Chapitre 4

Calculer avec les nombres réels

Fredrik JOHANSSON
traduit de l'anglais par Xavier CARUSO

Le calcul sur machine avec des nombres réels pose des difficultés fondamentales, liées d'une part aux questions de calculabilité elles-mêmes et, d'autre part, aux problèmes pratiques d'efficacité et de suivi de précision. Dans ce chapitre, nous faisons le point sur les problématiques de ce domaine et sur les concepts fondamentaux qui ont été introduits pour les résoudre. Nous présentons également quelques outils (comme les différentes manières de représenter les nombres réels) qui permettent de surmonter en pratique les difficultés qui apparaissent.

4.1 Introduction

Les ordinateurs ont été inventés avec l'ambition de manipuler les nombres. Pourtant, bien souvent, ils semblent n'avoir qu'une faible maîtrise des nombres réels. L'exemple suivant, extrait d'une session de SageMath, illustre parfaitement ceci :

```
sage: sqrt(2.0)/2.0 == 1.0/sqrt(2.0)
False
sage: sqrt(2.0)/2.0; 1.0/sqrt(2.0)
0.707106781186548
0.707106781186547
```

Bien sûr, le test d'égalité $\sqrt{2}/2 = 1/\sqrt{2}$ a échoué car les calculs ont été menés avec des nombres réels approchés au lieu de nombres réels exacts. Dans certaines situations, une erreur sur la 15^{e} décimale, comme celle

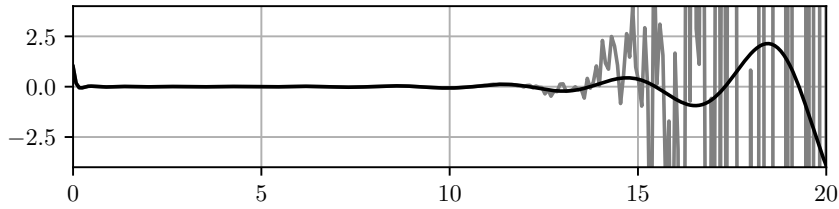

FIGURE 4.1 – *En noir : le graphe de la fonction hypergéométrique* ${}_1F_1(-50, 3, x)$ *sur l'intervalle* $[0, 20]$. *En gris : le graphe (erroné) de la même fonction calculé par* `scipy.special.hyp1f1`. *La différence entre les deux graphes provient des erreurs d'arrondis.*

observée ci-dessus, peut conduire *in fine* à des aberrations totales, comme illustré par la figure 4.1. Pourquoi autorisons-nous de tel crimes contre les mathématiques ? Plusieurs raisons acceptables peuvent être avancées :

1. les nombres réels exacts sont intrinsèquement des objets non calculables ; typiquement, un nombre réel tiré au hasard (comme 2,65323025327443 . . .) contient une quantité infinie d'information qui ne peut tenir dans la mémoire d'un ordinateur
2. si nous choisissons de nous limiter à un sous-ensemble « raisonnable » de nombres réels que l'on sait représenter par des structures finies, il existera toujours un certain nombre d'opérations impossibles à implémenter
3. et, même dans les cas où nous disposons d'algorithmes exacts pour les opérations les plus courantes, ceux-ci sont généralement coûteux et difficiles à implémenter.

Il y a donc un véritable besoin de considérer des nombres réels approchés, avec toutes les difficultés que cela implique.

L'objectif de ce texte est de présenter les difficultés inhérentes au calcul approché sur les nombres réels, sans toutefois entrer dans les rouages de l'arithmétique à virgule flottante (qui est le standard le plus répandu) ni dans ceux de l'implémentation. Nous verrons que certains problèmes sont intrinsèquement difficiles alors que d'autres peuvent être élégamment résolus avec les outils adéquats.

Ces outils, qui permettent de calculer avec des nombres réels approchés de façon plus fiable que l'arithmétique à virgule flottante classique, sont nombreux : il y a, par exemple, l'arithmétique multi-précision, l'arithmétique d'intervalles et différents types d'arithmétique paresseuse ou symbolique. Chacun d'entre eux a ses propres avantages et inconvénients.

Plusieurs d'entre eux sont disponibles à l'utilisation dans le logiciel SageMath. Nous encourageons la lectrice à tester par elle-même les nombreux exemples qui seront présentés dans la suite de ce texte. Pour aller plus loin, nous conseillons le livre *Calcul mathématique avec Sage* [1] dont plusieurs chapitres sont consacrés aux méthodes numériques.

4.2 Nombres algébriques

D'une manière générale, *calculer* avec une structure mathématique donnée S suppose que l'on dispose, d'une part, d'un moyen de représenter les éléments de S sous forme numérique, *i.e.*, par des suites de bits, et, d'autre part, d'algorithmes mettant en œuvre les opérations sur les éléments de S dans la représentation suscitée. Une structure S pour laquelle on dispose de ces prérequis est dite *calculable* ou *effective*. Dans la suite, nous emploierons le mot *effectif* pour ce concept et réservons l'utilisation du mot *calculable* pour une notion plus technique qui sera introduite plus tard.

Théorème 4.2.1. *L'anneau des entiers $\mathbb{Z}$ muni des opérations $\{+, -, \times\}$ et des prédicats de comparaison $\{=, \neq, \leqslant, <, \geqslant, >\}$ est effectif.*

La manière usuelle de représenter les entiers est d'utiliser des suites de chiffres (en base 10, ou plus traditionnellement pour les ordinateurs, en base 2 ou 2^{64}). Les algorithmes d'addition, de soustraction et de multiplication sont nombreux, les plus simples d'entre eux étant ceux des livres d'école où l'on effectue l'opération chiffre par chiffre en partant de la droite [1]. Bien entendu, beaucoup d'autres opérations sur les entiers, au delà de celles mentionnées dans le théorème 4.2.1, sont effectives : on peut citer la valeur absolue $|\cdot|$, le plus grand commun diviseur (pgcd), la factorisation en nombre premiers, *etc.*

Voici un corollaire simple du théorème 4.2.1 :

Théorème 4.2.2. *Le corps des nombres rationnels $\mathbb{Q}$ (muni des quatre opérations $\{+, -, \times, /\}$ et des prédicats de comparaison) est effectif.*

En transpirant davantage, il est aussi possible de démontrer un résultat analogue pour les nombres algébriques.

Théorème 4.2.3. *Le corps $\overline{\mathbb{Q}}$ formé des nombres complexes x qui sont solutions d'une équation algébrique de la forme $f(x) = 0$ avec $f \in \mathbb{Q}[x]$ (f n'étant pas le polynôme nul), est effectif pour les opérations de corps, la norme et la conjugaison complexe.*

1. Nous évoquerons brièvement des algorithmes plus rapides dans la partie 4.4.4 mais, pour ce qui concerne l'aspect effectif de $\mathbb{Z}$, ces considérations ne sont pas pertinentes.

À partir de là, on s'aperçoit que l'anneau des polynômes à coefficients dans $\overline{\mathbb{Q}}$ ou celui des matrices à coefficients dans $\overline{\mathbb{Q}}$ est aussi effectif.

Revenons un instant sur l'exemple de l'introduction. Il se trouve que SageMath propose une implémentation du corps $\overline{\mathbb{Q}}$ que nous pouvons utiliser à la place des nombres réels flottants. Voici le résultat que l'on obtient dans ce cas :

```
sage: x = QQbar(2)
sage: sqrt(x)/2 == 1/sqrt(x)
True
```

Soulignons bien que le corps $\overline{\mathbb{Q}}$ ne permet pas d'émuler parfaitement $\mathbb{R}$ ou $\mathbb{C}$, des constantes comme π ou e y étant inaccessibles. Il est cependant possible de faire déjà beaucoup de mathématiques à l'intérieur de $\overline{\mathbb{Q}}$; notamment, la majeure partie des problèmes calculatoires d'origine géométrique peut se traiter entièrement à l'aide de nombres algébriques.

L'inconvénient des nombres algébriques est le coût calculatoire qu'ils nous obligent à payer pour leur manipulation. L'évaluation de $1+1$ dans le corps `QQbar` de SageMath prend des milliers de cycles machine alors que tout processeur est *a priori* capable de réaliser plusieurs additions à un chiffre par cycle. Même en ignorant ces surcoûts qui ne jouent finalement que par un facteur constant, les algorithmes de calcul exact sur les nombres algébriques ont généralement une complexité asymptotique médiocre, même pour les tâches les plus simples (comme l'addition). Par exemple, ils nécessitent, la plupart du temps, de calculer un polynôme annulateur explicite de chaque nombre manipulé, polynômes qui sont souvent énormes.

Exemple 4.2.4. *Soit $x = \sqrt{2} + \sqrt{3} + \ldots + \sqrt{p_N}$ le nombre défini comme la somme des racines carrés de N premiers nombres premiers. Le polynôme minimal de x sur $\mathbb{Q}$ est de degré 2^N.*

Voici une expérience simple qui montre les limitations de `QQbar` :

```
sage: x = QQbar(sum(sqrt(nth_prime(n+1)) for n in range(6)))
sage: %time x - (x - 1) - 1 == 0
CPU times: user 8.98 s, sys: 14.4 ms, total: 9 s
Wall time: 9.17 s
True
```

Si l'on remplace `range(6)` par `range(7)`, le temps de calcul est multiplié par plus de 20 ! Dans ce cas particulier, il y aurait de bien meilleures façons de faire le calcul. Par exemple, nous aurions pu utiliser l'*anneau*

symbolique `SR` de SageMath à la place de `QQbar`. Toutefois, dans d'autres situations, `SR` pourrait être moins efficace, voire inutilisable [2].

Malgré l'existence de tels exemples, il ne faut pas exagérer les défauts de l'arithmétique exacte. Il serait trop facile de prendre un algorithme classique (comme, disons, l'élimination de GAUSS), de se rendre compte qu'il est peu performant dans le cas de l'arithmétique exacte, et de conclure hâtivement que le calcul exact ne peut conduire qu'à une impasse. Au contraire, il existe souvent des méthodes conçues spécialement pour l'arithmétique exacte qui, dans certaines situations, peuvent avoir d'excellentes performances, bien meilleures que celles de l'arithmétique approchée [3].

4.2.1 Logique et décidabilité

Il devient nettement plus difficile de mener des calculs algébriques lorsque interviennent, en même temps, des formules logiques avec quantificateurs. Dans ce contexte, plutôt que le mot *effectif*, nous utiliserons le terme *décidable* qui nous paraît plus approprié. Le célèbre 10e problème de HILBERT pose la question suivante sur les nombres entiers : existe-t-il un algorithme qui décide si une équation diophantienne donnée admet (au moins) une solution entière ? MATIYASEVICH donna une réponse négative à la question de HILBERT en 1970.

