Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Polynômes de meilleure approximation à coefficients flottants

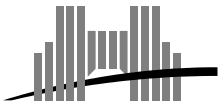
Sylvain Chevillard

Encadrants :
Nicolas Brisebarre
Jean-Michel Muller

Février - juin 2006

Rapport de M2 N° 2006-03

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



Polynômes de meilleure approximation à coefficients flottants

Sylvain Chevillard
Encadrants : *Nicolas Brisebarre* et *Jean-Michel Muller*

Février - juin 2006

Abstract

When one needs to compute approximated values of a function f at many points of an interval $[a, b]$, one generally replaces the function with another function, which approximates well f and which is simpler to evaluate. Polynomials are often used because one needs only additions and multiplications to evaluate them. The error made by replacing f with a polynomial p is $\varepsilon = p - f$. In the worst case, when one replaces f by p, one observes an error of $\max\{|\varepsilon(x)|,\ x \in [a, b]\}$. This is the infinite norm of ε and is denoted by $\|\varepsilon\|_\infty$. The general problem of searching a real-coefficient polynomial of degree not greater than n which minimizes $\|f - p\|_\infty$ has been deeply studied in the early years of the XX*th* century. Since every real number cannot be exactly stored in the finite memory of a computer, one generally uses the so-called floating-point numbers as an approximation of them. When implementing a function in a library, one uses polynomials with floating-point coefficients. But the best polynomial with such coefficients cannot be directly deduced from the best real-coefficient polynomial. The problem of searching the best polynomial with floating-point coefficients is thus an unsolved problem. We propose a heuristic to find a good polynomial to approximate f in infinite norm. We describe the method — which uses lattices theory and their reduction — and we illustrate its efficiency with a detailed example.

Keywords : polynomial approximation, floating-point numbers, lattices, LLL algorithm, CVP.

Résumé

Lorsqu'on a besoin de calculer des valeurs approchées d'une fonction f en de nombreux points d'un intervalle $[a, b]$, on remplace généralement f par un approximant plus simple à évaluer. Les polynômes sont souvent utilisés car leur évaluation ne nécessite que des additions et des muliplications. L'erreur commise en remplaçant la fonction f par le polynôme p est égale à $\varepsilon = p - f$. Dans le pire cas, remplacer f par p conduit donc à faire une erreur de $\max\{|\varepsilon(x)|,\ x \in [a, b]\}$. Cette quantité est appelée la norme infinie de ε et notée $\|\varepsilon\|_\infty$. Le problème général de la recherche d'un polynôme à coefficients réels de degré inférieur ou égal à n minimisant $\|p - f\|_\infty$ a été largement étudié au début du XX^e siècle. Cependant tous les réels ne sont pas exactement représentables dans la mémoire d'un ordinateur et on utilise généralement les nombres flottants comme approximation des nombres réels. Lorsqu'on implémente une fonction dans une bibliothèque logicielle, on utilise donc en pratique des polynômes à coefficients flottants. Il apparaît que le polynôme à coefficients flottants minimisant $\|f - p\|_\infty$ ne peut pas être directement déduit du polynôme de meilleure approximation à coefficients réels, ce qui relance donc le problème. Nous proposons une heuristique pour trouver un polynôme à coefficients flottants approchant bien une fonction f en norme infinie. Nous décrivons la méthode — qui s'appuie sur la théorie des réseaux et de leur reduction — et nous illustrons son efficacité à travers un exemple détaillé.

Mots-clés : approximation polynomiale, nombres flottants, réseaux, algorithme LLL, CVP.

Table des matières

1 Présentation du problème

1.1 Introduction

Beaucoup de réalisations techniques et de projets scientifiques d'ampleur nécessitent une grande masse de calculs scientifiques. Certains de ces calculs ne font intervenir que des opérations élémentaires telles que les additions ou les multiplications, mais la plupart nécessite l'utilisation de fonctions mathématiques plus complexes : calculs de fonctions trigonométriques, de fonctions algébriques (telle que la racine cubique par exemple), d'exponentielles, de fonctions spéciales, comme par exemple la fonction erf en statistiques et en économie.

Ces fonctions ont une définition mathématique abstraite qui ne permet pas un calcul explicite : la valeur $\cos(\theta)$ par exemple est définie comme l'abscisse du point du cercle unité formant l'angle θ avec l'axe $[Ox)$, la fonction exponentielle est définie comme la solution de l'équation différentielle $y' = y$ valant 1 en 0, la fonction erf est définie par une intégrale :

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t \quad .$$

Pour les calculer effectivement, il faut donc mettre au point des algorithmes capables de fournir des valeurs approchées de ces fonctions. Ces algorithmes doivent à la fois être rapides car ces fonctions sont impliquées dans des milliards de calculs, et le plus précis possible car les répercussions d'une erreur d'approximation trop grande peuvent être importantes.

L'arithmétique des ordinateurs est la branche de l'informatique qui se charge entre autres choses d'élaborer ces algorithmes. Comme les opérations de base telles que l'addition et la multiplication sont implantées avec soin et efficacité dans les processeurs modernes, il est naturel d'utiliser des polynômes pour approcher ces fonctions. On découpe l'intervalle de définition de la fonction en plusieurs intervalles sur chacun desquels la fonction est bien approchée par un polynôme de degré raisonnable (le degré sera inférieur à 5 pour des applications matérielles et pourra aller jusqu'à quelques dizaines pour des applications logicielles). C'est ce polynôme qu'on évalue en réalité lorsqu'on veut calculer une valeur approchée de la fonction en un point.

