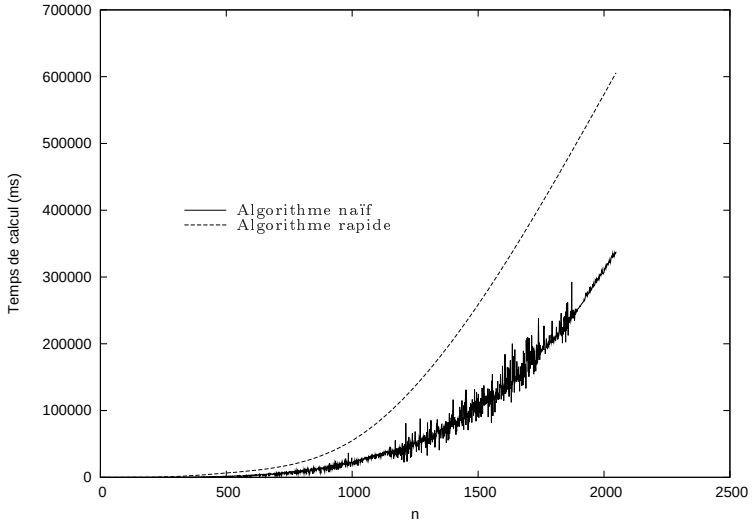

FIG. 6.1 – Comparaison des performances des deux algorithmes de calcul des poids de Newton-Cotes.

$$
\begin{aligned}
I_1 &= \int_0^3 \mathrm{d}x = 3 \\
I_2 &= \int_0^3 e^x \mathrm{d}x = e^3 - 1 \\
I_3 &= \int_0^1 x \log(1+x) \mathrm{d}x = \frac{1}{4} \\
I_4 &= \int_0^1 \sqrt{1-x^2} \mathrm{d}x = \frac{\pi}{4} \\
I_5 &= \int_0^1 \sin(\sin(x)) \mathrm{d}x \\
I_6 &= \int_0^\pi \sin(x) \exp(\cos(x)) \mathrm{d}x = e - \frac{1}{e} \\
I_7 &= \int_0^1 \frac{1}{1+x^2} \mathrm{d}x = \frac{\pi}{4} \\
I_8 &= \int_0^1 \frac{1}{1+10^{10}x^2} \mathrm{d}x = 10^{-5} \arctan(10^5) \\
I_9 &= \int_0^{1,1} \exp(-x^{100}) \mathrm{d}x \\
I_{10} &= \int_0^{10} x^2 \sin(x^3) \mathrm{d}x = \frac{1}{3}(1 - \cos(1000)) \\
I_{11} &= \int_0^1 \sqrt{x} \mathrm{d}x = \frac{2}{3} \\
I_{12} &= \int_0^1 \max(\sin(x), \cos(x)) \mathrm{d}x = \int_0^{\frac{\pi}{4}} \cos(x) \mathrm{d}x + \int_{\frac{\pi}{4}}^1 \sin(x) \mathrm{d}x \\
&= \sin\frac{\pi}{4} + \cos\frac{\pi}{4} - \cos 1 = \sqrt{2} - \cos 1.
\end{aligned}
$$

Le problème I_1 d'intégrer une constante est sans doute le cas le plus simple possible, il sert à vérifier un minimum la correction du système d'intégration. La fonction exponentielle a été choisie dans I_2 pour sa grande régularité et nous considérons qu'elle ne devrait pas poser de problème aux systèmes d'intégration.

Pour le reste des problèmes d'intégration nous nous sommes tournés vers des tests de performances déjà publiés : I_3 et I_4 sont issus de [2], le problème I_5 a servi dans la session d'entraînement de la compétition de calcul *Many Digits* [29] et les problèmes I_6 à I_{12} sont tirés de [26]. Pour l'intégrale I_{12} il s'agit bien de la première forme faisant apparaître la fonction max qui est soumise aux systèmes.

La comparaison de plusieurs systèmes d'intégration numérique est un exercice délicat du fait justement de la variété de paramètres à envisager. Les buts « fondateurs » de chacun des systèmes sont souvent suffisamment différents pour empêcher une comparaison raisonnable : que faut-il comparer entre un système comme Maple qui fournit une fonction générique d'intégration avec sélection automatique des paramètres, et CRQ où une liberté[6] de choix plus grande est laissée à l'utilisateur ? La comparaison devient d'autant plus ardue que la différence de conception des systèmes peut se situer non seulement au niveau de l'intégration elle-même mais déjà au niveau de l'arithmétique flottante sous-jacente ; et qu'un utilisateur n'est pas forcément au fait des subtilités et erreurs à ne pas faire pour chacun des systèmes (quand bien même il en aurait écrit un lui-même).

Ces précautions introductives étant posées, le protocole de conduite des expérimentations a été de comparer entre elles des stratégies d'intégration automatiques, vues plus ou moins comme des boîtes noires prenant en entrée la fonction à intégrer et une précision souhaitée en sortie. On mesure pour chacune la précision du résultat calculé par rapport à une valeur exacte de l'intégrale, et le temps processeur utilisé par le programme. Pour toutes les intégrales sauf I_5 et I_9 la valeur exacte a été calculée avec MPFR dans une précision supérieure à la plus grande des précisions demandées lors des tests en utilisant la formule explicite pour le résultat. Leur exactitude dépend donc de la correction des fonctions MPFR utilisées. Du fait de contraintes techniques (notamment l'absence de code source disponible pour certains systèmes) il est très difficile voire impossible d'isoler la méthode d'intégration d'un système pour la mesurer seule : on mesure donc bien chaque système dans son ensemble. Par exemple dans les performances mesurées pour Mathematica il y a implicitement la qualité de l'arithmétique sous-jacente qui influe fortement sur les temps de calcul.

Pour les intégrales I_5 et I_9 nous ne connaissons pas *a priori* de formule close (plus exactement : Maple ne connaît pas de formule close) et donc le calcul de la valeur de référence se fait par validation mutuelle des systèmes entre eux, et par test de cohérence d'un système. Le problème de la validation des résultats d'une bibliothèque de calcul numérique mérite une étude en soi ; tous les logiciels de calcul y sont confrontés. Dans notre cas on entend par validation mutuelle que le calcul de l'intégrale est demandé à plusieurs systèmes et que l'on considère comme valides les chiffres communs entre les réponses de ces systèmes. La vérification de la cohérence consiste à demander à un même système la valeur d'une intégrale à des précisions différentes et considérer comme valides les chiffres communs à ces deux valeurs.