Théorème 4.2.5 (Corollaire du théorème M-R-D-P)**.** *Il existe un polynôme $f \in \mathbb{Z}[x_1, \ldots, x_n]$ pour lequel aucun algorithme ne peut décider si l'équation $f(x_1, \ldots, x_n) = 0$ a une solution $(x_1, \ldots, x_n) \in \mathbb{Z}^n$.*

L'ingrédient principal derrière la démonstration du théorème M-R-D-P est le suivant : les équations diophantiennes sont des objets mathématiques suffisamment généraux pour coder le comportement de machines de TURING arbitraires et ainsi rendre compte, d'une certaine manière, des limitations fondamentales de la théorie de la calculabilité à l'instar du problème de l'arrêt de TURING ou du théorème d'incomplétude de GÖDEL.

À l'opposé, lorsque l'on remplace $\mathbb{Z}$ ou $\mathbb{Q}$ par un corps algébriquement clos comme $\overline{\mathbb{Q}}$, de puissants résultats de décidabilité sont disponibles, de sorte que certains problèmes de décision deviennent plus simples.

2. L'exemple présenté ici est, en réalité, trivial à traiter dans le cas de l'anneau symbolique car les éléments qui apparaissent sont des sommes de radicaux simples qui s'éliminent directement. Mais, de manière générale, les nombres algébriques ne s'expriment pas à l'aide de radicaux ou peuvent avoir des expressions plus compliquées qui rendent leur manipulation plus délicate.

3. À ce sujet, on peut comparer `random_matrix(QQ, n, n).det()` (très optimisé) et `random_matrix(RR, n, n).det()` en SageMath pour différentes valeurs de n.

Par exemple, si $R = \overline{\mathbb{Q}} \cap \mathbb{R}$ désigne le corps des nombres algébriques réels, il existe un algorithme qui, étant donné un polynôme $f \in R[x_1, \ldots, x_n]$ décide si $f(x_1, \ldots, x_n) \geqslant 0$ pour tous $x_1, \ldots, x_n \in R$. Dans le même registre, un résultat plus fort est le théorème d'élimination des quantificateurs de TARSKI, démontré dans les années 1950, que nous énonçons brièvement ci-dessous.

Théorème 4.2.6 (TARSKI)**.** *Toute formule logique du premier ordre (i.e. formée à partir des opérations booléennes et des quantificateurs $\forall, \exists$) en n variables $x_1, \ldots, x_n \in R$ ne faisant intervenir que des égalités et inégalités polynômiales est décidable.*

Une conséquence du théorème de TARSKI est que les énoncés de la géométrie euclidienne sont décidables (une fois qu'ils ont été proprement formalisés).

L'algorithme originel de TARSKI réalisant l'élimination des quantificateurs avait une complexité lovecraftienne [4]. En 1975, COLLINS inventa la méthode de décomposition cylindrique algébrique (CAD), ce qui lui permit de résoudre le problème de l'élimination des quantificateurs en temps « seulement » doublement exponentiel (*i.e.* $2^{2^{O(n)}}$ où n est le nombre de variables) dans le pire cas. Les algorithmes de calcul exact en géométrie, tels que la CAD, sont aujourd'hui disponibles dans de nombreux logiciels de calcul et sont utilisés pour des applications variées comme la planification de mouvements de robots.

4.3 Nombres réels

De par leur définition, les nombres réels sont le lieu propice aux constructions par passage à la limite et, de ce fait, à la définition de constantes telles que π, de fonctions transcendantes comme l'exponentielle et à la mise en place de l'analyse avec le calcul différentiel (dérivation, intégration), la sommation des séries infinies, *etc.*

Une suite de CAUCHY $a_0, a_1, \ldots$ de nombres rationnels possède une limite $\lim_{n\to\infty} a_n \in \mathbb{R}$ et, de fait, $\mathbb{R}$ peut être défini formellement comme l'ensemble des classes d'équivalence de suite de CAUCHY de nombres rationnels. De manière à peine moins formelle, on peut aussi définir $\mathbb{R}$ comme l'ensemble des « nombres avec une infinité de chiffres après la virgule », à l'instar de 3,141 ... (ou plus exactement des classes d'équivalence de telles écritures car, par exemple, $0{,}999\ldots = 1{,}000\ldots$).

4. De l'horreur cosmique de l'inconnu des histoires de H. P. LOVECRAFT. Le terme technique plus approprié serait *complexité non élémentaire*.

Bien entendu, il n'est pas possible de stocker une suite infinie $a_0, a_1, \ldots$ de manière exhaustive sur un ordinateur et, en général, il est même impossible de la coder de manière implicite. L'indénombrabilité de $\mathbb{R}$, démontrée par CANTOR, implique que la plupart des nombres réels ne peuvent être uniquement déterminés par une quantité finie d'information. Par conséquent, le corps $\mathbb{R}$ n'est, à l'évidence, pas effectif. Dans tous les cas, calculer avec $\mathbb{R}$ signifie donc nécessairement calculer avec un sous-ensemble restreint et dénombrable de $\mathbb{R}$, par exemple $R = \overline{\mathbb{Q}} \cap \mathbb{R}$ ou éventuellement d'autres ensembles plus gros incluant des nombres transcendants comme $R(\pi, e, \log(2))$.

4.3.1 Nombres réels calculables

Une possibilité pour décrire un nombres réel de manière effective est de se donner un algorithme qui calcule une suite de CAUCHY le représentant. Formellement, ceci conduit à la définition suivante d'un *nombre réel calculable*.

Définition 4.3.1. *Un* nombre réel calculable *est un nombre réel x pour lequel il existe un programme (dans le sens d'une machine de* TURING*) qui, étant donnée une précision $p \in \mathbb{Z}$, renvoie un nombre rationnel $\widehat{x}$ tel que $|x - \widehat{x}| < 2^{-p}$.*

Plus généralement, on définit les fonctions calculables comme suit.

Définition 4.3.2. *Une* fonction calculable *(sur $\mathbb{R}$) est une fonction f pour laquelle il existe un programme qui, étant donnés une précision $p \in \mathbb{Z}$ et un programme calculant un nombre réel (calculable) x, renvoie un nombre rationnel $\widehat{y}$ tel que $|f(x) - \widehat{y}| < 2^{-p}$.*

Les définitions précédentes s'étendent sans difficulté aux nombres complexes, aux fonctions de plusieurs variables, *etc*. Il est à noter également que les nombres calculables peuvent être considérés comme des fonctions calculables n'acceptant aucun argument (une fonction de 0 variable). Enfin, il est facile de vérifier que la composée de deux fonctions calculables est calculable.

Exemple 4.3.3. *L'addition est une fonction calculable. En effet, étant donnés une précision $p \in \mathbb{Z}$ et des programmes calculant les nombres réels x et y, on peut obtenir des approximations de x et y à 2^{-p-1} près (en appelant les programmes suscités avec la précision $p + 1$) et ajouter ces approximations.*

Les nombres calculables forment un sous-ensemble dénombrable de $\mathbb{R}$. Les nombres algébriques et les fonctions algébriques sont toutes calculables,

mais il existe aussi des nombres et des fonctions calculables qui sont transcendantes. C'est le cas, par exemple, du nombre π étant donné qu'il peut être approché par les sommes partielles de la série $4\sum_{k=0}^{\infty}(-1)^n/(2n+1)$. De la même manière, la fonction exponentielle est calculable car elle peut être approchée par sa série de TAYLOR. Toutefois, il n'est pas vrai que toutes les fonctions simples soient calculables : nous verrons dans la partie 4.3.3 une restriction essentielle à la calculabilité.

4.3.2 Expressions symboliques

Une autre possibilité pour représenter les nombres réels consiste à utiliser des formules symboliques. Par exemple, en supposant que les entiers, les opérations arithmétiques et la constante π sont des symboles connus, nous pouvons former la quantité $\sqrt{2}+\frac{5}{3}\pi$ en l'encodant, au choix, comme une simple chaîne de caractères ou sous forme d'arbre comme suit : $(+,(\sqrt{\cdot},2),(\times,(/,5,3),\pi))$.

Les nombres réels pouvant être décrits de cette manière (à partir d'un langage fixé *a priori*) sont parfois appelés les *nombres réels symboliques* ou *nombres réels définissables* [5]. Les expressions symboliques sont trivialement effectives dans le sens où il est toujours possible de calculer la valeur qu'elles représentent en enchaînant les opérations décrites. Cette affirmation doit toutefois être prise avec des pincettes : de même que l'argent n'a de véritable valeur que lorsqu'il peut être échangé contre des biens ou des services, une expression symbolique ne représente un nombre réel calculable que si on peut l'interpréter comme une recette pour fabriquer un programme complexe à partir de programmes élémentaires qui représentent π, l'addition, *etc*.

4.3.3 Le test d'égalité

Les fonctions calculables ou un système suffisamment riche de formules symboliques (incluant, en particulier, les opérateurs logiques et les opérations usuelles de l'analyse comme $\lim_x f(x)$, $\int f(x)dx$) permettent, plus ou moins, d'exprimer tous les nombres réels qui apparaissent dans les problèmes concrets que l'on rencontre.

5. Ces notions sont toutefois informelles. En réalité, la formalisation correcte de la notion de réel définissable est un exercice délicat qui ouvre la porte à des questions subtiles; le lecteur pourra consulter `https://mathoverflow.net/questions/44102/is-the-analysis-as-taught-in-universities-in-fact-the-analysis-of-definable-numb/44129#44129` à ce propos.

Cependant, savoir exprimer un certain ensemble de nombres n'est pas synonyme de savoir le manipuler efficacement. Dans le cas des réels, un problème critique est le test d'égalité.

Problème 4.3.4 (Test d'égalité). *Étant donnés deux nombres réels, décider si $a = b$ ou, de manière équivalente, si $a - b = 0$.*

Dans l'idée de comparer des nombres réels, une idée naturelle est de comparer leurs approximations. Considérons par exemple le nombre $a = 8\int_0^\infty \cos(2x)\prod_{n=1}^\infty \cos\left(\frac{x}{n}\right) dx$. Voici une approximation numérique de a avec 15 décimales obtenue grâce à la bibliothèque `mpmath` de SageMath [6].

```
sage: from mpmath import mp
sage: print(8 * mp.quadosc(lambda x: mp.cos(2*x) * mp.nprod(
...       lambda n: mp.cos(x/n), [1,mp.inf]),
...           [0,mp.inf], omega=1))
3.14159265358979
```

Le résultat obtenu ressemble étrangement à π et, en effet, les 15 premières décimales de π sont exactement les mêmes que celles de a :

```
sage: print(mp.pi)
3.14159265358979
```

Il s'agissait toutefois d'un piège ! En fait, le nombre a n'est *pas* égal à π, mais l'est seulement à 10^{-41} près [21]. Cet exemple illustre que, de manière générale, il n'est pas possible de démontrer l'égalité de deux nombres réels en se contentant de comparer des approximations numériques. Certes, ceci est bien évident et ne surprendra aucunement notre lectrice aguerrie mais il est, malgré tout, extrêmement important de garder cette conclusion bien présente à l'esprit.