Le but étant naturellement de trouver le polynôme le plus précis possible, le problème suivant se pose : étant donnée une fonction continue f, un intervalle $[a, b]$ sur lequel f est bien définie, et un entier n, que dire de $\|f - p\|_\infty$ lorsque p décrit l'ensemble des polynômes à coefficients réels de degré inférieur ou égal à n (noté $\mathbb{R}_n[X]$) ? Rappelons que $\|f - p\|_\infty$ s'appelle la norme L_∞ de $f - p$ et représente la valeur maximale atteinte par l'erreur $|f(x) - p(x)|$ quand x décrit l'intervalle $[a, b]$. Ce maximum est bien défini car $f - p$ est une fonction continue et $[a, b]$ est un compact de $\mathbb{R}$.

Est-ce que $\|f - p\|_\infty$ peut prendre des valeurs arbitrairement petites lorsque p varie dans $\mathbb{R}_n[X]$? Sinon, quelle est la valeur minimale et comment trouver un polynôme p la réalisant ?

1.2 Travaux précédents

Ces questions ont été étudiées dès le XIX^e siècle par les mathématiciens ; à cette époque où les ordinateurs n'existaient pas encore, les polynômes représentaient plus encore que maintenant la voie naturelle pour calculer des approximations de fonctions et les mathématiciens se sont donc naturellement trouvés confrontés à ces questions. À présent que les ordinateurs permettent d'effectuer une multitude de calculs en une fraction de seconde, d'autre méthodes sont parfois utilisées pour calculer des approximations de fonctions (utilisation de fractions continues, résolution numérique d'équations différentielles, par exemple). Cependant, les polynômes restent encore l'outil le plus utilisé et constitue un sujet d'étude central.

1.2.1 Approche classique

Des grands noms sont associés à la théorie de l'approximation polynomiale : Bernstein, Féjer, Lagrange, Lebesgue, Runge, Tchebychev, Weierstrass... Parmi eux, Tchebychev se dégage particulièrement à propos des questions qui nous occupent. Il a montré que $\|f - p\|_\infty$ est borné inférieurement lorsque p décrit $\mathbb{R}_n[X]$ et a montré l'existence et l'unicité du polynôme p réalisant l'optimum. En outre, il a caractérisé de façon très pratique le polynôme de meilleure approximation (on trouvera un exposé détaillé de ces

théorèmes dans [16] par exemple). En somme, ne manquait plus qu'un algorithme permettant de calculer ce polynôme de meilleure approximation.

En 1934, dans [13], Remes décrit un algorithme permettant de calculer effectivement le polynôme optimal. En outre, cet algorithme converge de façon quadratique, ce qui en fait un outil pratique très efficace ; le problème peut donc être considéré comme clos.

1.2.2 Précisons le problème

L'usage de l'ordinateur introduit de nouvelles contraintes et remet le problème à l'ordre du jour : en effet, un ordinateur ne peut pas stocker exactement les coefficients réels du polynôme de meilleure approximation. En pratique, un ordinateur manipule des nombres appelés *nombres flottants* et qui ont été standardisés par la norme IEEE-754. Pour des applications particulières, on peut aussi utiliser un format différent appelé *virgule fixe* ; pour simuler une plus grande précision, il arrive aussi parfois qu'on représente un nombre sous la forme d'une somme non évaluée de deux ou trois nombres flottants. Sans s'attarder pour l'instant sur ces nuances, disons simplement qu'un nombre flottant de précision t en base β est un nombre de la forme $s \cdot m \cdot 2^e$ où $s \in \{-1, 1\}$ est le signe, $m \in [\![\beta^{t-1}, \beta^t - 1]\!]$ * est la mantisse (qui est donc un nombre entier de t β-chiffres), et $e \in [\![E_{\min}, E_{\max}]\!]$ est l'exposant. Par exemple, le format `double` de la norme IEEE-754 correspond à $\beta = 2$, $t = 53$, $E_{\min} = -1074$ et $E_{\max} = 971$.

Pour approcher une fonction sur un intervalle $[a, b]$ par un polynôme à coefficients flottants, on procède donc habituellement de la façon suivante : on calcule le polynôme de meilleure approximation à l'aide de l'algorithme de Remes, en utilisant une grande précision pour les calculs. On arrondit ensuite les coefficients du polynôme de meilleure approximation au nombre flottant le plus proche. Le polynôme ainsi obtenu est celui utilisé par la suite pour approcher la fonction.

Il n'est pas évident du tout *a priori* que ce polynôme, naïvement arrondi, soit le meilleur polynôme en norme L_∞ parmi ceux dont les coefficients sont des nombres flottants. En réalité, ce polynôme n'est la plupart du temps pas optimal ; pire, il arrive fréquemment que le polynôme à coefficients flottants optimal soit nettement meilleur que celui obtenu par le procédé susdit. Considérons un exemple aussi simple que celui-ci : approcher la fonction $f : x \mapsto \sqrt{2} + \pi x + ex^2$ sur l'intervalle $[2, 4]$ par un polynôme de degré 2 à coefficients au format `double` IEEE-754. Le polynôme de meilleure approximation est la fonction f elle-même. Si on arrondit les coefficients aux `double` les plus proches, on obtient le polynôme

$$\widehat{P} = \frac{6369051672525773}{4503599627370496} + \frac{884279719003555}{281474976710656}X + \frac{6121026514868073}{2251799813685248}X^2$$

et on a $\|f - \widehat{P}\|_\infty \simeq 2.70622 \cdot 10^{-15}$. Or le meilleur polynôme à coefficients flottants dans le cas présent est le polynôme

$$P^* = \frac{6369051672525769}{4503599627370496} + \frac{3537118876014221}{1125899906842624}X + \frac{6121026514868073}{2251799813685248}X^2$$

pour lequel $\|f - P^*\|_\infty \simeq 2.2243 \cdot 10^{-16}$. Il y a donc plus d'un facteur 10 entre l'optimum et le polynôme arrondi.