Nous avons cependant validé la valeur de l'intégrale I_5 à l'aide de CRQ en utilisant non pas la stratégie adaptative dont la convergence est heuristique, mais avec la composition de la méthode de Gauss-Legendre pour des paramètres bien choisis et avec une erreur bornée directement par CRQ. Nous avons réutilisé pour cela le programme développé pour la compétition Many Digits, dont la stratégie est de recommencer en augmentant le nombre de points n de 10% à chaque fois

[6] Qui ne vient pas sans effort.

que la précision obtenue en sortie est insuffisante par rapport à la précision décimale requise. Dans ce programme pour un nombre de points n nous avons borné la dérivée $(2n+1)$-ième de $f(x) = \sin(\sin(x))$ par sa valeur en 0, conjecture vérifiée pour des petites valeurs de n par Maple, puis par intégration on en déduit la même borne pour la dérivée $(2n)$-ième (la fonction étant impaire ses dérivées impaires s'annulent en 0). On calcule la valeur en 0 par le développement en série de f en 0.

En fixant l'ordre de composition arbitrairement à $m = 512$ et une précision de travail de 10340 bits (soit un peu plus de 3110 chiffres décimaux), cette stratégie s'arrête avec succès sur $n = 399$. On calcule une borne de $|f^{(799)}|$ par le développement en série pour obtenir que

$$|f^{(799)}| < 2^{4759}$$

puis l'utilisation de cette borne dans CRQ donne le résultat suivant :

```
val = 4.30606103120690604912377355 24 [...] e-1
1.8829901944567536e-3181 [method error]
8.6995407371417717e-3115 [roundoff error]
8.6995407371417732e-3115 [total error]
```

ce qui permet de conclure que, sous toutes les hypothèses faites, la valeur du juge a été calculée avec une erreur relative inférieure à 10^{-3110} et donc avec une précision en sortie suffisante pour valider les résultats des autres systèmes.

Protocole expérimental

Nous avons choisi pour faire la comparaison des systèmes un environnement de calcul qui soit publiquement accessible, raisonnablement performant et où le maximum de systèmes soient installés. Les tests ont donc été conduits sur la machine `laurent1.medicis.polytechnique.fr` du centre de calcul MÉDICIS. MÉDICIS met à disposition de ses utilisateurs 36 machines sous environnement UNIX (souvent Debian GNU/Linux [28]) ainsi qu'un ensemble de ressources logicielles, surtout dans le domaine du calcul formel. Les exigences de chacun des systèmes de calcul retenus pour nos tests (Maple en particulier n'y est pas installé sur les machines 64 bits) ont conduit au choix de `laurent1` comme meilleure machine de test possible. Techniquement la machine `laurent1` est munie de deux processeurs AMD Athlon™ MP 2200+ cadencés à 1,8MHz et de 2Go de mémoire RAM et s'exécute sous environnement Debian GNU/Linux 3.1 (noyau version 2.6.8). Dans un souci de complétude, les informations renvoyées par le système d'exploitation dans `/proc/cpuinfo` sont les suivantes :

```
fousse@laurent1:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 6
model           : 8
model name      : AMD Athlon(TM) MP 2200+
stepping        : 0
cpu MHz         : 1800.416
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
                  pse36 mmx fxsr sse syscall mp mmxext 3dnowext 3dnow
bogomips        : 3555.32
[...]
```

Le fait que la machine soit bi-processeur au lieu de mono-processeur est anecdotique dans la mesure où les systèmes testés ne savent pas profiter du parallélisme offert par deux processeurs.

Nous avons testé les performances des fonctions d'intégration des logiciels à des précisions variables qui sont en chiffres décimaux : 31, 61, 151, 302, 603, 1506, 3011, ce qui correspond environ aux précisions binaires 100, 200, 500, 1000, 2000, 5000 et 10000 bits. À part CRQ, les autres systèmes utilisent des précisions en base 10 dans leur interface, afin de ne pas les désavantager la précision décimale p_{10} correspondant à une précision binaire p_2 donnée a été arrondie supérieurement :

$$p_{10} = \left\lceil \frac{\log 10}{\log 2} p_2 \right\rceil .$$

Chaque système a été lancé avec une limite de temps CPU fixée à 1h pour éviter qu'il ne prenne trop de temps et ne pas rester bloqué sur un bug, et ce pour chaque problème d'intégration (I_1 à I_{12}) et chaque précision.

Les erreurs données dans les tableaux sont données en ulp décimaux, et on indique un échec du système à produire un résultat dans le temps imparti d'une heure par une erreur non déterminée notée « @NaN@ » et un temps d'exécution non spécifié noté « - ».

Nous donnons maintenant une description des particularités de chaque système, des scripts utilisés ainsi qu'un commentaire de ces résultats.

ARPREC

La bibliothèque ARPREC [1] est écrite en `C++` et supporte le calcul numérique en précision arbitraire pour les nombres entiers, flottants et complexes. En plus des opérations élémentaires et de fonctions spéciales (trigonométriques et autres), la distribution d'ARPREC fournit un ensemble d'applications de l'arithmétique en précision arbitraire, dont quelques routines d'intégrations décrites dans [2] et livrées avec un programme de démonstration sur un jeu de quinze intégrales. Nous avons remplacé dans ce programme les exemples d'intégrales par notre sélection des intégrales I_1 à I_{12}. Trois méthodes d'intégration sont données :

1. la méthode de Gauss-Legendre, baptisée QUADGS,
2. une méthode basée sur la fonction d'erreur $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$ baptisée QUADERF,
3. une méthode doublement exponentielle basée sur la transformation $\tanh(\sinh(x))$ baptisée QUADTS.

Dans ARPREC, l'utilisateur ne définit pas la taille de la mantisse de chaque variable de type flottant, mais initialise au début du programme la bibliothèque avec une précision, donnée en chiffres décimaux, que l'utilisateur requiert en sortie de chaque calcul élémentaire.