En contrepartie, il est toujours possible de démontrer que deux nombres calculables ne sont *pas* égaux en en calculant des approximations suffisamment fines. Typiquement, pour l'exemple précédent, on se serait rendu compte de la différence si on avait fait l'effort de poursuivre le calcul jusqu'à 50 chiffres après la virgule. Autrement dit, l'*inégalité* entre nombres réels calculables est un problème semi-décidable : l'algorithme qui consiste à comparer des approximations de plus en plus précises s'arrête lorsque les nombres réels sont différents mais boucle indéfiniment dans le cas contraire. Cependant, même dans le cas favorable de nombres réels

6. À cause des oscillations de la fonction dont on calcule l'intégrale, il est nécessaire d'utiliser la fonction spéciale `quadosc` pour avoir une valeur précise de a. Le calcul dans cet exemple particulier prend énormément de temps !

différents, il n'est pas possible de prédire *a priori* s'il sera nécessaire, pour conclure, de calculer 50 ou $10^{10^{50}}$ décimales.

A contrario, décider l'égalité de nombres entiers est un problème facile, lié au fait qu'il existe une représentation canonique des entiers, à savoir l'écriture en base 10 (ou 2, ou 2^{64}). En outre, les algorithmes classiques d'addition et de multiplication respectent cette représentation canonique. Ceci permet littéralement de *démontrer* par le calcul que $2 \times 6 = 3 \times 4 = 12$, par exemple. De même, les nombres algébriques admettent des représentations canoniques ; on en déduit que le problème 4.3.4 est effectif sur $\overline{\mathbb{Q}}$.

Pour les nombres calculables ou les expressions symboliques, il n'existe généralement pas de forme canonique qui permette de tester l'égalité. Comparer les valeurs de deux nombres calculables représentés par deux programmes se heurte rapidement au problème de l'arrêt de TURING. Pour les expressions symboliques, on peut certes implémenter des règles de simplification qui permettent de conclure dans *certains* cas (par exemple $\sqrt{2}/2 - 1/\sqrt{2} = 0$ ou $\sin(\pi) = 0$) ; cependant, il est illusoire de penser pouvoir résoudre le problème 4.3.4 en général, étant donné que l'expressivité du calcul symbolique est telle qu'elle permet *grosso modo* de coder n'importe quel énoncé mathématique.

Exemple 4.3.5. *L'hypothèse de* RIEMANN *est équivalente à l'égalité suivante :*

$$\frac{1}{\pi}\int_0^\infty \log\left(\left|\frac{\zeta\left(\frac{1}{2}+it\right)}{\zeta\left(\frac{1}{2}\right)}\right|\right)\frac{1}{t^2}\,dt \stackrel{?}{=} \frac{\pi}{8}+\frac{\gamma}{4}+\frac{\log(8\pi)}{4}-2 \qquad (4.1)$$

où $\zeta(s)$ est la fonction de RIEMANN *et $\gamma = \lim_{n\to\infty}\left[\left(\sum_{k=1}^{n}\frac{1}{k}\right)-\log(n)\right]$ est la constante d'*EULER [7].

Une démonstration (ou une infirmation) de l'égalité (4.1) entre nombres réels est donc équivalente à l'un des problèmes du millénaire, littéralement une question à un million de dollars.

Conséquences sur les fonctions calculables

Les problèmes de décidabilité que nous venons de discuter se posent plus généralement lorsque l'on cherche à construire des représentations paresseuses ou symboliques de nouveaux nombres réels. Par exemple :

— Étant donnée une description symbolique ou algorithmique d'une suite a_n, comment décider si la limite $x = \lim_{n\to\infty} a_n$ existe ? (En

7. `https://mathoverflow.net/q/279936`. Exercice : vérifier l'égalité (4.1) numériquement à l'aide de `mpmath`. (Indication : on pourra découper l'intégrale entre les zéros successifs de la fonction zêta de RIEMANN.)

général, cette question est indécidable, comme on le démontre en la réduisant au problème de l'arrêt.)

— Étant donné un nombre réel x, il est nécessaire de savoir si $x \neq 0$ avant de pouvoir dire que $1/x$ représente un nombre réel. Pareillement, étant donnée une fonction continue par morceaux comme

$$f(x) = \begin{cases} 1 & x \geqslant 0 \\ 0 & x < 0 \end{cases},$$

calculer $f(x)$ où x est donné par un programme demande de décider préalablement si x est positif ou négatif. Or, ceci est à nouveau indécidable étant donné que des approximations arbitrairement proches de x peuvent être de signe variable lorsque $x = 0$.

Le dernier exemple que nous avons donné met en lumière le principe suivant :

Théorème 4.3.6. *Les fonctions calculables sont toutes continues.*

À la lumière du théorème 4.3.6, examinons quelques exemples de fonctions calculables et non calculables. Pour commencer, la solution d'un système non singulier d'équations linéaires est calculable dès lors que les coefficients du système le sont.

Théorème 4.3.7. *La matrice A^{-1} est calculable si $A \in M_{n,n}(\mathbb{C})$ est inversible et calculable.*

L'algorithme d'élimination de Gauss fournit une solution effective au problème du calcul de l'inverse d'une matrice. En effet, bien qu'il fasse *a priori* intervenir des tests d'égalité à zéro, on se convainc facilement que dans le cas d'une matrice de rang maximal, il existe toujours n pivots dont on peut démontrer la non-nullité à l'aide d'approximations suffisamment précises. Par contre, on notera que calculer le rang d'une matrice ne fait pas partie des problèmes calculables ; par exemple, étant donnée la matrice

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & \varepsilon & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

où ε est un nombre réel calculable qui représente 0, le mieux que l'on puisse décider est l'encadrement $1 \leqslant \operatorname{rang}(A) \leqslant 2$.

D'autres exemples de quantités calculables sont les racines de polynômes et les valeurs propres de matrices (on notera que ces deux quantités varient de façon continue avec les coefficients).

Théorème 4.3.8. *Si les coefficients d'un polynôme $f \in \mathbb{C}[x]$ sont calculables et que le coefficient dominant de f ne s'annule pas, alors il existe un entier p_0 tel que pour tout $p \geqslant p_0$, on puisse calculer une liste de disques deux à deux disjoints de rayon 2^{-p} ayant la propriété suivante : toute racine de f appartient à l'un de ces disques et, réciproquement, chacun de ces disques contient au moins une racine de f.*

La multiplicité d'une racine (ou, de même, la multiplicité d'une valeur propre) n'est pas une fonction calculable en général. Informellement, ceci est dû au fait que la détermination d'une multiplicité requiert des tests d'égalité. De manière plus formelle, on pourra remarquer que le nombre de racines distinctes de $f(x) = x(x + \varepsilon)$ est une fonction discontinue de ε, et conclure par le théorème 4.3.6. Le mieux que l'on puisse faire dans le cas de racines multiples est de démontrer (à l'aide du théorème de Rouché) qu'un disque suffisamment petit contient exactement m racines *comptées avec multiplicité*. De même, il est impossible de décider si un polynôme admet des racines réelles de multiplicité $m > 1$ puisque, à nouveau, le nombre de racines réelles de $f(x) = x^2 + \varepsilon$ est une fonction discontinue de ε.

Il existe de nombreux algorithmes classiques de calcul de racines ou de valeurs propres, ceux-ci venant toutefois rarement avec une preuve de correction. En pratique, les meilleures méthodes rigoureuses consistent généralement à calculer des racines approchées à l'aide d'heuristiques numériques puis de valider *a posteriori* le résultat du calcul par des méthodes spécifiques.

4.3.4 Décidabilité pour certains ensembles de nombres

Dans certaines circonstances particulières, on peut démontrer l'égalité de deux expressions par calcul direct. C'est le cas, par exemple, lorsque l'on se restreint aux nombres algébriques. En effet, étant donné un polynôme $f \in \mathbb{Z}[x]$, on peut écrire des bornes explicites (qui s'expriment en fonction du degré et de la taille des coefficients de f) sur la distance entre deux racines quelconques de f. Ainsi, étant donnés deux nombres algébriques α et β, il est possible de calculer *a priori* un nombre $\varepsilon > 0$ explicite pour lequel l'inégalité $|\alpha - \beta| < \varepsilon$ implique $\alpha = \beta$.

Au delà de $\overline{\mathbb{Q}}$, il est difficile d'exhiber d'autres sous-ensembles de $\mathbb{R}$ pour lesquels le test d'égalité est décidable ; au fil des années, il n'y a eu que peu de succès notables dans cette direction. Un cas prometteur, malgré tout, est celui des *nombres élémentaires*, définis comme les nombres qui peuvent être représentés par des expressions symboliques ne faisant intervenir que des nombres algébriques, des fonctions algébriques et des

fonctions élémentaires (exp, log, les fonctions trigonométriques et leurs inverses).

Théorème 4.3.9 (RICHARDSON et FITCH)**.** *L'égalité entre nombres élémentaires est décidable si la conjecture de* SCHANUEL *est vraie [22].*

La conjecture de SCHANUEL est un énoncé de transcendance qui prédit *grosso modo* qu'il n'existe pas de relations algébriques inattendues entre nombres élémentaires ; par exemple, elle implique que $\pi + e$ doit être transcendant. La conjecture de SCHANUEL est considérée par la communauté comme un problème difficile. Toutefois, RICHARDSON et FITCH ont réussi à écrire un semi-algorithme décidant l'égalité de deux nombres élémentaires ; celui-ci fonctionne systématiquement, à moins qu'il ne tombe, durant son exécution, sur un contre-exemple à la conjecture de SCHANUEL, auquel cas le programme boucle indéfiniment.

En lien étroit avec le théorème de RICHARDSON et FITCH, on notera que le problème de décision suivant concernant les fonctions élémentaires n'est pas décidable.