Cette remarque était bien connue, mais il semble que personne jusqu'à peu n'avait réussi à attaquer avec succès le problème de la recherche du polynôme de meilleure approximation à coefficients flottants. En 2005, Brisebarre, Muller et Tisserand ont proposé une première approche (voir [3]). Pour simplifier la modélisation, ils font abstraction des limites $E_{\min}$ et $E_{\max}$ en les supposant respectivement égales à $-\infty$ et $+\infty$. À supposer qu'on ait une idée de l'ordre de grandeur des coefficients recherchés, on peut fixer l'exposant de sorte que la recherche d'un coefficient se ramène à la recherche de sa mantisse. Ils formulent donc le problème sous la forme suivante : étant donnés, des entiers $m_0, \cdots, m_n$, trouver parmi les polynômes de la forme

$$P = \sum_{i=0}^{n} \frac{a_i}{2^{m_i}} X^i \quad \text{avec } a_i \in \mathbb{Z}$$

ceux qui minimisent $\|f - P\|_\infty$. Il faut noter que l'existence de tels polynômes est assurée mais l'unicité n'a pas forcément lieu : il peut y avoir une multitude de solutions au problème. Dans leur approche, Brisebarre, Muller et Tisserand les trouvent tous. Pour ce faire, ils circonscrivent les solutions possibles dans un polytope de l'espace $\mathbb{R}^n$ puis ils parcourent les points entiers de ce polytope pour en extraire les points qui fournissent effectivement des optima.

∗. $[\![a, b]\!]$ désigne l'ensemble des entiers compris entre a et b.

1.3 Limites des travaux précédents

Cette méthode présente l'avantage de fournir toutes les solutions au problème, mais elle souffre de deux inconvénients majeurs.

D'abord, l'algorithme dépend de deux paramètres, lesquels influent notablement sur la forme du polytope qu'ils parcourent, et par suite, sur les performances de l'algorithme. L'un de ces paramètres est une estimation de l'erreur optimale. Si cette erreur est trop surévaluée, le polytope est très gros et contient beaucoup trop de points ; si elle est sous-évaluée, le polytope ne contient aucune solution (dans ce cas, la plupart du temps, il ne contient aucun point à coordonnées entières, mais il faut néanmoins beaucoup de temps pour s'en apercevoir). Or la seule information dont on dispose, c'est que cette erreur est supérieure à celle du polynôme de meilleure approximation réel et qu'elle est inférieure à celle du polynôme de meilleure approximation naïvement arrondi. Comme on l'a vu sur le petit exemple, ces estimations peuvent être de piètre qualité (dans l'exemple précédent, l'erreur du polynôme réel optimal est nulle ; quand on l'arrondit naïvement, l'erreur obtenue est de l'ordre de $2.7 \cdot 10^{-15}$: la marge est très importante. En l'occurrence, on peut descendre à une erreur de l'ordre de $2.3 \cdot 10^{-16}$, mais comment le deviner ?)

Ensuite, la complexité algorithmique de leur méthode est difficile à établir en toute généralité, mais on peut craindre qu'elle reste exponentielle dans le pire cas. Cela ne signifie pas nécessairement que la méthode est définitivement inutilisable car il se pourrait qu'elle soit polynomiale dans les cas pratiques. D'ailleurs, elle fonctionne déjà de manière satisfaisante en petit degré (jusqu'à 4 ou 5). Néanmoins, cela incite à chercher d'autres angles d'attaque au problème.

1.4 Problématique

La recherche d'un polynôme optimal est intéressante sur le plan théorique mais n'est pas forcément nécessaire dans la pratique. Ce qui intéresse les gens qui implémentent des fonctions, c'est avant tout d'avoir en un temps raisonnable un polynôme aussi bon que possible : peu importe si ce n'est pas exactement l'optimum, du moment que l'erreur est du même ordre de grandeur que l'erreur optimale.

Nous proposons une méthode pour obtenir en temps polynomial un bon polynôme à coefficients flottants. L'écart à l'optimal est difficile à quantifier théoriquement et nous ne fournissons pour l'instant aucune garantie sur la qualité du polynôme fourni par notre méthode. Néanmoins, son utilisation pratique montre qu'elle permet d'améliorer sensiblement les résultats fournis par le polynôme de meilleure approximation arrondi.

En outre, notre méthode permet donc de fournir à l'algorithme de Brisebarre, Muller et Tisserand, une estimation de l'erreur optimale bien plus fine et souvent du bon ordre de grandeur. Ainsi, nous améliorons les performances de leur approche.

Formellement, nous cherchons, sinon le meilleur, du moins un bon polynôme de la forme

$$P = \sum_{i=0}^{n} \frac{a_i}{2^{m_i}} X^i \quad \text{avec } a_i \in \mathbb{Z}$$

pour approcher f en norme L_∞. Pour ce faire, nous allons nous appuyer sur des résultats de théorie de l'approximation polynomiale et sur des résultats de théorie des réseaux. Après avoir présenté ces outils théoriques, nous verrons comment ils s'appliquent à la résolution du problème qui nous occupe.

2 Prérequis théoriques

2.1 Interpolation polynomiale

La méthode la plus naturelle pour approcher une fonction f par un polynôme de degré inférieur ou égal à n sur l'intervalle $[a, b]$ consiste à choisir des points dans l'intervalle et à imposer au polynôme d'égaler la fonction en ces points. Cette méthode est connue sous le nom de *méthode d'interpolation*. Puisqu'il y a $n+1$ coefficients à déterminer, il y a $n+1$ degrés de libertés ; si l'on choisit $n+1$ points distincts dans l'intervalle, on a donc un unique polynôme interpolant la fonction aux points considérés.

Plus formellement, l'espace des polynômes de degré inférieur ou égal à n est un espace vectoriel de dimension $n+1$. Si $x_0, \cdots, x_n$ sont $n+1$ points distincts de $[a, b]$, l'application $\Theta : P \mapsto (P(x_0), \cdots, P(x_n))$

est linéaire, injective (car le seul polynôme de $\mathbb{R}_n[X]$ à avoir $n+1$ racines est le polynôme nul) entre deux espaces vectoriels de même dimension finie $n+1$: c'est donc un isomorphisme. En particulier, cette application est surjective et il existe donc un unique polynôme P tel que $(P(x_0), \cdots, P(x_n)) = (f(x_0), \cdots, f(x_n))$.