Les méthodes d'intégration implémentent chacune une stratégie adaptative basée sur un certain nombre de « niveaux » de points (pré-calculés). Alors que la stratégie adaptative utilisée pour intégrer dans CRQ repose sur une bissection récursive de l'intervalle d'intégration à nombre de points constant sur chaque intervalle, la stratégie d'ARPREC est de doubler le nombre de points de la méthode utilisés à chaque niveau. Le critère d'arrêt est l'obtention de la précision demandée en sortie, et celle-ci est estimée de façon empirique en comparant les résultats numériques des 3 derniers niveaux calculés. Il est à noter que si le nombre de points utilisés double d'un niveau à l'autre, le nombre de points du premier niveau n'est pas donné précisément dans [2]. Pour QUADERF et QUADTS, le nombre de points au niveau k dépend de la précision requise et serait d'environ $4 \cdot 2^k$ et $k \cdot 3,3 \cdot 2^k$, respectivement. Pour QUADGS le nombre de points au niveau k est $3 \cdot 2^k$.

Le programme d'expérimentation commence par initialiser un objet de type « *Integrate* » en respectant le choix de méthode et les paramètres donnés par l'utilisateur. Ces paramètres sont le nombre de niveaux à pré-calculer, la taille maximale prise par ces tables, la précision « primaire » et la précision « secondaire ». Ces précisions sont apparemment utilisées comme tolérances d'erreur dans le calcul des abscisses et des poids, mais leur signification exacte reste mystérieuse à la lecture du code. Il a été choisi de fixer ces tolérances à la précision demandée, et d'initialiser la précision globale à cette même valeur, plus dix bits de garde.

Du fait des choix d'implémentation, les temps de calculs d'ARPREC ne sont pas directement mesurables à ceux de CRQ puisque chacune des méthodes d'intégration d'ARPREC commence par une initialisation de tables relativement lourde, puis résout chacun des problèmes en réutilisant ces tables ; tandis que l'on a choisi pour CRQ de mesurer des temps totaux (pré-calcul et intégration proprement dite) car ils sont jugés plus représentatifs d'une utilisation typique de la bibliothèque. Il faudrait attendre que CRQ soit assimilé dans un système de calcul plus complexe pour que la réalisation de gros pré-calculs au démarrage soit justifiée.

Enfin la stratégie adaptative des méthodes d'ARPREC n'est que partiellement dynamique, dans la mesure où le nombre de niveaux utilisés dans l'intégration n'est pas augmenté au besoin mais limité par un nombre fixé au démarrage par l'utilisateur (et qui influe fortement sur le temps consacré à ce pré-calcul). Nous avons choisi d'exécuter les tests pour chacune des méthodes d'intégration d'ARPREC avec un nombre de tables pré-calculées variant de 2 à 10.

La version d'ARPREC testée est la version 2006-06-06 disponible sur le site `http://crd.lbl.gov/~dhbailey/mpdist/`.

Les résultats d'ARPREC ne sont pas inclus, les tests ayant été abandonnés après que le système se montre incapable d'intégrer correctement la fonction constante de l'exemple I_1. L'auteur de la bibliothèque n'a pas répondu favorablement à nos demandes de précisions techniques sur la bonne utilisation et le fonctionnement interne des fonctions d'intégration (lesquelles sont fournies sans documentation).

PARI/GP

PARI/GP utilise une précision interne de calcul pour les nombres flottants définie en nombre de chiffres décimaux et appelée `realprecision`. On la modifie avec la commande `\p <precision>`. La fonction d'intégration s'appelle `intnum` et utilise une méthode doublement exponentielle dans la version testée. Il est possible d'aider PARI en lui donnant des informations sur la régularité de la fonction à intégrer aux extrémités de l'intervalle d'intégration ; nous avons choisi d'utiliser les paramètres par défaut et de ne pas donner d'information à PARI, afin de ne pas avantager ce système par rapport aux autres qui ne disposent pas de cette fonctionnalité. Le fichier de test complet utilisé pour calculer l'intégrale I_5 avec 31 chiffres décimaux est le suivant :

```
\p 31
#
intnum(x = 0, 1, sin(sin(x)))
```

La commande `#` demande d'afficher le temps CPU pris par chaque calcul. Lors de l'expérimentation il est arrivé que PARI manque de mémoire ; en effet ce système utilise une taille de mémoire de pile bornée pour éviter qu'un calcul problématique fasse boguer toute l'application. Cette limite a été augmentée manuellement lorsque cela était nécessaire (argument `-s <taille>` en ligne de commande).

La version de PARI testée est la dernière version stable de numéro 2.3.0, compilée pour utiliser en tant qu'arithmétique sous-jacente la bibliothèque GMP 4.2.1.

		31	61	151	302	603	1506	3011
I_1	Temps	10ms	30ms	160ms	970ms	6s	94s	644s
	Erreur	0	0	0	0	0	0	0
I_2	Temps	20ms	60ms	350ms	2s	15s	251s	1748s
	Erreur	1.7e-1	4.2e-1	2.7e-1	2.8e-1	5.5e-2	5.4e-2	1.3e1006
I_3	Temps	20ms	70ms	400ms	3s	18s	222s	1502s
	Erreur	0	0	0	0	0	0	0
I_4	Temps	10ms	30ms	200ms	1s	7s	100s	704s
	Erreur	2.4e-1	4.8e-1	2.0e-1	1.1e-1	1.4e-1	5.9e2	5.2e1738
I_5	Temps	20ms	80ms	510ms	3s	22s	307s	2503s
	Erreur	3.4e-1	4.9e-1	5.6e-2	4.4e-1	3.9e-1	1.2e-1	1.5e-1
I_6	Temps	20ms	100ms	660ms	4s	32s	431s	-
	Erreur	2.0e-1	1.7e-1	4.9e-1	2.0e-2	3.9e-1	2.5e-1	@NaN@
I_7	Temps	0ms	30ms	180ms	1s	7s	99s	693s
	Erreur	2.4e-1	4.8e-1	2.0e-1	1.1e-1	1.4e-1	3.3e-1	3.7e-1
I_8	Temps	10ms	40ms	190ms	1s	7s	100s	766s
	Erreur	7.1e7	7.8e14	8.5e58	3.2e117	5.9e235	4.9e771	1.5e1542
I_9	Temps	30ms	150ms	980ms	5s	34s	408s	3478s
	Erreur	7.4e18	2.5e36	4.7e98	1.6e186	7.0e337	1.1e874	5.8e1448
I_{10}	Temps	20ms	50ms	340ms	2s	14s	196s	1571s
	Erreur	3.9e30	1.6e60	1.4e86	4.8e-1	4.0e-1	4.0e-2	3.7e-1
I_{11}	Temps	0ms	30ms	170ms	1s	7s	98s	705s
	Erreur	3.3e-1	3.3e-1	3.3e-1	3.3e-1	3.3e-1	3.3e-1	1.3e5
I_{12}	Temps	20ms	80ms	480ms	3s	22s	301s	2802s
	Erreur	3.1e25	2.0e54	2.7e144	1.0e295	1.7e595	9.2e1496	8.2e3001