Théorème 4.3.10 (RICHARDSON)**.** *Il n'existe pas d'algorithme qui décide, en général, si une fonction $f(x)$ représentée par une formule symbolique ne faisant intervenir que des nombres rationnels, π, $\log(2)$, e^x et $\sin(x)$ et $|x|$ s'annule partout.*

L'un des candidats les plus prometteurs à être une extension transcendante effective de $\overline{\mathbb{Q}}$ est l'anneau des périodes [30]. Par définition, une *période* est un nombre complexe qui peut s'écrire sous la forme d'une intégrale $\int_A f$ où f est une fonction algébrique en n variables et A est un sous-ensemble de $\mathbb{R}^n$ défini par des inégalités algébriques (dans lesquelles toutes les constantes qui apparaissent sont aussi des nombres algébriques). Par exemple, $\pi = 4\int_0^1 \sqrt{1-x^2}dx$ et $\log(2) = \int_1^2 x^{-1}dx$ sont des périodes.

Il a été démontré que les périodes sont calculables dans le sens de la définition 4.3.1 [23]. La question de la décidabilité du test d'égalité reste toutefois un problème ouvert :

Conjecture 4.3.11 (KONTSEVICH-ZAGIER)**.** *L'égalité entre périodes est décidable. Concrètement, étant données deux intégrales $\int_A f$ et $\int_B g$ représentant la même période, il existe un procédé algorithmique qui permet de transformer la première en la seconde en lui appliquant une suite finie de transformations simples (changement de variables, formule de* STOKES*).*

4.4 Nombres réels approchés

Dans cette partie, nous abordons les principes de base du calcul numérique fiable, voire prouvé, sur machine. Nous discuterons également du coût des algorithmes mis en jeu.

L'idée fondamentale qui sous-tend tout le calcul numérique consiste à remplacer un nombre réel x (possiblement très difficile à décrire) par une approximation $\widehat{x} = x + \varepsilon$ qui est un nombre rationnel facile à manipuler. L'erreur ε, quant à elle, reste inconnue mais on saura généralement la borner ou, du moins, l'estimer. Il est souvent commode de séparer les sources d'erreurs en deux catégories :

— les erreurs d'arrondis, conséquence des opérations arithmétiques sous-jacentes qui s'effectuent à précision finie ;

— les erreurs de troncation ou de discrétisation qui apparaissent par exemple lorsque l'on remplace une somme infinie $\sum_{n=0}^{\infty} a_n$ par l'une de ses sommes partielles $\sum_{n=0}^{N} a_n$, ou lorsque l'on remplace la solution d'une équation différentielle par une approximation calculée par une méthode de discrétisation avec un pas $h > 0$.

L'étude de la manière dont les erreurs apparaissent et se propagent au fil des calculs est, bien entendu, un sujet de première importance mais nous ne l'évoquerons que brièvement dans la suite de ce cours. En particulier, nous n'étudierons pas les concepts de stabilité directe et stabilité inverse, ni la théorie du conditionnement, ni les détails des analyses de précision dans le cadre de l'arithmétique à virgule flottante. Le lecteur intéressé pourra trouver des discussions approfondies sur ces sujets dans n'importe quel ouvrage d'analyse numérique.

4.4.1 L'arithmétique à virgule flottante

Pour des raisons d'efficacité, la plupart des implémentations disponibles utilisent des approximations de la forme $\widehat{x} = a \cdot 2^b$ avec $a, b \in \mathbb{Z}$. Un nombre de cette forme est appelé un *nombre à virgule flottante binaire* (ou un *nombre dyadique*). Les entiers a et b sont respectivement appelés la *mantisse* et l'*exposant* de $\widehat{x}$. Si on se restreint aux a tels que $|a| < 2^p$, alors $\widehat{x}$ est appelé un *nombre à virgule flottante sur p bits* [8]. Pour certaines applications dans lesquelles les quantités considérées sont toutes du même ordre de grandeur,

8. Souvent, $\widehat{x}$ est plutôt choisi dans $\pm[0{,}5; 1[$, ou dans $[0{,}5; 1[$ et un bit supplémentaire est utilisé pour encoder le signe. Dans ce texte, nous omettrons volontairement ce type de considérations de même que les valeurs particulières *NaN* (Not a Number) et ∞ et les problèmes de dépassement de capacité (*overflow* et *underflow*).

l'*arithmétique à virgule fixe* (correspondant au cas où b est fixé, typiquement égal à $-p/2$) est parfois préférée.

Effectuer des opérations sur les nombres à virgule flottante implique généralement de faire des arrondis afin de préserver les conditions $a, b \in \mathbb{Z}$ et $|a| < 2^{-p}$. La règle d'or consiste à d'imposer que chaque arrondi n'introduise qu'une erreur relative $|x - \widehat{x}|/|x|$ de l'ordre de 2^{-p}. Pour un calcul impliquant de nombreuses opérations élémentaires, la précision p doit être ajustée en fonction de la précision finale souhaitée, en tenant compte des pertes de précision s'accumulant à chaque étape du calcul.

Les types *binary32* et *binary64* définis dans le standard IEEE 754 (qui correspondent respectivement à $p = 24$ et $p = 53$) sont aujourd'hui implémentés dans la majorité des processeurs et, dans la pratique, sont pratiquement devenus synonymes d'arithmétique à virgule flottante. Le calcul à plus haute précision, quant à lui, doit être implémenté au niveau logiciel. Les implémentations les plus répandues sont la *quadruple précision* ($p = 106$), émulée par des paires de *binary64*, et la *précision arbitraire* (où p n'est limité que par la mémoire disponible). L'inconvénient principal de l'arithmétique en précision arbitraire est sa lenteur ; en comparaison de l'arithmétique flottante qui existe nativement dans les processeurs, elle peut aller entre 10 et 1000 fois moins vite.

SageMath dispose de plusieurs fonctionnalités pour travailler avec des réels en précision arbitraire : le constructeur `RealField` (qui s'appuie sur la libraire MPFR), la libraire `mpmath` et le logiciel `Pari/GP`. Voici un exemple simple avec `mpmath` calculant $\pi = 2\int_{-1}^{1} \sqrt{1-x^2}dx$ (la ligne `mp.dps = 100` fixe la précision de calcul à 100 chiffres décimaux, ce qui correspond à $p = 336$ bits) :

```
sage: from mpmath import mp
sage: mp.dps = 100
sage: print(2 * mp.quad(lambda x: mp.sqrt(1-x**2), [-1,1]))
3.14159265358979323846264338327950288419716939937510
58209749445923078164062862089986280348253421170679
```

Faisons une remarque à propos de la pertinence d'un calcul à si haute précision. En réalité, les applications mathématiques pour lesquelles on a besoin d'une précision $p > 10^3$ ne sont pas anecdotiques. On peut citer, par exemple :

— le calcul des solutions en temps long de systèmes chaotiques,
— le calcul de termes éloignés d'une suite récurrente,
— l'évaluation de séries entières à signes alternés près du bord du disque de convergence,

— le calcul de n'importe quelle petite quantité qui, pour des raisons algorithmiques, doit être écrite comme la différence de deux grandes quantités,
— la démonstration d'inégalités entre deux nombres très proches,
— la reconnaissance (heuristique ou prouvée) de valeurs discrètes ou de formules exactes à partir de données numériques approchées.

Au delà des erreurs d'arrondis et de troncature, une troisième source d'erreurs en calcul scientifique est l'incertitude sur les entrées lorsque celles-ci proviennent de mesures physiques ou d'expériences statistiques. En réalité, il y a assez peu de contextes en physique ou en ingénierie où cela a du sens de donner un résultat avec plus de 7 chiffres significatifs (ou, au pire, disons 16 chiffres). Malgré cela, il y a beaucoup de situations où il est important de mener les calculs intermédiaires avec plus de précision en raison des erreurs numériques qui peuvent apparaître puis être amplifiées au gré des algorithmes utilisés. En général, au plus les données sont nombreuses et les calculs sont longs, au plus la précision numérique doit être augmentée [9].

4.4.2 Propagation des erreurs et arithmétique d'intervalles

On peut majorer (ou estimer) l'erreur totale commise lors d'un calcul complexe en majorant (ou en estimant) les erreurs engendrées par chaque opération élémentaire et en calculant des bornes (ou des estimations) sur la propagation de celles-ci lors des opérations futures. Au delà des algorithmes excessivement simples, ce travail est souvent fastidieux à faire à la main. Une solution alternative consiste à propager les erreurs automatiquement en utilisant l'arithmétique d'intervalles [14].

Le principe à la base de toute l'arithmétique d'intervalles est d'approcher un nombre réel x par un sous-ensemble X de $\mathbb{R}$ contenant x. Un tel ensemble X est parfois appelé une *inclusion*. Il doit être aisément représentable sur machine.

Définition 4.4.1. *Soif $f : A \to B$ une fonction. Une* fonction d'inclusion *de f est une fonction $F : \mathcal{P}(A) \to \mathcal{P}(B)$ qui envoie un sous-ensemble de A sur un sous-ensemble de B de façon à ce que $f(x) \in F(X)$ pour tout sous-ensemble X de*

9. Il y a cependant des exceptions notables à ce principe : par exemple, il a été observé que certains réseaux de neurones fonctionnent déjà bien avec un encodage de $\widehat{x}$ sur 16 bits (ce qui correspond peu ou prou à $p = 10$ ou $p = 8$), voire sur 8, 4, 2 et même 1 seul bit. La recherche est actuellement active sur les modèles à précision mixte (pas seulement dans le cadre des réseaux de neurones) où l'essentiel des calculs est conduit à faible précision, avec une augmentation possible ponctuelle de la précision pour certaines parties critiques [4].

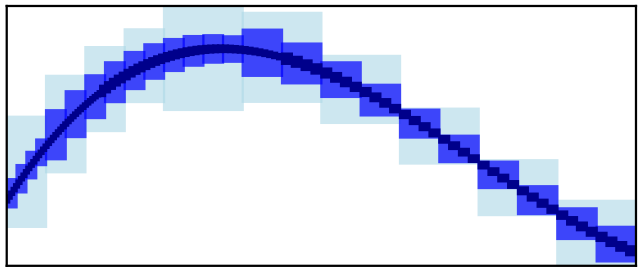

FIGURE 4.2 – *Illustration de la propriété de préservation de continuité d'une fonction d'inclusion d'une fonction à variable réelle : les intervalles $F(X)$ diminuent de taille avec X, tout en restant concentrés autour du graphe de la fonction $y = f(x)$. Dans cet exemple, on observe que les inclusions $F(X)$ ne sont pas les plus fines possibles.*

A et tout élément $x \in X$. En d'autres termes, on doit avoir $f(X) \subset F(X)$ pour tout $X \subset A$.