Un moyen d'obtenir explicitement ce polynôme est d'utiliser les polynômes de Lagrange qui sont les images inverses de la base canonique de $\mathbb{R}^{n+1}$ par Θ. Une autre manière de procéder consiste à écrire la matrice A de Θ dans les bases canoniques de $\mathbb{R}_n[X]$ et $\mathbb{R}^{n+1}$ et à résoudre le système $A\alpha = Y$ où $\alpha = {}^t(a_0, \cdots, a_n)$ est le vecteur des inconnues et $Y = {}^t(f(x_0), \cdots, f(x_n))$ est l'image à atteindre. Le coefficient (i, j) de la matrice A est par définition la i-ème composante de $\Theta(X^{j-1})$ de sorte que

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \quad .$$

Si α désigne la solution du système $A\alpha = Y$, le polynôme interpolateur de f aux points x_i est donc

$$P = \sum_{i=0}^{n} a_i X^i \quad .$$

Une question reste en suspens : comment choisir en pratique les points x_i ? On peut les prendre équirépartis dans l'intervalle $[a, b]$. Malheureusement, il existe des fonctions f pour lesquelles cela conduit à des polynômes p tels que $\|p - f\|_\infty$ n'est pas petit du tout. Ce phénomène, connu sous le nom de *phénomène de Runge* est décrit en annexe A.

Néanmoins, nous allons voir que, lorsqu'on choisit bien les points d'interpolation, cette méthode peut donner d'excellents résultats.

2.2 Approximation polynomiale

Nous allons aborder ici le problème de l'existence et de la recherche du polynôme de meilleure approximation. Cette question a été particulièrement étudiée par Weierstrass ([18]), Bernstein ([2]), Tchebychev, ou encore la Vallée Poussin et Remes ([13]).

L'ensemble

$$\{\|f - p\|_\infty\}_{p \in \mathbb{R}_n[X]}$$

est un ensemble de réels positifs ; il admet donc une borne inférieure que nous notons μ. Le premier problème consiste à montrer que μ est en réalité un minimum, c'est-à-dire qu'il existe effectivement un polynôme de meilleure approximation. On utilise pour cela un argument de compacité : soit $(p_i)_{i \in \mathbb{N}}$ une suite de polynômes tels que $\|f - p_i\|_\infty \to \mu$ lorsque i tend vers $+\infty$. Alors pour i assez grand, $\|f - p_i\|_\infty \leq \mu + 1$ d'où, par inégalité triangulaire, $\|p_i\|_\infty \leq \mu + 1 + \|f\|_\infty$. Par conséquent, la suite p_i est bornée dans l'espace vectoriel de dimension finie $\mathbb{R}_n[X]$; elle admet donc une sous-suite convergente. Notons p la limite de cette sous-suite. Comme $\|f - p_i\|_\infty \to \mu$, on a après passage à la limite $\|f - p\|_\infty = \mu$.

La question de l'unicité est plus délicate. On utilise en réalité un théorème de Tchebychev qui donne une caractérisation très utile des polynômes de meilleure approximation. Nous nous contenterons ici de le citer. Le lecteur intéressé pourra consulter une preuve dans [16].

Théorème 2.1 (Tchebychev)**.** *Soit f une fonction continue sur un intervalle $[a, b]$. Soit n un entier et soit $\mu = \inf\{\|f - p\|_\infty\}_{p \in \mathbb{R}_n[X]}$.*

Un polynôme p vérifie $\|f - p\|_\infty = \mu$ (autrement dit : est un polynôme de meilleure approximation) si et seulement s'il existe $n+2$ points

$$x_0 \ < \ x_1 \ < \ \cdots \ < \ x_{n+1}$$

dans $[a, b]$ vérifiant

1. *$\forall i \in [\![0, n+1]\!]$, $|f(x_i) - p(x_i)| = \|f - p\|_\infty$*
2. *Pour tout $i \in [\![0, n]\!]$, le signe de $f(x_{i+1}) - p(x_{i+1})$ est opposé au signe de $f(x_i) - p(x_i)$.*

Autrement dit p est optimal si et seulement si la fonction $f - p$ atteint au moins $n + 2$ fois sa valeur extrême et en changeant de signe entre deux x_i.

En utilisant ce théorème, on démontre aisément l'unicité du polynôme de meilleure approximation :

Proposition 2.1. *Le polynôme de meilleure approximation est unique.*

Démonstration. Voir annexe B. □

Le théorème de Tchebychev donne lieu à des raffinements : un théorème de la Vallée Poussin relie la qualité d'une approximation polynomiale p aux oscillations de $f-p$: informellement, si $f-p$ oscille $n+2$ fois, et qu'on note α et β respectivement la hauteur de l'oscillation de plus petite amplitude et de plus grande amplitude, on a $\alpha \leq \mu \leq \beta$, où μ désigne l'erreur du polynôme de meilleure approximation. Ceci fournit donc un critère pour juger *a priori* de la qualité d'une approximation polynômiale, sans avoir à connaître effectivement le polynôme de meilleure approximation. Pour plus de détails sur ce théorème, on consultera [16].

Les théorèmes de Tchebychev et de la Vallée Poussin sont à l'origine de l'algorithme de Remes (voir algorithme 1). Pour démarrer l'algorithme, il faut choisir $n+2$ points $y_0, \cdots, y_{n+1}$ dans l'intervalle $[a, b]$. Notons $x_0, \cdots, x_{n+1}$ les points du théorème de Tchebychev. Si les points y_i sont suffisamment près des points x_i l'algorithme de Remes converge quadratiquement. On se reportera à [5] pour une preuve de la convergence et à [17] pour une preuve de la vitesse de convergence.