Un avantage de PARI est sa rapidité comparée aux autres systèmes. Il a ainsi réussi à répondre à presque tous les problèmes d'intégration dans le temps requis. On peut lui reprocher de ne pas allouer dynamiquement la mémoire nécessaire (c'est à l'utilisateur de gérer la limite de la taille mémoire utilisée), et de donner des résultats dont parfois presque tous les chiffres sont faux sans prévenir l'utilisateur d'un éventuel problème.

MuPAD

Nous avons lancé les tests de MuPAD en suivant les conseils de la documentation en ligne, à savoir laisser la fonction d'intégration `numeric::quadrature` choisir le nombre de points automatiquement lorsque la précision demandée était inférieure à 200 chiffres. Dans les autres cas nous avons demandé un nombre de points d'intégration égal à la précision demandée. La méthode d'intégration choisie est Gauss-Legendre avec stratégie adaptative (le choix par défaut), et un nombre d'appels de fonction illimité. Par exemple pour calculer I_{10} avec 603 chiffres de précision on utilise le script MuPAD suivant :

```
DIGITS := 603
time (print
    (numeric::quadrature (x^2 * sin(x^3), x= 0 .. 10, MaxCalls = infinity,
                          GaussLegendre = 603)))
quit
```

La version de MuPAD testée est la 2.5.3, dernière version disponible sur le *cluster* de calcul MÉDICIS.

		31	61	151	302	603	1506	3011
I_1	Temps	60ms	0ms	60ms	660ms	1s	8s	22s
	Erreur	0	0	0	0	0	0	0
I_2	Temps	40ms	0ms	150ms	480ms	1s	1357s	-
	Erreur	1.7e-1	4.2e-1	2.7e-1	2.8e-1	5.5e-2	8.7e1504	@NaN@
I_3	Temps	0ms	140ms	230ms	180ms	29s	2145s	-
	Erreur	0	0	0	0	0	2.3e1506	@NaN@
I_4	Temps	0ms	0ms	-	-	-	-	-
	Erreur	2.4e-1	5.1e-1	@NaN@	@NaN@	@NaN@	@NaN@	@NaN@
I_5	Temps	40ms	110ms	230ms	1s	14s	3548s	-
	Erreur	3.4e-1	4.9e-1	5.6e-2	4.4e-1	3.9e-1	4.0e1506	@NaN@
I_6	Temps	0ms	0ms	310ms	2s	53s	-	-
	Erreur	2.0e-1	1.7e-1	4.9e-1	2.0e-2	3.9e-1	@NaN@	@NaN@
I_7	Temps	70ms	0ms	80ms	320ms	3s	37s	-
	Erreur	2.4e-1	4.8e-1	2.0e-1	1.1e-1	1.4e-1	7.8e1507	@NaN@
I_8	Temps	120ms	0ms	410ms	450ms	10s	62s	-
	Erreur	8.4e-2	1.4e-1	4.3e-1	6.9e-2	1.5e606	1.5e1513	@NaN@
I_9	Temps	150ms	100ms	510ms	5s	83s	1344s	-
	Erreur	7.4e-2	3.7e-1	6.5e-3	1.6e-1	2.0e-1	9.9e2339	@NaN@
I_{10}	Temps	0ms	0ms	1s	10s	68s	1442s	-
	Erreur	1.3e-2	2.8e-1	6.2e-1	4.8e-1	5.9e-1	1.4e1504	@NaN@
I_{11}	Temps	0ms	280ms	2s	6s	10s	61s	-
	Erreur	3.3e-1	3.3e-1	4.7e94	4.7e273	4.8e599	6.5e1507	@NaN@
I_{12}	Temps	0ms	3s	27s	-	-	-	-
	Erreur	2.1e-1	1.7e-1	7.0e150	@NaN@	@NaN@	@NaN@	@NaN@

Mathematica

La fonction d'intégration numérique `NIntegrate` du système de calcul Mathematica accepte un certain nombre d'options régissant son comportement, parmi lesquels les plus intéressants pour nos besoins sont `PrecisionGoal` qui indique un objectif en terme de précision relative sur le résultat renvoyé et `MaxRecursion` qui limite la profondeur maximale de la récursion dans la stratégie de bissection utilisée. On peut regretter que la précision de calcul n'est pas automatiquement sélectionnée en fonction de la précision objectif, nous l'avons arbitrairement choisie à celle-ci augmentée de 10 chiffres de garde (MuPAD fait ce choix pour nous de même que Pari).

Le choix de la profondeur maximale de récursion est de 7 par défaut, nous avons relancé le test en augmentant ce paramètre à chaque fois que Mathematica affichait un message d'erreur suggérant de l'augmenter. Un calcul de I_7 avec un objectif de 61 chiffres de précision et une profondeur de récursion bornée par 20 se fait donc par

```
Timing[NIntegrate[1 / (1 + x^2), {x, 0, 1}, {PrecisionGoal -> 61,
       MaxRecursion -> 20, WorkingPrecision -> 71}]]//InputForm
```

À cette précision la borne par défaut de 7 suffit, nous l'indiquons pour illustrer l'utilisation de ce paramètre. La version utilisée est la version 5.0.0.