La règle de la définition précédente est parfois appelée le *principe d'inclusion*. Avant de poursuivre, notons que ce principe s'applique à des objets très généraux, et pas uniquement aux nombres réels : dans la définition 4.4.1, il n'est pas nécessaire de supposer que A et B sont des sous-ensembles de $\mathbb{R}$. Il est évident, par ailleurs, que les fonctions d'inclusion peuvent être composées dès lors que les fonctions sous-jacentes sont, elles-mêmes, composables. Notons encore qu'il n'est pas requis que les inclusions soient fines ; par exemple, la meilleure inclusion possible pour la fonction $\sin(x)$ sur $X = [0, \pi]$ est $[0, 1]$ mais une fonction d'inclusion qui retournerait $[-2, 2]$ serait également tout à fait acceptable. En pratique, il y aura souvent un compromis à trouver entre la qualité de la fonction d'inclusion et le coût calculatoire.

Une propriété raisonnable que l'on impose généralement sur les fonctions d'inclusion est la préservation de la continuité : si f est continue en un point $x \in \mathbb{R}$, on demande à ce que pour toute suite $(X_n)_{n \geqslant 0}$ d'intervalles contenant x telle que $\text{taille}(X_n) \to 0$ quand $n \to \infty$, on ait aussi $\text{taille}(F(X_n)) \to 0$ [10] (voir figure 4.2 pour une illustration de ce principe). Cette propriété résonne comme un analogue de la définition 4.3.2 dans le contexte de l'arithmétique d'intervalles. Elle est en outre souvent décisive pour démontrer la convergence d'algorithmes.

Il existe plusieurs possibilités pour représenter des nombres réels

10. Si l'arithmétique en virgule flottante est utilisée pour représenter les intervalles, cet axiome nécessite que la précision utilisée augmente rapidement avec n.

par des intervalles. L'une d'elles consiste à utiliser des intervalles $[a, b]$ où les extrémités a et b sont des nombres à virgule flottante et de représenter ceux-ci par le couple (a, b). C'est ainsi qu'est implémenté le corps `RealIntervalField` en SageMath, qui repose sur la bibliothèque MPFI. Une autre possibilité est d'utiliser des intervalles de la forme $[m \pm r] = [m - r, m + r]$ (appelés parfois des *boules arithmétiques*). Cette implémentation est disponible en SageMath *via* le constructeur `RealBallField` qui fait appel au logiciel Arb en coulisse. De manière générale, la première représentation paraît plus naturelle pour représenter des *subdivisions de l'espace* alors que les boules sont plus efficaces pour représenter des *nombres réels isolés*. Ceci dit, dans de nombreux cas, ces représentations sont simplement interchangeables.

Tester l'égalité de nombres réels représentés par des intervalles n'a, en général, pas de sens, hormis dans le cas très particulier où les intervalles en question sont tous réduits à un point. Il est cependant possible de tester si les intervalles sont disjoints (auquel cas, on peut conclure à l'inégalité des nombres réels correspondants) ou s'ils se chevauchent (auquel cas, l'égalité est possible). À noter que le résultat de la soustraction de deux intervalles qui représentent le même nombre réel est un intervalle contenant zéro :

```
sage: R = RealBallField(53)
sage: x = R(2); sqrt(x)/2 - 1/sqrt(x)
[+/- 4.45e-16]
sage: R = RealBallField(333)
sage: x = R(2); sqrt(x)/2 - 1/sqrt(x)
[+/- 1.72e-100]
```

Il est facile d'émuler les réels et les fonctions calculables (voir §4.3.1) par de l'arithmétique d'intervalles à précision arbitraire, simplement en écrivant une boucle qui effectue des évaluations de plus en plus précises jusqu'à atteindre la précision demandée :

```
sage: def f(p):
....:   prec = 10
....:   while True:
....:     R = RealBallField(prec)
....:     x = (R(163).sqrt()*R.pi()).exp()-640320**3-744
....:     print("prec = %s gave %s" % (prec, x))
....:     if x.accuracy() > p:   # relative accuracy
....:         break
....:     prec *= 2

sage: f(30)
prec = 10 gave [+/- 4.99e+16]
```

```
prec = 20 gave [+/- 6.83e+13]
prec = 40 gave [+/- 4.71e+7]
prec = 80 gave [+/- 4.76e-5]
prec = 160 gave [-7.499274028018143e-13 +/- 5.65e-29]
```

Remarquons qu'un tel programme peut boucler indéfiniment lorsque la fonction que l'on cherche a évaluer présente une discontinuité ou lorsque l'on souhaite atteindre une précision *relative* donnée pour un résultat exact égal à 0.

4.4.3 Sur la dépendance des erreurs

L'arithmétique d'intervalles assure un suivi rigoureux des erreurs mais, en contrepartie, ne le fait pas toujours de façon optimale. Pour illustrer cette affirmation, considérons le problème de calculer une somme $S_N = \sum_{k=1}^{N} x_k$, connaissant des approximations $\widehat{x}_k$ de x_k avec $\varepsilon_k = \widehat{x}_k - x_k \leqslant 2^{-p}$. Dans ces conditions, quelle est l'erreur maximale $|S_N - \sum_{k=1}^{N} \widehat{x}_k|$?

Dans le *pire cas*, la meilleure borne que l'on puisse donner est $N \cdot 2^{-p}$. Et, de fait, si nous effectuons le calcul dans le modèle de l'arithmétique d'intervalles, nous obtiendrons précisément cette borne (plus, éventuellement, des termes supplémentaires provenant d'erreurs d'arrondis si la somme des $\widehat{x}_k$ est elle-même calculée par de l'arithmétique approchée).

Toutefois, la borne du pire cas est parfois trop pessimiste. Typiquement, dans le cas où les erreurs ε_k sont réparties uniformément et indépendamment dans un intervalle centré en 0, le théorème de la limite centrale (ou, au choix, la théorie des marches aléatoires) nous apprend que l'erreur *moyenne* attendue est de l'ordre de $O(\sqrt{N}) \cdot 2^{-p}$. Ce type d'heuristique fournit des estimés qui sont souvent plus réalistes qu'une analyse dans le pire cas. Au contraire, l'arithmétique d'intervalles ne prend aucunement en compte l'indépendance des erreurs et calcule systématiquement l'estimation la plus pessimiste du pire cas [11].

Un cas extrême est celui où les erreurs sont entièrement corrélées et s'annulent. Par exemple, si $N = 2$ et $\varepsilon_1 = -\varepsilon_2$, on a $x_1 + x_2 = \widehat{x}_1 + \widehat{x}_2$ *exactement* mais l'arithmétique d'intervalles continue d'afficher la borne d'erreur pessimiste $2 \cdot 2^{-p}$. Quand ce phénomène se produit de façon répétée dans une boucle, on peut être amené à observer des bornes d'erreur qui grandissent *exponentiellement* vite. L'exemple le plus simple de ce phénomène apparaît lorsque l'on soustrait une quantité à elle même plusieurs fois d'affilée :

11. Exercice : tester quelques calculs avec, d'une part, l'arithmétique en virgule flottante et, d'autre part, l'arithmétique d'intervalles et observer la différence en pratique.

```
sage: x = RealBallField(53)(2).sqrt()
sage: x = x-x; print(x)
[+/- 4.45e-16]
sage: x = x-x; print(x)
[+/- 8.89e-16]
sage: x = x-x; print(x)
[+/- 1.78e-15]
...
[+/- 2.10e+6]
sage: x = x-x; print(x)
[+/- 4.20e+6]
```

Cette explosion exponentielle signifie, réciproquement, qu'il est nécessaire d'augmenter la précision initiale de manière exponentielle avec le nombre d'itérations si l'on désire atteindre une précision finale fixée à l'avance. Le calcul à haute précision est parfois inévitable, par exemple lorsque l'on itère des systèmes dynamiques chaotiques très sensibles aux conditions initiales, mais il peut aussi ne refléter aucune réalité intrinsèque. Ainsi, lorsqu'on est amené à utiliser l'arithmétique d'intervalles, il est souvent souhaitable de réécrire les formules évaluées et de reconcevoir les algorithmes utilisés afin de minimiser autant que possible les dépendances entre les erreurs.

Une situation classique où les problèmes de dépendance acquièrent une importance considérable est la résolution de systèmes linéaires. En effet, lorsque l'on exécute l'algorithme d'élimination de GAUSS en arithmétique d'intervalles, on observe généralement des bornes d'erreur qui grandissent exponentiellement vite avec la taille n de la matrice d'entrée ; on a ainsi besoin d'une précision d'au moins $O(n)$ chiffres significatifs. Le calcul à haute précision est, de fait, inévitable pour les matrices mal conditionnées. Par contre, pour les matrices bien conditionnées, l'arithmétique d'intervalles conduit à la même explosion de complexité alors que le même algorithme exécuté en arithmétique en virgule flottante est parfaitement stable.

À cause de cela, la meilleure façon de résoudre un système linéaire en arithmétique d'intervalles est de commencer par calculer une solution approchée en arithmétique à virgule flottante puis, dans un second temps, d'utiliser des méthodes de certification *a posteriori* pour obtenir des bornes d'erreur rigoureuses [12].

12. Cette observation selon laquelle la propagation directe des bornes d'erreur a tendance à surestimer exponentiellement vite les erreurs réelles (aussi bien pour l'algorithme d'élimination de GAUSS que pour d'autres algorithmes) est considérée comme l'un des

4.4.4 Analyse de complexité en arithmétique approchée

L'archétype des algorithmes rapides impliquant des nombres réels est la transformée de FOURIER rapide (FFT) qui met en œuvre une stratégie de type diviser pour régner pour calculer la transformée de FOURIER discrète (DFT) d'un vecteur de nombres complexes

$$X_k = \sum_{j=0}^{n-1} x_j e^{-2\pi ikj/n}, \quad k = 0, 1, \ldots, n-1$$

en $O(n \log n)$ opérations au lieu de $O(n^2)$ pour un algorithme naïf. L'une des principales applications de la FFT est la multiplication rapide des polynômes. On peut multiplier des polynômes de degré strictement inférieur à n à coefficients réels ou complexes en seulement $O(n \log n)$ opérations (au lieu de $O(n^2)$ pour un algorithme naïf) comme ceci : on évalue les polynômes d'entrée en $2n$ racines de l'unité, on multiplie ces évaluations point par point et, enfin, on interpole pour retrouver ainsi les $2n$ coefficients du polynômes produit. Les étapes d'évaluation multi-point et d'interpolation sont simplement des DFT.

Toutefois, il n'est généralement pas suffisant de compter les opérations arithmétiques (complexité algébrique) pour analyser l'efficacité des algorithmes numériques. En effet, ce décompte sommaire ne tient pas compte de la précision utilisée pour représenter les nombres et il peut très bien arriver qu'un algorithme effectuant moins d'opérations ait besoin en contrepartie d'augmenter sensiblement la précision de calcul, induisant finalement un coût total supérieur. Pour cette raison, les algorithmes « rapides » ne sont pas toujours les meilleurs. Souvent, il est plus réaliste de se placer dans le modèle de complexité binaire où l'on ne compte pas les opérations arithmétiques mais les opérations sur les bits.