Pour finir, remarquons une dernière chose qui nous permet de faire le lien avec la méthode d'interpolation. Le théorème de Tchebychev admet le corollaire suivant :

Corollaire 2.1. *Si p est le polynôme de meilleure approximation de f sur $[a, b]$, $f-p$ possède au moins $n+1$ zéros distincts : $z_0, \cdots, z_n$.*

Démonstration. Voir annexe B. □

Comme nous l'avons vu page 6, un mauvais choix des points dans la méthode d'interpolation peut conduire à des résultats désastreux (divergence de la suite des polynômes interpolateurs). Mais ce corollaire nous indique que pour toute fonction f et pour tout degré n, il existe $n+1$ points (les z_i du corollaire) tels que le polynôme interpolateur de f aux points z_i réalise la meilleure approximation. Cette remarque sera d'une grande importance pour notre méthode.

2.3 Réseaux

Nous abordons à présent l'étude d'un outil central de notre méthode : les réseaux. Les réseaux sont une structure algébrique discrète, ce qui en fait un outil de modélisation très commode dans de nombreux problèmes informatiques. Ils trouvent des applications notamment en cryptanalyse, mais aussi dans divers autres domaines de l'informatique (on pourra par exemple consulter [11], [15], [12] ou [14]). Ils apparaissent naturellement dans de nombreuses circonstances, en particulier dans certains domaines de la physique, tels que la cristallographie. Notre méthode consiste à transposer le problème de la recherche d'un polynôme à coefficients flottants en la recherche d'un vecteur à coefficients entiers dans un certain espace vectoriel. La structure de réseau sera partout présente dans cette approche.

2.3.1 Définition des réseaux

Définissons tout d'abord la notion de réseau :

Définition 2.1 (Réseau)**.** *On se donne un $\mathbb{R}$-espace vectoriel E de dimension finie n et une famille libre $(\overrightarrow{b_1}, \cdots, \overrightarrow{b_p})$ de E. L'ensemble*

$$\Gamma = \left\{ \sum_{j=1}^{p} \lambda_j \overrightarrow{b_j} \right\}_{(\lambda_1, \cdots, \lambda_p) \in \mathbb{Z}^p}$$

est appelé réseau. La famille $(\overrightarrow{b_1}, \cdots, \overrightarrow{b_p})$ est appelée une base de Γ.

Algorithme : Remes

Data : une fonction f continue sur un intervalle $[a,\, b]$;
la précision visée : ε_0

Result : un polynôme p tel que $\|f-p\|_\infty \leq (1+\varepsilon_0)\|f-p^*\|_\infty$
(où p^* désigne le polynôme de meilleure approximation de f sur $[a,\, b]$)

begin

Choisir $n+2$ points $y_0, \cdots, y_{n+1}$ dans $[a,\, b]$;
$\varepsilon := +\infty$;
while $\varepsilon > \varepsilon_0$ **do**
Résoudre le système d'inconnues $a_0, \cdots, a_n, \varepsilon$:

$$\begin{cases} \sum_{i=0}^{n} a_i y_0^i & - & f(y_0) & = & (-1)^0\varepsilon \\ & & \vdots & & \\ \sum_{i=0}^{n} a_i y_{n+1}^i & - & f(y_{n+1}) & = & (-1)^{n+1}\varepsilon \end{cases}$$

Poser $p := a_0 + a_1 X + \cdots + a_n X^n$;
Chercher les extrema locaux de $f-p$: $y_0, \cdots, y_{n+1}$;
$\alpha := \min_i |f(y_i) - p(y_i)|$;
$\beta := \max_i |f(y_i) - p(y_i)|$;
$\varepsilon := \beta/\alpha - 1$;
end
return p ;
end

Algorithme 1 : Algorithme de Remes

Par exemple, $\mathbb{Z}$ est un réseau de l'espace $E = \mathbb{R}$, de base 1 ; $\mathbb{Z}\times\mathbb{Z}\sqrt{3}$ est un réseau de l'espace $E = \mathbb{R}^2$ de base

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \sqrt{3} \end{pmatrix} \quad .$$

Remarque 1. *Lorsque $E = \mathbb{R}^n$, il est naturellement muni de sa base canonique. En écrivant les vecteurs dans cette base, on peut considérer la matrice $B \in \mathcal{M}_{n,\,p}(\mathbb{R})$ dont les colonnes sont les $\overrightarrow{b_j}$ exprimés dans la base canonique. Notons alors que, si Λ désigne le vecteur $(\lambda_1, \cdots, \lambda_p)^t$, $B\Lambda$ est l'expression du vecteur $\sum_{j=1}^{p} \lambda_j \overrightarrow{b_j}$ dans la base canonique. En particulier, on a*

$$\Gamma = B \cdot \mathcal{M}_{p,\,1}(\mathbb{Z}) \quad .$$

En outre, si M est une matrice à p lignes et r colonnes, la k-ième colonne du produit BM est l'expression dans la base canonique du vecteur $\sum_{j=1}^{p} \lambda_j \overrightarrow{b_j}$ où $(\lambda_1, \cdots, \lambda_p)^t$ désigne la k-ième colonne de M.