		31	61	151	302	603	1506	3011
I_1	Temps	6ms	27ms	199ms	2s	-	-	-
	Erreur	0	0	0	0	@NaN@	@NaN@	@NaN@
I_2	Temps	10ms	29ms	210ms	2875s	-	-	-
	Erreur	1.2e-9	3.0e-5	3.2e-11	2.2e91	@NaN@	@NaN@	@NaN@
I_3	Temps	19ms	64ms	378ms	4s	-	-	-
	Erreur	0	0	0	0	@NaN@	@NaN@	@NaN@
I_4	Temps	110ms	354ms	2s	1306s	-	-	-
	Erreur	2.4e-1	4.8	2.0e-1	9.1e91	@NaN@	@NaN@	@NaN@
I_5	Temps	15ms	46ms	370ms	192s	-	-	-
	Erreur	2.1e-3	4.9e-1	3.5e-11	4.4e90	@NaN@	@NaN@	@NaN@
I_6	Temps	69ms	161ms	735ms	323s	-	-	-
	Erreur	3.0e-5	2.6e-5	6e-11	1.9e91	@NaN@	@NaN@	@NaN@
I_7	Temps	12ms	46ms	257ms	1381s	-	-	-
	Erreur	2.4e-1	4.8e-1	2.8e-10	9.1e91	@NaN@	@NaN@	@NaN@
I_8	Temps	258ms	810ms	4s	124s	-	-	-
	Erreur	1.5e-2	1.4e-1	3e-2	1.8e91	@NaN@	@NaN@	@NaN@
I_9	Temps	125ms	380ms	2s	2223s	-	-	-
	Erreur	7.4e-2	3.7e-1	6.5e-3	1.2e92	@NaN@	@NaN@	@NaN@
I_{10}	Temps	3s	6s	14s	1432s	-	-	-
	Erreur	1.3e-2	2.8e-1	3.7e-1	9.6e92	@NaN@	@NaN@	@NaN@
I_{11}	Temps	90ms	316ms	1s	1106s	-	-	-
	Erreur	3.3e-1	3.3e-1	3.3e-1	6.6e91	@NaN@	@NaN@	@NaN@
I_{12}	Temps	179ms	487ms	708ms	4s	-	-	-
	Erreur	1.3e23	3.1e44	3.4e142	5.3e287	@NaN@	@NaN@	@NaN@

Mathematica n'a pas donné de résultat pour les précision supérieures ou égales à 603 chiffres décimaux. Le message d'erreur systématique était le suivant :

```
                                                    0
Divide::indet: Indeterminate expression - encountered.
                                                    0
```

Mathematica a pour avantage par rapport aux autres systèmes testés de prévenir l'utilisateur d'un problème apparent de convergence de la méthode d'intégration. L'utilisateur peut alors essayer d'ajuster les divers paramètres de la fonction, même si le diagnostic d'échec ne permet pas forcément de deviner lesquels. Dans le tableau de résultats de Mathematica lorsque l'erreur est supérieure à 1 ulp le système nous a prévenu du problème ; l'erreur mentionnée dans le tableau est celle du meilleur résultat que nous avons obtenu après quelques essais d'ajustement des paramètres.

Maple

Maple est avec Pari/GP le système le plus simple pour une utilisation « naïve » de l'intégration numérique : la précision cible est inférée de la précision de travail et les autres paramètres sont choisis automatiquement. À titre d'exemple le script utilisé pour calculer I_3 avec une précision de 151 chiffres décimaux est le suivant :

```
Digits := 151:
t := time():
evalf(Int(x * ln(1 + x), x = 0..1));
time() - t;
```

Les calculs ont été réalisés avec la version 10 de Maple.

		31	61	151	302	603	1506	3011
I_1	Temps	6ms	5ms	4ms	4ms	3ms	3ms	6ms
	Erreur	0	0	0	0	0	0	0
I_2	Temps	71ms	76ms	262ms	8s	3s	50s	167s
	Erreur	1.7e-1	4.2e-1	2.7e-1	2.8e-1	5.5e-2	5.4e-2	3.8e-2
I_3	Temps	116ms	287ms	4s	16s	91s	2099s	3514s
	Erreur	0	0	0	0	0	0	0
I_4	Temps	266ms	398ms	4s	10s	36s	490s	2440s
	Erreur	2.4e-1	4.8e-1	2.0e-1	1.1e-1	1.4e-1	3.3e-1	3.7e-1
I_5	Temps	88ms	334ms	462ms	29s	5s	81s	-
	Erreur	3.4e-1	4.9e-1	5.6e-2	4.4e-1	3.9e-1	1.2e-1	@NaN@
I_6	Temps	121ms	438ms	12s	25s	150s	1874s	-
	Erreur	2.0e-1	1.7e-1	4.9e-1	2.0e-2	3.9e-1	2.5e-1	@NaN@
I_7	Temps	55ms	122ms	3s	9s	42s	479s	1717s
	Erreur	2.4e-1	4.8e-1	2.0e-1	1.1e-1	1.4e-1	3.3e-1	3.7e-1
I_8	Temps	646ms	1s	4s	10s	82s	-	2161s
	Erreur	8.4e-2	1.4e-1	4.3e-1	6.9e-2	6.6e-1	@NaN@	2.7e-1
I_9	Temps	3s	12s	123s	943s	-	-	-
	Erreur	1.1e29	8.6e58	3.4e148	2.8e299	@NaN@	@NaN@	@NaN@
I_{10}	Temps	4s	11s	25s	170s	168s	-	-
	Erreur	1.3e-2	2.3	3.7e-1	4.8e-1	5.9e-1	@NaN@	@NaN@
I_{11}	Temps	94ms	97ms	109ms	117ms	142ms	332ms	780ms
	Erreur	3.3e-1	3.3e-1	3.3e-1	3.3e-1	3.3e-1	3.3e-1	3.3e-1
I_{12}	Temps	6s	38s	371s	2618s	-	-	-
	Erreur	2.1e-1	1.7e-1	1.7e-1	2.9e-1	@NaN@	@NaN@	@NaN@

Maple calcule dans l'ensemble des résultats suffisamment précis, sauf pour I_9 où il donne des résultats où presque tous les chiffres sont faux, et ceci sans informer l'utilisateur d'un éventuel problème de convergence qu'il aurait détecté. C'est aussi le système testé le plus lent.