Clairement, on peut additionner et soustraire des nombres entiers ou des nombres à virgule flottante de p bits en $O(p)$ opérations binaires. La multiplication, quant à elle, a un coût quasi-linéaire :

Théorème 4.4.2 (HARVEY–van der HOEVEN)**.** *Il est possible de multiplier deux entiers de p bits en $O(p \log p)$ opérations binaires.*

De façon surprenante, ce résultat n'a été démontré qu'en 2019 [27] [13]. L'idée fondamentale derrière la multiplication rapide des entiers est l'obser-

phénomènes les plus importants de l'analyse numérique. Elle a été la principale découverte de J. H. WILKINSON et a conduit au développement de l'analyse d'erreur par stabilité inverse dans les années 1960. WILKINSON a reçu le prix TURING en 1970.

13. En fait, au moment où j'écris ces notes, l'article de HARVEY et van der HOEVEN n'a pas été complètement arbitré. Croisons les doigts !

vation que l'entier 325 peut être vu comme l'évaluation en $x = 10$ du polynôme $3x^2 + 2x + 5$. À partir de là, avec un peu de travail, il est possible de ramener de la multiplication des entiers à celle des polynômes et d'utiliser des méthodes de type FFT. Le premier algorithme de multiplication rapide basé sur la FFT a été publié en 1971 par SCHÖNHAGE et STRASSEN [14] ; sa complexité était de $O(p \log p \log\log p)$ opérations binaires [26]. Le facteur parasite $\log\log p$ provient de ce que les opérations internes à l'algorithme de FFT n'ont pas, au moins en apparence, un coût de $O(1)$: en effet, celles-ci reposent elles-mêmes sur la multiplication d'entiers [15] et la précision totale dans ce procédé récursif complexe croît avec p.

L'algorithme de HARVEY et van der HOEVEN utilise plusieurs techniques ingénieuses pour effacer le facteur $\log\log p$. Parmi elles, on peut citer le passage d'une FFT unidimensionnelle à une FFT multidimensionnelle et l'utilisation d'un ré-échantillonage approché dans le but d'ajuster la taille des vecteurs. Toutes les étapes de l'algorithme doivent être analysées avec précaution afin de prendre en compte simultanément les erreurs d'approximation, les pertes de précision et, bien sûr, la complexité binaire.

En pratique, la différence entre $O(p \log p \log\log p)$ et $O(p \log p)$ n'est visible que pour des p de taille astronomique ; dans les implémentations, une mauvaise constante dans le $O(\cdot)$ importe souvent plus. Dans la bibliothèque GMP *bignum*, la FFT (précisément, l'algorithme de SCHÖNHAGE-STRASSEN) n'est utilisé que pour des nombres de plus de 100 000 chiffres décimaux. Pour les entiers de taille plus petite, il est plus rapide d'utiliser l'algorithme naïf (jusqu'à, à peu près, 1000 chiffres) ou l'algorithme de KARATSUBA et ses généralisations (de 1000 chiffres à 100 000 chiffres) [19]. Les calculs impliquant des nombres avec plusieurs dizaines de milliers de chiffres ne sont pas monnaie courante, mais ils peuvent apparaître, de temps en temps, en théorie algorithmique des nombres. Il y a également des situations où il est rentable de remplacer un grand nombre d'opérations sur des petits entiers par un nombre restreint d'opérations sur des entiers gigantesques.

Certaines autres opérations algébriques sur les entiers ou les nombres à virgule flottante reposent sur la multiplication rapide des entiers [3].

Théorème 4.4.3. *Il est possible de calculer le quotient $\lfloor a/b \rfloor$ d'un entier a de $2p$ bit par un entier b de p bits, ou la racine carrée $\lfloor \sqrt{a} \rfloor$ d'un entier a de $2p$ bits, ou*

14. En réalité, SCHÖNHAGE et STRASSEN ont publié deux algorithmes, le premier utilisant des nombres complexes et le second n'utilisant que de l'arithmétique exacte sur les entiers. C'est cette deuxième version qui est appelée couramment l'algorithme de SCHÖNHAGE-STRASSEN et à laquelle nous faisons référence dans la suite du texte.

15. Ces entiers sont plus petits, ce qui permet une approche récursive.

encore une approximation à 2^{-p} près de quotients de racines carrées de nombres à virgule flottante, pour un coût de $O(p \log p)$ opérations binaires.

L'idée derrière ce théorème est de reformuler la question en un problème de recherche de point fixe d'une équation algébrique que l'on résout finalement par une méthode de NEWTON en prenant garde à ce que les formules itératives obtenues ne fassent intervenir que des additions, des soustractions et des multiplications. Par exemple, pour calculer $1/b$, on peut résoudre l'équation $x - 1/b = 0$, ce qui conduit *in fine* à l'itération $x_{k+1} = 2x_k - bx_k^2$. À chaque étape, le nombre de décimales correctes est, plus ou moins, doublé. Ainsi, l'algorithme ne nécessite au total que $O(\log p)$ itérations pour une précision de p bits. Nous encourageons la lectrice à écrire proprement la démonstration du théorème 4.4.3 et, en particulier, à se convaincre que, malgré les $O(\log p)$ itérations du schéma de NEWTON, la complexité totale n'augmente pas jusqu'à $O(p \log^2 p)$.

Théorème 4.4.4. *Étant donné un nombre complexe en virgule flottante z, il est possible de calculer une approximation rationnelle de e^z et de* $\log(z)$ *(pour la détermination principale du* log*) à 2^{-p} près pour un coût de $O(p \log^2 p)$ opérations binaires.*

Ce résultat de complexité s'étend plus généralement à l'évaluation des fonctions élémentaires (c'est-à-dire des fonctions s'écrivant comme une composée d'opérations arithmétiques, d'exponentielles, de logarithmes, de fonctions trigonométriques et de leurs inverses), en dehors des points singuliers du domaine de définition des fonctions mises en jeu [16].

L'algorithme sous-jacent au théorème 4.4.4 s'appuie sur l'*itération de la moyenne arithmético-géométrique* :

$$a_{k+1}, b_{k+1} = \frac{a_k + b_k}{2}, \sqrt{a_k b_k}. \tag{4.2}$$

Pour n'importe quelles valeurs initiales a_0 et b_0 strictement positives, on peut démontrer que les suites (a_k) et (b_k) définies ci-dessus convergent vers une limite commune $a_\infty = b_\infty$ qui, pour a_0 et b_0 bien choisis, est reliée à $\log(z)$. À l'instar de la méthode de NEWTON, la convergence est quadratique, dans le sens où le nombre de chiffres corrects double à chaque itération ; ainsi, pour obtenir une précision de p bits, $O(\log p)$ itérations suffisent et la complexité annoncée en découle. Il est à noter que cette méthode, qui permet d'évaluer les fonctions élémentaires pour un coût de

16. Il est important de se rendre compte, malgré tout, que ce type de bornes de complexité n'est pas uniforme vis-à-vis de z. À cet effet, le lecteur intéressé pourra chercher à estimer comment le coût du calcul de e^z ou de $\tan(z)$ varie en fonction de z.

$O(p \log^2 p)$ opérations binaires, passe par des évaluations de fonctions non élémentaires (des intégrales elliptiques) qui sont elles-mêmes calculées par une variante de (4.2) adaptée aux nombres complexes.

De manière plus pédestre, on sait atteindre une complexité à peine moins bonne, à savoir $O(p \log^3 p)$, en utilisant uniquement des équations fonctionnelles classiques (du type $e^{x+y} = e^x e^y$) et des évaluations de séries de TAYLOR tronquées. Ces dernières doivent toutefois être réalisées avec précaution par un algorithme de type diviser pour régner appelé le *scindage binaire*.

L'idée du scindage binaire peut être illustrée simplement par le calcul de la factorielle : $N! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot N$. En effet, remarquons qu'un calcul itératif naïf a un coût de $N^{2+o(1)}$ opérations binaires, alors qu'une méthode de type diviser pour régner conduit à une complexité moindre, de l'ordre de $N^{1+o(1)}$. Cette technique s'étend au calcul de produits de matrices $M_N M_{N-1} \cdots M_0$ lorsque les coefficients des M_i sont des petits nombres rationnels. Or, il se trouve que la série de TAYLOR de e^x peut être décrite à l'aide de tels produits matriciels. Plus généralement, cette même méthode fonctionne pour l'évaluation des fonctions D-finies (c'est-à-dire des fonctions solutions d'une équation différentielle linéaire ordinaires à coefficients polynomiaux), ce qui inclut en particulier la fonction d'erreur $\mathrm{erf}(x)$ et les fonctions de BESSEL.

4.5 Dérivation et intégration

Dans la dernière partie de ce chapitre, nous nous proposons d'évoquer quelques autres questions sur les fonctions réelles et complexes, allant au-delà de la simple évaluation en un point donné. Plus précisément, nous nous focalisons sur deux opérations fondamentales de l'analyse : la dérivation et l'intégration.

La difficulté de chacun de ces problèmes dépend de la manière dont les fonctions sont représentées ainsi que de la forme du résultat attendue. Nous allons traiter trois cas : (1) les fonctions sont données par des expressions symboliques, (2) les fonctions sont données par des programmes en « boîte noire » et (3) les fonctions sont représentées par des approximations.

Par souci de simplicité, nous considèrerons uniquement le cas de fonctions d'une variable réelle ou complexe (en se tenant à l'écart de la boîte de Pandore qu'est la dérivation et l'intégration de fonctions à plusieurs variables).

4.5.1 Calcul symbolique

Supposons que nous soit donnée une fonction représentée par une expression symbolique (du type $f(x) = e^{x^2}/x$) qui puisse être évaluée pour une valeur numérique donnée de x en parcourant l'arbre qui encode l'expression symbolique et en appliquant successivement les opérations rencontrées (ici $x^2, e^x, /$).

Il est facile d'écrire une expression symbolique pour la dérivée (ici $f'(x) = e^{x^2}(2x^2 - 1)/x^2$) en utilisant les règles de dérivation de façon répétée. Cependant, l'expression que l'on obtient ainsi pour $f'(x)$ peut croître rapidement en taille. Pour évaluer $f'(x)$ en un point x donné, il est ainsi souvent plus efficace de parcourir l'arbre de f et d'évaluer simultanément les valeurs de $f(x)$ et $f'(x)$ en appliquant les opérations rencontrées à des séries tronquées de la forme $a + b\varepsilon + O(\varepsilon^2)$; ce faisant, le coefficient b coïncide avec la valeur de la dérivée. Cette méthode est connue sous le nom de *différentiation automatique* (AD) et peut être facilement généralisée au cas des dérivées supérieures.