Les réseaux partagent certaines propriétés classiques des espaces vectoriels :

Proposition 2.2. *Soient $(\overrightarrow{b_1}, \cdots, \overrightarrow{b_p})$ et $(\overrightarrow{c_1}, \cdots, \overrightarrow{c_q})$ deux bases d'un réseau Γ. Alors $p = q$; on appelle p le rang du réseau. En outre, la matrice de passage de B à C est unimodulaire, c'est-à-dire qu'elle est dans $\mathcal{M}_{p,p}(\mathbb{Z})$ et de déterminant égal à ± 1.*

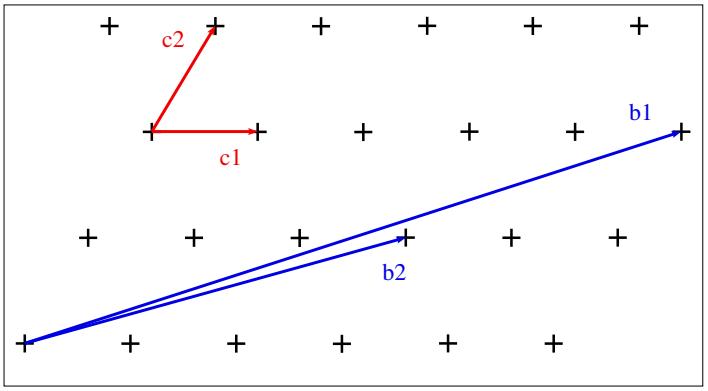
En revanche, certains aspects des réseaux sont plus déroutants : un réseau Γ de rang p étant donné, il existe un sous-réseau strict $\Gamma' \subset \Gamma$ qui est aussi de rang p. Dit autrement : il existe des familles de p éléments de Γ qui sont libres mais qui ne constituent pas une base de Γ. Par exemple, dans le réseau $\mathbb{Z}\times\mathbb{Z}\sqrt{3}$, la famille

$$\begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \sqrt{3} \end{pmatrix}$$

est libre, mais elle engendre le réseau $2\,\mathbb{Z} \times \mathbb{Z}\sqrt{3}$.

Une conséquence notable de la proposition 2.2 est qu'un réseau de rang supérieur ou égal à 2 possède une infinité de bases : en effet, étant donné une base, on peut toujours en obtenir de nouvelles en multipliant la matrice associée par une matrice unimodulaire. Il faut avoir conscience que certaines bases sont plus adaptées que d'autres pour manipuler un réseau donné. Par exemple, observons la figure 1 :

FIGURE 1 – Deux bases d'un même réseau.



les familles $(\overrightarrow{b_1}, \overrightarrow{b_2})$ et $(\overrightarrow{c_1}, \overrightarrow{c_2})$ sont deux bases d'un même réseau. Il apparaît clairement que la base C est mieux adaptée car elle donne une vision locale du réseau : lorsqu'on ajoute un vecteur de base à un point du réseau, on obtient un point situé à proximité. En revanche, avec la base B, un vecteur de base nous éloigne considérablement plus du point de départ.

Si on cherche à explorer le voisinage d'un point dans un réseau (ce qui sera notre objectif dans peu de temps), une base est d'autant plus pratique à utiliser qu'elle est constituée de vecteurs courts : de cette façon, lorsqu'on ajoute un vecteur de base à un point du réseau on ne s'écarte pas trop de ce point. Ainsi, la base C sera mieux adaptée à l'exploration que la base B.

Cependant, lorsqu'un réseau nous est donné sous la forme d'une base, rien ne garantit *a priori* que cette base soit bien adaptée. Aussi faut-il apporter un soin particulier au choix de la base qu'on utilise pour manipuler le réseau. Mais comme on va le voir, il n'est pas aisé de définir précisément le concept de *base bien adaptée*.

2.3.2 Plus courts vecteurs d'un réseau

Un premier problème algorithmique apparaît donc : comment trouver un vecteur non nul de norme minimale dans un réseau ? Ce problème est connu sous le nom de *shortest vector problem* (SVP). Il n'est pas évident *a priori* qu'un tel vecteur existe : il pourrait très bien y avoir une suite de vecteurs tous non nuls mais de normes tendant vers 0. Le théorème suivant justifie l'existence d'une solution à SVP :

Théorème 2.2. *Les réseaux de $\mathbb{R}^n$ sont exactement les sous-groupes additifs discrets de $\mathbb{R}^n$.*

Corollaire 2.2. *Dans tout réseau, il existe (au moins) un plus court vecteur non nul.*

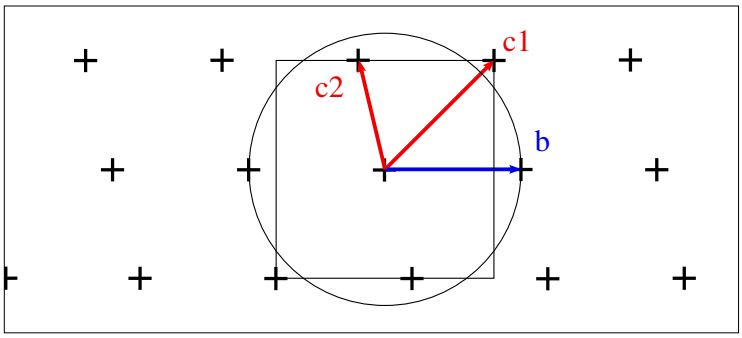
Démonstration. On consultera par exemple [7]. □

Il faut noter que le problème SVP dépend de la norme qu'on choisit sur $\mathbb{R}^n$. En dimension finie, toutes les normes d'un espace vectoriel sont équivalentes, c'est-à-dire (en particulier) qu'elles définissent la même topologie. Cela implique que *l'existence* d'un plus court vecteur est indépendante de la norme choisie. En revanche le plus court vecteur lui-même dépend de la norme. La figure 2 montre un exemple en dimension 2 avec la norme euclidienne (dont la boule est un disque) et avec la norme infinie (la norme infinie d'un vecteur est le maximum des valeurs absolues des composantes de ce vecteur) dont la boule est un carré.

On notera que la solution à SVP n'est pas forcément unique [†] : par exemple, dans $\mathbb{Z}^2$ muni de la

†. Remarque : si $\overrightarrow{v}$ est un plus court vecteur, $-\overrightarrow{v}$ l'est aussi. Quand on parle d'unicité ici, c'est toujours modulo passage à l'opposé.

FIGURE 2 – Solutions différentes de SVP suivant la norme choisie : le vecteur $\vec{b}$ est le plus court vecteur en norme euclidienne mais pas en norme infinie.



norme euclidienne habituelle, les vecteurs (1, 0) et (0, 1) sont deux solutions linéairement indépendantes de SVP.