CRQ

La bibliothèque CRQ se distingue des autres systèmes en ceci qu'elle n'est disponible justement qu'en tant que bibliothèque et n'a pas d'interface texte ou graphique pour faciliter les calculs (Pari/GP fournit à l'utilisateur aussi bien une bibliothèque qu'une interface texte). Nous avons donc implémenté les fonctions à intégrer dans un programme de test dont l'interface en ligne de commande sert à sélectionner les différents paramètres (nombre de points, précision de calcul, méthode d'intégration utilisée, choix de l'intégrale à calculer).

Les deux méthodes d'intégration testées pour CRQ sont la méthode de Gauss-Legendre avec stratégie adaptative notée $\mathrm{CRQ}_{\mathrm{gl}}$ et la méthode de Newton-Cotes avec stratégie adaptative (notée $\mathrm{CRQ}_{\mathrm{nc}}$). Pour chaque méthode, précision et problème d'intégration, plusieurs exécutions ont été effectuées en variant le nombre de points n de la méthode d'intégration, dans le but de trouver le nombre de points optimal (au sens du temps de calcul). Cette détermination du nombre de points optimal permet d'une part de proposer un algorithme de choix automatique du nombre de points en fonction de la précision et l'ajouter à CRQ (dans une version ultérieure), et d'autre part de confirmer expérimentalement la comparaison théorique des méthodes de Newton-Cotes et de Gauss-Legendre.

		31	61	151	302	603	1506
I_1	Temps	0ms	11ms	1ms	92ms	183ms	575ms
	Erreur	0	0	0	0	0	0
I_2	Temps	9ms	14ms	99ms	235ms	3s	21s
	Erreur	3.7e-1	6.2e-1	5.2e-1	2.1e-1	1.7	5.4e-2
I_3	Temps	8ms	25ms	145ms	729ms	4s	78s
	Erreur	0	0	0	0	0	0
I_4	Temps	20ms	88ms	907ms	6s	58s	-
	Erreur	7.4	2.8e-1	4.0e-1	6.2	7.6	@NaN@
I_5	Temps	6ms	16ms	103ms	514ms	3s	54s
	Erreur	9.5e-1	1.3	1.4e-1	3	4	1.2
I_6	Temps	16ms	35ms	219ms	1s	7s	95s
	Erreur	6.0e-1	3.7e-1	7.9e-1	1	2.3	3.5e-1
I_7	Temps	11ms	10ms	58ms	294ms	1s	29s
	Erreur	4.5e-1	2.8e-1	4.0e-1	3.1	1.1	2.5
I_8	Temps	3ms	35ms	172ms	847ms	5s	85s
	Erreur	2.3	1.8	7.3e-1	4.6e-1	1.8	7.0e-2
I_9	Temps	22ms	64ms	445ms	2s	18s	279s
	Erreur	2.3	1.5	8.9e-1	9.3e-1	5.6	1.5
I_{10}	Temps	2s	861ms	7s	16s	42s	305s
	Erreur	8	3	1.2	7.2	7.2e1	3.4e1
I_{11}	Temps	18ms	81ms	777ms	5s	46s	-
	Erreur	2.6	2	1	3.1	3	@NaN@
I_{12}	Temps	26ms	110ms	1s	11s	108s	3138s
	Erreur	1	2.6e-2	9.2e-1	3.6	4.2	3.5

21 Conclusion

Les résultats (ou le manque de résultat) pour certains systèmes confirment la difficulté du calcul numérique d'intégrales à grande précision. Les erreurs mesurées sont parfois surprenantes et montrent les limites de l'intégration numérique automatique telle qu'elle est disponible dans les systèmes actuels — il n'est pas impossible que l'on puisse trouver une bonne combinaison de paramètres pour « aider » un système sur une intégrale qu'il n'a pas su calculer dans nos tests, ou qu'il a calculé avec une grande erreur.

Certains des résultats donnés dans les tableaux sont accompagnés d'une notification par le système que la procédure d'intégration n'a pas convergé, mais cela n'est pas toujours le cas. Quelques erreurs « flagrantes » sont passées sous silence et le résultat retourné à l'utilisateur est alors presque complètement faux (c'est le cas des résultats de I_9 pour Maple pour toutes les précisions où il n'y a toujours qu'au plus 3 chiffres corrects).

Concernant les performances de CRQ la méthode adaptative qui est sans garantie sur le résultat donne des résultats très satisfaisants : l'erreur est toujours bornée par quelques dizaines d'ulp au plus, et très souvent le résultat renvoyé est un arrondi fidèle de la valeur exacte. De plus les temps mesurés se comparent favorablement aux systèmes ayant fourni une réponse raisonnablement juste (CRQ est souvent plus rapide d'un facteur environ 2 par rapport à Mathematica, de même par rapport à PARI/GP). Nous rappelons cependant que CRQ a bénéficié pour chaque problème de plusieurs exécutions et d'une présélection du nombre de points optimal au niveau du temps de calcul. Une fois que CRQ sera doté d'une stratégie de sélection automatique du nombre de point il sera judicieux de refaire des tests.

Conclusion

Dans cette thèse nous avons étudié le problème de l'intégration numérique en précision arbitraire. Les deux méthodes de Newton-Cotes et de Gauss-Legendre ont été analysées avec le double objectif de savoir maîtriser les erreurs, d'origines diverses, qui apparaissent dans ce type de calcul, et bien entendu d'intégrer efficacement. Il a fallu pour cela aborder chaque méthode d'intégration sous plusieurs aspects : au niveau mathématique d'abord pour adapter voire retrouver certains résultats « bien connus » à nos desseins d'analyse d'erreur, mais aussi d'un point de vue algorithmique pour voir quels procédés utilisés en arithmétique s'appliquent avec profit. Concernant la méthode de Gauss-Legendre plus particulièrement, son examen nous a amené à regarder le raffinement des racines réelles d'un polynôme (il s'agit là d'un problème essentiel en analyse numérique). Les concepts de l'arithmétique flottante ont naturellement joué un rôle crucial dans l'analyse d'erreur proprement dite.