Le calcul de primitives est nettement plus compliqué. En effet, sauf dans des cas très particuliers où l'on peut utiliser des « astuces » comme des intégrations par parties ou des changements de variables, il n'existe malheureusement pas de formules générales d'intégration. Typiquement, la fonction e^{x^2} n'admet pas de primitive qui s'exprime à l'aide de fonctions élémentaires.

Plus précisément, les problèmes que l'on peut résoudre sous forme symbolique dépendent fortement des symboles que l'on s'autorise : si on accepte la fonction $\operatorname{erf}(x)$, alors on peut représenter une primitive de e^{x^2}. Pareillement, si on introduit un symbole pour la fonction $\Gamma(x)$, il n'existe pas de forme close pour la fonction $\Gamma'(x)$ (sauf à introduire encore un nouveau symbole).

Intégration indéfinie

Il existe une méthode symbolique systématique pour calculer des intégrales indéfinies, connue sous le nom d'algorithme de RISCH, qui décide si une fonction élémentaire donnée en entrée admet une primitive qui est, elle-même, élémentaire et, le cas échéant, la détermine. Il existe même des généralisations de l'algorithme de RISCH permettant de traiter certains types de fonctions non élémentaires comme $\operatorname{erf}(x)$. Tous ces algorithmes fonctionnent sur le principe suivant : le problème d'intégration est traduit en un problème algébrique exprimé dans la théorie des corps différentiels.

L'algorithme de RISCH n'est, en réalité, pas un algorithme à proprement parler car il nécessite un test d'égalité pour les expressions symboliques. À vrai dire, les constantes qui interviennent dans l'expression à intégrer sont déjà problématiques : la fonction $f(x) = x + (b-a)e^{x^2}$ possède une primitive élémentaire si et seulement si $a = b$. En outre, l'algorithme de RISCH est extrêmement complexe et n'a jamais été totalement implémenté [17]. De plus, il est coûteux et ne fournit pas nécessairement des résultats sous forme simplifiée. Pour ces raisons, les logiciels de calcul formel traditionnels essaient souvent des heuristiques basées sur la reconnaissance de motifs avant de se lancer dans l'algorithme de RISCH en cas d'échec.

Intégration définie

De manière peut-être surprenante, le calcul symbolique d'une intégrale définie $\int_a^b f(x)dx$ est assez différent du calcul de primitives. Au moins deux raisons à cela peuvent être invoquées :

— Une intégrale définie entre deux points particuliers a et b peut avoir une expression simple, quand bien même l'intégrale indéfinie correspondante n'en aurait pas. Par exemple, $\int_0^\infty e^{-zx^2}dx = \frac{1}{2}(\pi/z)^{1/2}$ pour $z > 0$. Pour les calculer, les logiciels de calcul formel utilisent des méthodes variées qui dépassent le cadre de ce cours.

— La formule $\int_a^b f(x)dx = F(b) - F(a)$ (où F est une primitive de f) s'applique uniquement lorsque f est continue sur $[a, b]$. En général, l'intégration symbolique nécessite donc de localiser les points singuliers de f, ce qui constitue un problème annexe délicat en lui-même. Ce problème est d'ailleurs à l'origine de bugs fréquents dans les systèmes de calcul formel où, parfois même, l'intégrale $\int_a^b f(x)dx$ d'une fonction à valeurs réelles pouvait renvoyer un résultat aberrant comme $1{,}23456 + 3{,}14159i$ à cause d'un mauvais choix de point de branchement. Prendre le temps de comparer le résultat d'une intégration symbolique avec celui d'une méthode numérique est souvent une bonne idée !

Lorsque les méthodes d'intégration indéfinies s'appliquent, elles ont souvent un avantage décisif sur un calcul numérique : elles sont beaucoup moins sensibles aux comportements, possiblement irréguliers, de la fonction à intégrer. Par exemple, évaluer l'intégrale $\int_0^1 \sin(Nx)dx$ avec $N = 1$

17. Actuellement, le logiciel FriCAS se réclame d'avoir l'implémentation la « plus complète » de l'algorithme de RISCH. Le statut exact de cette implémentation est discuté sur `http://fricas-wiki.math.uni.wroc.pl/RischImplementationStatus`.

ou $N = 10^{10}$ revient à peu près au même si l'on utilise une méthode symbolique, là où les algorithmes numériques peinent pour les grandes valeurs de N à cause des oscillations rapides.

4.5.2 Fonctions calculables en boîte noire

Étant donnée une implémentation en boîte noire de $f(x)$ — c'est-à-dire un programme qui évalue numériquement $f(x)$ étant donné x — il existe de nombreuses méthodes numériques pour calculer des approximations de la dérivée de f ou de son intégrale. Les plus simples d'entre elles reposent sur les approximations classiques $\int_a^b f(x)dx \approx h \cdot \sum_n f(a+nh)$ et $f'(x) \approx (f(x+h) - f(x))/h$ pour un certain $h > 0$. Ces approximations convergent vers les valeurs exactes attendues lorsque h tend vers 0 si f est, respectivement, intégrable au sens de RIEMANN ou dérivable.

Afin de majorer l'erreur commise dûe à la discrétisation (ou, de manière équivalente, de choisir convenablement le pas h étant donnée une précision souhaitée 2^{-p}), il est nécessaire de connaître des informations supplémentaires sur la régularité de f comme, typiquement, une borne sur ses dérivées supérieures. *A contrario*, une telle borne ne peut être uniquement déduite des valeurs prises par f en un nombre fini de points isolés : l'intégrale $\int_a^b f(x)dx$ peut varier de façon arbitrairement grande alors que la fonction f ne subit qu'une perturbation très localisée (par exemple, par ajout d'une fonction en escaliers ou d'une fonction infiniment dérivable du type Ne^{-Nx^2}).

De même, à une précision 2^{-p} donnée, la fonction $f(x)$ peut rester indistinguable d'une de ses perturbations dont la dérivée, quant à elle, explose ; par exemple, $f(x) + \varepsilon \sin(x/\varepsilon^2)$ ou $f(x) + \varepsilon H(x)$ où $H(x)$ est une fonction en escaliers avec, disons, $H'(0) = \infty$. Un exemple encore plus pathologique est donné par la fonction de WEIERSTRASS $f(x) = \sum_{n=0}^{\infty} 2^{-n} \cos(3^n \pi x)$ qui est continue et calculable, mais dérivable en aucun point ; évidemment, il n'y a aucun moyen de déduire cette dernière propriété à partir d'un nombre fini de valeurs prises par f.

Étant donnée une implémentation en boîte noire d'une fonction d'inclusion (voir définition 4.4.1) de $f(x)$, il est généralement trivial d'obtenir des bornes rigoureuses sur l'intégrale $\int_a^b f(x)dx$ en utilisant une méthode de subdivision comme illustré par la figure 4.2 (page 131). Cette méthode ne nécessite d'ailleurs pas la continuité de f, elle fonctionne pareillement, par exemple, pour des fonctions continues par morceaux. En particulier, $g(a,b) = \int_a^b f(x)dx$ peut être calculable pour tous a et b (dans le sens de la définition 4.3.2) sans que $f(x)$ soit lui-même calculable pour tout x.

La dérivation, quant à elle, est un problème intrinsèquement mal posé : il est impossible d'évaluer rigoureusement $f'(x)$ à partir d'une implémentation en boîte noire d'une fonction d'inclusion de f, quand même bien on disposerait d'informations additionnelles de régularité.

Le cas des fonctions holomorphes mérite cependant qu'on s'y attarde. En effet, grâce à la formule de CAUCHY (qui permet de ramener le calcul de dérivées à celui d'une intégrale de contour), une implémentation en boîte noire d'une fonction d'inclusion complexe d'une fonction holomorphe f est suffisante pour le calcul de dérivées et d'intégrales de f avec des bornes d'erreur rigoureuses. De surcroît, des algorithmes rapides sont disponibles dans ce cas : alors qu'une méthode sommatoire basée sur une subdivision d'intervalles requiert généralement un nombre exponentiel (en p) d'évaluations pour obtenir un résultat correct à 2^{-p} près, on dispose d'algorithmes spécifiques aux fonctions holomorphes qui ne nécessitent que $O(p)$ évaluations. Ainsi si, par exemple, on sait évaluer $f(z)$ pour un coût de $p^{1+o(1)}$ opérations binaires, on peut évaluer son intégrale avec un complexité de seulement $p^{2+o(1)}$ opérations. Il est à noter que ces méthodes ne s'appliquent que « localement » lorsque le chemin d'intégration est de longueur finie et reste éloigné des singularités ; dans le cas contraire, la tâche est souvent plus difficile.

Exemple : une intégrale épineuse

Les exemples pathologiques pour les algorithmes d'intégration numérique sont nombreux. L'un d'entre eux est l'« intégrale épineuse » [18]

$$\int_0^1 \operatorname{sech}^2(10(x-0{,}2)) + \operatorname{sech}^4(100(x-0{,}4)) + \operatorname{sech}^6(1000(x-0{,}6))\ dx.$$

Voici le résultat que l'on obtient lorsqu'on l'évalue avec la fonction `numerical_integral` de SageMath (qui fait appel au code d'intégration numérique de la bibliothèque GSL) :

```
sage: numerical_integral(lambda x: sech(10*x-2)**2
...         + sech(100*x-40)**4 + sech(1000*x-600)**6, 0, 1)
(0.2097360688339336, 6.166358647858423e-14)
```

Le deuxième nombre affiché est supposé être une estimation de l'erreur commise. Or, bien que celle-ci suggère que les 13 premières décimales sont correctes, en réalité, seulement 2 le sont. La raison en est que la fonction à

18. *Spike integral* en anglais.

intégrer, représentée sur la figure 4.3, a trois pics ; or, les points d'évaluation choisis par la bibliothèque GSL ratent la contribution du pic le plus étroit.