Cependant, le théorème 2.2 et son corollaire ne sont pas constructifs. On a longtemps soupçonné — sans pouvoir précisément quantifier — que la recherche d'un plus court vecteur dans un réseau était un problème algorithmiquement dur. On sait depuis quelques années qu'il s'agit d'un problème NP-dur. ‡

Cela étant, ce qui nous intéressera par la suite est de posséder une base permettant de se déplacer localement dans le réseau, afin d'explorer le voisinage d'un point. Il n'est donc pas forcément indispensable de connaître le plus court vecteur. D'ailleurs une base constituée d'un vecteur de norme minimale et de vecteurs beaucoup plus grands ne nous serait que de peu d'intérêt. Au contraire, on cherche à ce que *tous* les vecteurs de la base soient *assez* courts (il existe des problèmes pour lesquels seule la connaissance d'un vecteur très court est nécessaire ; mais dans notre cas, nous avons besoin d'une base entièrement formée de vecteurs les plus courts possibles. Cette nécessité apparaîtra clairement lorsqu'on décrira l'algorithme ApproximatedCVP qui calcule une approximation d'un plus proche vecteur dans un réseau). En outre, on souhaite que la base soit la plus orthogonale possible : ainsi, on pourra l'utiliser pour se déplacer avec la même aisance dans n'importe quelle direction de l'espace (en réalité l'orthogonalité et la longueur des vecteurs d'une base sont deux paramètres liés ; cependant on contrôlera préférentiellement l'un ou l'autre des deux paramètres suivant ce sur quoi on veut mettre l'accent).

Il n'y a pas de caractérisation universelle de ce qu'on pourrait appeler une *base idéale* d'un réseau. L'approche la plus naturelle consisterait à demander que le plus court vecteur soit dedans, ainsi que le plus court vecteur linéairement indépendant du premier, puis le plus court linéairement indépendant des deux premiers, et ainsi de suite. Pour la norme euclidienne, cette approche est valide jusqu'à la dimension 4. Mais à partir de la dimension 5, il se peut qu'aucune base ne vérifie cette propriété. Différentes notions de bases réduites ont alors été définies, qui collent plus ou moins à cet idéal inaccessible : citons par exemple la réduction de Hermite, de Minkowski, de Korkine-Zolotarev, etc. On pourra se reporter à [10] pour avoir un aperçu des différentes réductions et une liste de références plus complète.

Puisque SVP est NP-dur, aucune de ces réductions n'est effectivement calculables en un temps raisonnable. On cherche alors des critères de réduction approchée : on renonce à exiger d'avoir le plus court vecteur dans la base réduite, mais on essaie néanmoins d'obtenir, en un temps raisonnable, une base *assez* orthogonale et formée de vecteurs *assez* courts. On va voir que l'on peut atteindre cet objectif en temps polynomial avec un algorithme dont les résultats pratiques sont de très grande qualité.

2.4 Algorithme LLL

En 1982, A. K. Lenstra, H. W. Lenstra Jr. et L. Lovász proposent une définition de réduction (la réduction LLL) qui concilie l'exigence d'orthogonalité approchée et de vecteurs courts et ils fournissent un algorithme (l'algorithme LLL) qui produit une base LLL-réduite en temps polynomial (voir [8]).

Cet algorithme va être au centre de notre méthode car il donne une base bien adaptée pour se

‡. En réalité, le problème est NP-dur sous des réductions randomisées, et non pas NP-dur au sens classique. Pour plus de détails, voir par exemple [1], [4] et [9]).

promener dans le réseau. Cette base n'est certes pas théoriquement parfaite, mais comme on le verra, elle est en pratique très souvent bien adaptée pour travailler dans le réseau.

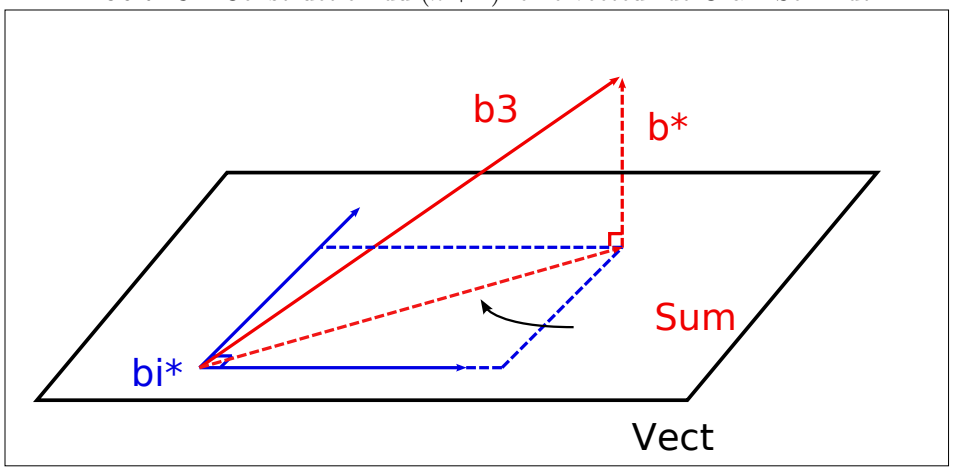
L'objectif de l'algorithme LLL est d'obtenir une base la plus orthogonale possible ; la notion d'orthogonalité étant une notion purement euclidienne, c'est évidemment cette norme qui est choisie pour normer l'espace. Par la suite, sauf contre-indication explicite, la norme utilisée sera donc la norme euclidienne. Si $\overrightarrow{x} = (x_1, \cdots, x_n)$ et $\overrightarrow{y} = (y_1, \cdots, y_n)$ sont deux vecteurs, nous noterons $\langle \overrightarrow{x}, \overrightarrow{y} \rangle = \sum_i x_i y_i$ le produit scalaire de $\overrightarrow{x}$ et $\overrightarrow{y}$. Pour rappel, la norme euclidienne est définie par $\| \overrightarrow{x} \|^2 = \langle \overrightarrow{x}, \overrightarrow{x} \rangle$.