Nos contributions algorithmiques sont les suivantes. Nous avons proposé une procédure de calcul d'intégrale par la méthode de Newton-Cotes avec calcul simultané d'une borne d'erreur sur le résultat rendu. Il s'agit de la borne donnée dans le théorème 4 (p. 41). Nous avons proposé un algorithme asymptotiquement rapide de calcul de ces coefficients se basant sur des méthodes d'évaluation polynomiale rapide. L'examen de la méthode de Gauss-Legendre nous a permis de donner aussi un algorithme de calcul avec borne d'erreur qui est donné dans le théorème 8 (p. 54). Nous insistons sur le fait qu'il s'agit aussi bien pour Newton-Cotes que pour Gauss-Legendre d'une borne effective et non pas un ordre de grandeur sur l'erreur, et qu'elle est générique en fonction de la précision et du nombre de points d'intégration. Nous avons regardé en détail divers algorithmes de raffinement de racines réelles de polynômes (l'itération de Newton scalaire et par intervalle, la méthode de la sécante, la dichotomie) en proposant des heuristiques explicites permettant de s'assurer de la convergence en pratique de la méthode.

Ces résultats sont utilisés dans notre contribution logicielle sous la forme d'une bibliothèque d'intégration numérique avec erreur bornée baptisée *« Correctly Rounded Quadrature »* (CRQ) et distribuée sous la licence GNU LGPL 2.1. On peut la télécharger librement depuis l'adresse `http://komite.net/laurent/soft/crq/`. Nous en donnons une comparaison avec d'autres systèmes d'intégration numérique.

Sur le plan théorique un certain nombre de problèmes restent ouverts. À propos du calcul des coefficients de Newton-Cotes d'abord il serait satisfaisant de vérifier si l'algorithme « rapide » est de complexité théorique optimale (ce que nous soupçonnons étant données les tailles mesurées des résultats calculés). L'algorithme d'évaluation multi-points rapide utilisé pour le calcul des coefficients de Newton-Cotes pourrait être appliqué à la méthode de Gauss-Legendre. Il serait en effet possible de calculer en une seule évaluation multi-points les valeurs de $P(x_1), P(x_2), \ldots, P(x_n)$ où les x_i désignent les approximations courantes des racines $u_1, u_2, \ldots, u_n$ pour accélérer le raffinement par la méthode de la sécante (ou de Newton en appliquant la même idée au calcul de

$P'(x_1), P'(x_2), \ldots, P'(x_n)$). La difficulté théorique à résoudre est celle de l'analyse d'erreur de l'évaluation multi-points en nombres flottants.

Une suite possible au travail effectué serait l'étude d'autres méthodes d'intégration numérique. En particulier la famille des méthodes de Gauss (Gauss-Tschebyscheff, Gauss-Laguerre, Gauss-Hermite) qui suivent la même structure générale que la méthode de Gauss-Legendre avec une fonction de poids w différente est susceptible de se voir appliquer la méthodologie du chapitre 4. De même la méthode de Gauss-Kronrod nous semble suffisamment voisine des méthodes de Gauss pour y transposer nos résultats.

Sur le sujet de l'intégration numérique plus généralement les méthodes autres que celle de Gauss pourraient bénéficier de l'approche « calculs numériques sûrs » que nous avons prise ici. Nous pensons en particulier à la transformation double-exponentielle (DE) : est-il envisageable de chercher une méthode DE plus fiable, c'est-à-dire avec erreur bornée ?

Une perspective à plus haut niveau serait cette application naturelle de l'opération d'intégration numérique en analyse qu'est la résolution des équations différentielles. Lorsqu'un système d'équations différentielles n'admet pas de solution exprimable sous forme d'une formule close, on a recours (tout comme pour l'intégration) à des méthodes numériques de calcul. Il n'est pas impossible qu'une intégration numérique fiable ait un rôle à jouer aussi dans ce domaine.

Citons enfin la perspective ambitieuse de valider les démonstrations du présent mémoire à l'aide d'un outil de preuve formelle tel que Coq [4]. On peut considérer que la preuve formelle apporte un certain degré de confiance supplémentaire dans les démonstrations (par rapport à une relecture humaine), et il s'agit en tout état de cause d'un prérequis avant une éventuelle implémentation prouvée des algorithmes présents dans CRQ. La validation formelle de l'algorithme de sommation de Demmel et Hida [13] donne un exemple à petite échelle du travail à fournir dans ce sens, on peut la consulter dans [17].

Bibliographie

[1] David H. Bailey, Yozo Hida, Karthik Jeyabalan, Xiaoye S. Li, and Brandon Thompson. ARPREC version date 2006-06-06. `http://crd.lbl.gov/~dhbailey/mpdist/`, June 2006.

[2] David H. Bailey and Xiaoye S. Li. A comparison of three high-precision quadrature schemes. In *Proceedings of the RNC'5 conference (Real Numbers and Computers)*, pages 81–95, September 2003. `http://www.ens-lyon.fr/LIP/Arenaire/RNC5`.

[3] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to PARI-GP*, 2006. `http ://pari.math.u-bordeaux.fr/`.

[4] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art : The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[5] Alin Bostan, Philippe Flajolet, Bruno Salvy, and Éric Schost. Fast computation of special resultants. *Journal of Symbolic Computation*, 41(1) :1–29, January 2006.

[6] R. P. Brent. The complexity of multiple-precision arithmetic. In R. S. Anderssen and R. P. Brent, editors, *The Complexity of Computational Problem Solving*, pages 126–165. University of Queensland Press, 1976.

[7] Richard P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1975)*, pages 151–176. Academic Press, New York, 1976.

[8] D. Calvetti, G. H. Golub, W. B. Gragg, and L. Reichel. Computation of Gauss-Kronrod quadrature rules. *Mathematics of Computation*, 69(231) :1035–1052, 2000.

[9] Jean-Luc Chabert, editor. *A History of Algorithms : From the Pebble to the Microchip*. Springer-Verlag, New York, 1999.

[10] George E. Collins and Alkiviadis G. Akritas. Polynomial real root isolation using Descarte's rule of signs. In *SYMSAC '76 : Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 272–275, New York, NY, USA, 1976. ACM Press.