Le code d'intégration numérique en arithmétique d'intervalles de Arb fournit, quant à lui, un résultat rigoureux. De plus, il permet de calculer l'« intégrale épineuse » à haute précision très facilement :

```
sage: f = lambda x, _: (10*x-2).sech()**2 +
...         (100*x-40).sech()**4 + (1000*x-600).sech()**6
sage: ComplexBallField(53).integral(f, 0, 1)
[0.21080273550055 +/- 4.44e-15]
sage: ComplexBallField(333).integral(f, 0, 1)
[0.21080273550054927737564325570572915436090918643678119034
78505058787206131281455002050586892615576 4 +/- 3.67e-99]
```

L'algorithme sous-jacent à ce calcul exploite l'hypothèse que la fonction à intégrer est holomorphe (en dehors des pôles) ; elle utilise l'arithmétique d'intervalles pour confiner de manière contrôlée le chemin d'intégration suffisamment loin des pôles et ainsi obtenir des bornes rigoureuses sur les erreurs commises par les algorithmes d'intégration numérique (ici, une méthode de quadrature de GAUSS avec subdivision), comme illustré par la figure 4.3 [28].

La qualité de l'approximation finale obtenue par l'arithmétique d'intervalles peut être très sensible à l'ordre des opérations effectuées ainsi qu'au choix des fonctions élémentaires appelées. En guise de preuve par l'exemple, le lecteur pourra essayer d'utiliser `cosh(x)**-n` à la place de `sech(x)**n` et remarquer que le calcul devient alors bien plus lent.

4.5.3 Approximants

Enfin, une possibilité pour représenter une fonction à variable réelle ou complexe est de l'approcher par une fonction plus simple, appelée un *approximant*. Parmi les approximants les plus classiques, on peut citer les fonctions polynomiales, les fonctions polynomiales par morceaux, les polynômes trigonométriques et les fractions rationnelles. Chacun des types de fonctions suscités a en outre la propriété agréable qu'ils permettent une dérivation et une intégration exacte, terme à terme.

Un candidat naturel d'approximant d'une fonction est l'une des troncations de sa série de TAYLOR au voisinage d'un point a. Ce choix aboutit à une approximation optimale locale au voisinage de a. Au moins localement, seulement $O(p)$ termes sont nécessaires pour obtenir des approximations correctes à 2^{-p} près ; la complexité binaire d'une opération typique utilisant

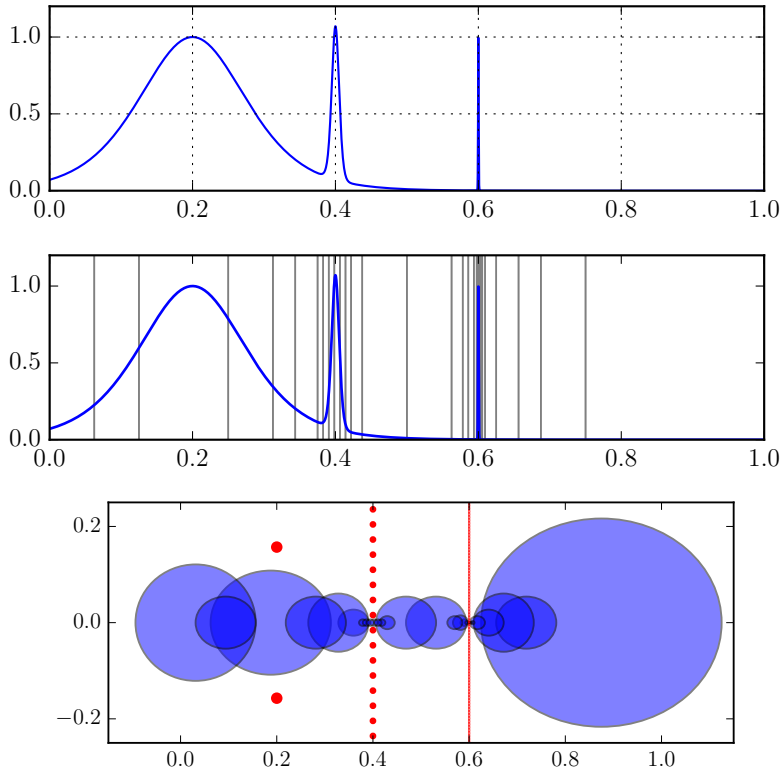

FIGURE 4.3 – *Haut : la « fonction épineuse » à intégrer. Milieu : les subdivisions choisies par le code d'intégration de Arb. Bas : les pôles de la « fonction épineuse » (points rouges) dans le plan complexe et les ellipses choisies par Arb pour confiner le chemin d'intégration loin des pôles et ainsi obtenir des bornes d'erreur rigoureuses.*

cette stratégie est donc de $p^{2+o(1)}$ étant donné que l'arithmétique polynomiale rapide basée sur la FFT a une complexité de $O(n \log n)$ opérations arithmétiques pour des polynômes de degré n.

Une série de TAYLOR tronquée accompagnée d'une borne sur l'erreur correspondante est appelée un *modèle de* TAYLOR [29]. Au delà de l'application évidente à la manipulation de fonctions, les modèles de TAYLOR peuvent être utilisées pour atténuer les problèmes de dépendance en arithmétique d'intervalles (voir §4.4.3) : plutôt que de modéliser une quantité à l'aide d'une constante avec terme d'erreur ε, on utilise une série de TAYLOR tronquée munie d'un terme d'erreur de la forme $C\varepsilon^N$; ce procédé peut rendre visibles certaines dépendances qui se compensent seulement à l'ordre ε^N.

Un autre choix naturel d'approximant est donné par les séries de TCHEBYCHEV tronquées. Ces dernières conduisent à de meilleures approximations uniformes. En outre, on dispose d'algorithmes rapides pour la manipulation de polynômes dans la base de TCHEBYCHEV, tout comme dans la base monomiale standard. Les développements de TCHEBYCHEV constituent le fondement de la bibliothèque *Chebfun* décrite par les auteurs comme « un analogue pour les fonctions de l'arithmétique en virgule flottante » [24]. Plus récemment, les développements de TCHEBYCHEV avec bornes d'erreur rigoureuses ont été considérées comme une alternative viable aux modèles de TAYLOR [25].

Bibliographie

[1] P. ZIMMERMANN et al. *Calcul mathématique avec Sage.* `http://sagebook.gforge.inria.fr/`, 2013.

[2] J. von zur GATHEN and J. GERHARD. *Modern Computer Algebra*. Cambridge University Press, 2013.

[3] R. P. BRENT and P. ZIMMERMANN. *Modern Computer Arithmetic*, Cambridge University Press, 2010. `http://www.loria.fr/~zimmerma/mca/mca-cup-0.5.7.pdf`.

[4] N. HIGHAM. The Rise of Multiprecision Computations, 2017. `https://www.maths.manchester.ac.uk/~higham/talks/samsi17.pdf`

[5] D. V. CHUDNOVSKY and G. V. CHUDNOVSKY. Computer algebra in the service of mathematical physics and number theory. *Computers in mathematics*, 125:109, 1990.

[6] D. H. BAILEY and J. M. BORWEIN. High-precision arithmetic in mathematical physics. *Mathematics*, 3(2):337–367, 2015.

[7] T. Y. CHOW. What is a closed-form number ? *The American Mathematical Monthly*, 106.5, 440–448 (1999).

[8] B. POONEN. Undecidable problems: a sampler. *Dans Interpreting Gödel : Critical Essays*, 211–241, 2014.

[9] S. M. RUMP. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[10] N. MÜLLER. The iRRAM: Exact arithmetic in C++. *Dans Computability and Complexity in Analysis*, pages 222–252. Springer, 2001. `http://irram.uni-trier.de`.

[11] N. REVOL and F. ROUILLIER. Motivations for an arbitrary precision interval arithmetic library and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.

[12] M. SOFRONIOU and G. SPALETTA. Precise numerical computation. *Journal of Logic and Algebraic Programming*, 64(1):113–134, 2005.

[13] W. TUCKER. A rigorous ODE solver and Smale's 14th problem. *Foundations of Computational Mathematics*, 2(1):53–117, 2002.

[14] W. TUCKER. *Validated numerics*: *a short introduction to rigorous computations*. Princeton University Press, 2011.

[15] J. van der HOEVEN. Fast evaluation of holonomic functions. *Theoretical Computer Science*, 210:199–215, 1999.

[16] J. van der HOEVEN. Ball arithmetic. HAL preprint, 2009. `http://hal.archives-ouvertes.fr/hal-00432152/fr/`.

[17] J. van der HOEVEN, G. LECERF, B. MOURRAIN, P. TRÉBUCHET, J. BERTHOMIEU, D. N. DIATTA, and A. MANTZAFLARIS. Mathemagix: the quest of modularity and efficiency for symbolic and certified numeric computation ? *ACM Communications in Computer Algebra*, 45(3/4):186–188, January 2012. `http://mathemagix.org`.

[18] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, and P. ZIMMERMANN. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13, 2007.

[19] T. GRANLUND and the GMP development team. *GNU MP*: *The GNU Multiple Precision Arithmetic Library*, 6.1.2 edition, 2017.

[20] A. ENGE, M. GASTINEAU, P. THÉVENY, and P. ZIMMERMANN. MPC: a library for multiprecision complex arithmetic with exact rounding. `http://www.multiprecision.org/mpc/`, 2018.

[21] D. H. BAILEY, J. M. BORWEIN, V. KAPOOR and E. W. WEISSTEIN. Ten Problems in Experimental Mathematics. *The American Mathematical Monthly* 113:481–509, 2006.

[22] D. RICHARDSON and J. FITCH. The identity problem for elementary functions and constants *Actes de ISSAC'94*, ACM, 285–290, 1994.

[23] P. LAIREZ, M. MEZZAROBBA and M. SAFEY EL DIN. Computing the volume of compact semi-algebraic sets arXiv preprint arXiv:1904.11705, 2019.

[24] T. A. DRISCOLL, N. HALE, and L. N. TREFETHEN, editors. *Chebfun Guide*. Pafnuty Publications, Oxford, 2014.

[25] F. BRÉHARD, N. BRISEBARRE and M. JOLDES. Validated and numerically efficient Chebyshev spectral methods for linear ordinary differential equations. *ACM Transactions on Mathematical Software*, 44(4), 1–42, 2018.

[26] A. SCHÖNHAGE and V. STRASSEN. Schnelle Multiplikation grosser Zahlen. *Computing* 7, 281–292, 1971.

[27] D. HARVEY and J. van der HOEVEN. Integer multiplication in time $O(nlogn)$. HAL preprint, 2019. `https://hal.archives-ouvertes.fr/hal-02070778`.

[28] F. JOHANSSON. Numerical integration in arbitrary-precision ball arithmetic. *Mathematical Software – ICMS 2018*, 255–263, 2018.

[29] M. BERZ and K. MAKINO. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing* 4, 361–369, 1998.

[30] M. KONTSEVICH and D. ZAGIER. Periods. Dans B. ENGQUIST et W. SCHMID (eds.), *Mathematics unlimited–2001 and beyond*, Berlin, New York : Springer-Verlag, 771–808, 2001.