[11] Catherine Daramy-Loirat, David Defour, Florent de Dinechin, Matthieu Gallet, Nicolas Gast, and Jean-Michel Muller. *CR-LIBM : A library of correctly rounded elementary functions in double-precision*, September 2005.

[12] Philip J. Davis and Philip Rabinowitz. *Methods of numerical integration*. Academic Press, New York, 2nd edition, 1984.

[13] James Demmel and Yozo Hida. Accurate floating point summation. `http://www.cs.berkeley.edu/~demmel/AccurateSummation.ps`, May 2002.

[14] J. Dieudonné. *Calcul Infinitésimal*. Hermann, Paris, 1968.

[15] W. J. Ellison et M. Mendès-France. Les nombres premiers. *Actualités Scientifiques et Industrielles*, 1366, 1975.

[16] Laurent Fousse. Correctly rounded Newton-Cotes quadrature. Research Report 5605, June 2005. 20 pages.

[17] Laurent Fousse and Paul Zimmermann. Accurate summation : Towards a simpler and formal proof. In *Proceedings of the RNC'5 conference (Real Numbers and Computers)*, pages 97–108, September 2003. `http://www.ens-lyon.fr/LIP/Arenaire/RNC5`.

[18] Walter Gander and Walter Gautschi. Adaptative quadrature – revisited. Research Report 306, Departement Informatik, ETH Zürich, August 1998.

[19] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1) :5–48, 1991.

[20] G. H. Golub and J.H. Welsch. Calculation of Gauss quadrature rules. *Mathematics of Computation*, 23 :221–230, 1969.

[21] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2nd edition, 2002.

[22] IEEE standard for binary floating-point arithmetic. Technical Report ANSI-IEEE Standard 754-1985, New York, 1985. approved March 21, 1985 : IEEE Standards Board, approved July 26, 1985 : American National Standards Institute, 18 pages.

[23] Dirk P. Laurie. Computation of Gauss-Kronrod quadrature rules. *Mathematics of Computation*, 66(219) :1133–1145, 1997.

[24] V. Lefèvre and J.-M. Muller. Worst cases for correct rounding of the elementary functions in double precision. In Neil Burgess and Luigi Ciminiera, editors, *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 111–118, Vail, Colorado, 2001. IEEE Computer Society Press, Los Alamitos, CA.

[25] Masatake Mori. Discovery of the Double Exponential Transformation and Its Developments. *Publ. RIMS*, 41 :897–935, 2005.

[26] Walter Oevel. Numerical computations in MuPAD 1.4. *mathPAD*, 8(1), 1998.

[27] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[28] The Debian Project. Debian GNU/Linux 3.1 (sarge). `http://www.debian.org/`, June 2005.

[29] The Netherlands Radboud University, Nijmegen. `http://www.cs.ru.nl/~milad/manydigits/`.

[30] Nathalie Revol. Interval Newton iteration in multiple precision for the univariate case. Research Report RR-4334, INRIA, December 2001.

[31] L.F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stress in a masonry dam. In *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, volume 83, pages 335–336. The Royal Society, March 1910.

[32] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of a polynomial real roots. *Journal of Computational and Applied Mathematics*, 162(1) :33–50, 2003.

[33] Ken'ichiro Tanaka, Masaaki Sugihara, and Kazuo Murota. Numerical indefinite integration by double exponential sinc method. *Mathematics of Computation*, 74(250) :655–679, 2005.

[34] The Spaces project. The MPFR library, version 2.0.1. `http ://www.mpfr.org/`, 2002.

[35] Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw-Curtis ? submitted to SIAM Review.

[36] M. Vincent. Sur la résolution des équations numériques. *Journal de Mathématiques Pures et Appliquées*, pages 341–372, 1836.

[37] J. von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.

[38] Paul Zimmermann and Bruce Dodson. 20 years of ECM. In F. Hess, S. Pauli, and M. Pohst, editors, *Proceedings of the 7th Algorithmic Number Theory Symposium (ANTS VII)*, volume 4076, pages 525–542, Berlin Heidelberg, 2006.

[39] Abraham Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Trans. Math. Softw.*, 17(3) :410–423, 1991.

Résumé

L'intégration numérique est une opération fréquemment disponible et utilisée dans les systèmes de calcul numérique. Nous nous intéressons dans ce mémoire à la maîtrise des erreurs commises lors d'un calcul numérique d'intégrale réelle à une dimension dans le contexte de la précision arbitraire pour les deux méthodes d'intégration que sont Newton-Cotes et Gauss-Legendre. Du point de vue algorithmique nous proposons pour chacune des méthodes une procédure de calcul avec une borne effective sur l'erreur totale commise. Dans le cadre de l'étude de la méthode de Gauss-Legendre nous avons étudié les algorithmes connus de raffinement de racines réelles d'un polynôme (la méthode de la sécante, l'itération de Newton, la dichotomie), et nous en avons proposé des heuristiques explicites permettant de s'assurer en pratique de la convergence.

Les algorithmes proposés ont été implémentés dans une bibliothèque d'intégration numérique baptisée *« Correctly Rounded Quadrature »* (CRQ) disponible à l'adresse `http://komite.net/laurent/soft/crq/`. Nous comparons CRQ avec d'autres logiciels d'intégration dans ce mémoire.

Mots-clés: intégration numérique, précision arbitraire, arithmétique flottante, erreur bornée

Abstract

Numerical integration is commonly available in numerical computation systems. We study in this thesis the error on the result of a numerical quadrature using the Newton-Cotes and Gauss-Legendre methods for a monodimensional real function in the context of arbitrary precision. From the algorithmic point of view we give for each method a computation procedure with a guaranteed bound on the error. For the analysis of the Gauss-Legendre method we studied polynomial root refinement schemes (secante, Newton's iteration, dichotomy) and we gave heuristics to make sure the method converges in practice.

The algorithms proposed in this thesis were written in a numerical quadrature library called "Correctly Rounded Quadrature" (CRQ) available at `http://komite.net/laurent/soft/crq/`. We also give a comparison of CRQ with other numerical integration softwares.

Keywords: numerical integration, arbitrary precision, floating-point arithmetics, bounded error