À toute famille libre $(\overrightarrow{b_1}, \cdots, \overrightarrow{b_p})$ d'un espace vectoriel euclidien est associée une famille orthogonale $(\overrightarrow{b_1^*}, \cdots, \overrightarrow{b_p^*})$, appelée *famille de Gram-Schmidt*, qui vérifie $\forall i$, $\text{Vect}(\overrightarrow{b_1}, \cdots, \overrightarrow{b_i}) = \text{Vect}(\overrightarrow{b_1^*}, \cdots, \overrightarrow{b_i^*})$. Cette famille est au cœur de l'algorithme LLL et il est nécessaire de se familiariser avec.

Définition 2.2 (Famille de Gram-Schmidt)**.** *On définit la famille de Gram-Schmidt associée à la famille* $(\overrightarrow{b_1}, \cdots, \overrightarrow{b_p})$ *par récurrence de la façon suivante :*

$$\begin{cases} \overrightarrow{b_1^*} = \overrightarrow{b_1} \\ \overrightarrow{b_{k+1}^*} = \overrightarrow{b_{k+1}} - \displaystyle\sum_{j=1}^{k} \frac{\langle \overrightarrow{b_{k+1}}, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*}, \overrightarrow{b_j^*} \rangle} \overrightarrow{b_j^*} \end{cases} .$$

FIGURE 3 – Construction du $(k+1)$-ième vecteur de Gram-Schmidt.



Géométriquement, la somme dans la définition de $\overrightarrow{b_{k+1}^*}$ *est la projection orthogonale du vecteur* $\overrightarrow{b_{k+1}}$ *sur l'espace* $E_k = \text{Vect}(\overrightarrow{b_1^*}, \cdots, \overrightarrow{b_k^*})$*. Ainsi,* $\overrightarrow{b_{k+1}^*}$ *est la composante de* $\overrightarrow{b_{k+1}}$ *orthogonale à* E_k*.*

Une récurrence évidente montre que $\forall i$, $\text{Vect}(\overrightarrow{b_1}, \cdots, \overrightarrow{b_i}) = \text{Vect}(\overrightarrow{b_1^*}, \cdots, \overrightarrow{b_i^*})$ *et que la famille de Gram-Schmidt est orthogonale.*

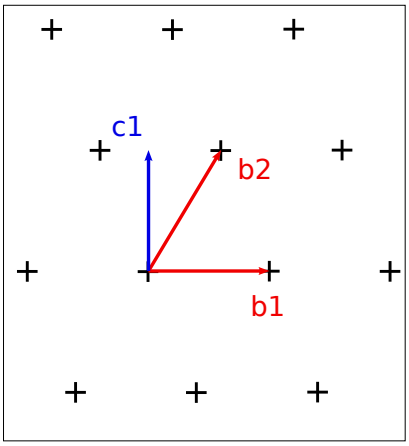
On notera que la famille de Gram-Schmidt associée à une famille dépend de l'ordre dans lequel sont rangés les vecteurs de la famille. On notera aussi que la famille de Gram-Schmidt associée à la base d'un réseau n'a aucune raison d'être constituée de vecteurs du réseau. En revanche, le premier vecteur de la famille de Gram-Schmidt est toujours égal au premier vecteur de la base du réseau. La figure 4 l'illustre.

La famille de Gram-Schmidt est fortement reliée aux caractéristiques du réseau.

Proposition 2.3 (Volume d'un réseau)**.** *Soient* B *et* C *deux bases d'un réseau. On a* $\sqrt{\det({}^tBB)} = \sqrt{\det({}^tCC)}$ *: en effet, puisqu'on passe de* B *à* C *par une matrice unimodulaire* P*, on a*

$$\begin{aligned} \det({}^tCC) &= \det({}^t(BP)(BP)) \\ &= \det({}^tP\,{}^tBBP) \\ &= \det({}^tP)\det({}^tBB)\det(P) \\ &= \det({}^tBB) \quad . \end{aligned}$$

FIGURE 4 – Une base d'un réseau et la famille de Gram-Schmidt associée. Le vecteur b_2^* n'est pas un vecteur du réseau.



Cette quantité dépendant uniquement du réseau (et non de la base considérée) est appelée volume du réseau.

Soit B^* la famille de Gram-Schmidt associée à B. Alors on montre par le même genre de raisonnement que

$$\det({}^tBB) = \prod_{j=1}^{n} \left\| \overrightarrow{b_j^*} \right\|^2 \quad .$$

Proposition 2.4. *Soit B une base d'un réseau et B^* la famille de Gram-Schmidt associée. Si λ désigne la taille du plus court vecteur non nul du réseau, on a $\lambda \geq \min_i \left\| \overrightarrow{b_i^*} \right\|$.*

Démonstration. Voir annexe B. □

La proposition 2.4 amène la réflexion suivante : le seul vecteur de la famille de Gram-Schmidt sur lequel on ait un réel contrôle est le premier vecteur puisqu'il est égal au premier vecteur de la base du réseau. Pour imposer que le premier vecteur d'une base soit un plus court vecteur, il suffit donc de s'arranger pour que les vecteurs de la famille de Gram-Schmidt soient de normes croissantes. À défaut de pouvoir obtenir une monotonie parfaite des normes de la famille de Gram-Schmidt en temps polynomial, on peut au moins espérer imposer que les normes ne décroissent pas trop. En outre, si les vecteurs de Gram-Schmidt ne décroissent pas trop, il ne peuvent pas trop croître non plus à cause de la proposition 2.3. Par conséquent, une telle contrainte imposerait à la famille de Gram-Schmidt d'avoir des vecteurs de tailles relativement homogènes. Nous verrons que cette contrainte peut être réalisée à l'aide du *test de Lovász*.

Nous pouvons à présent dégager le principe de l'algorithme LLL :

1. étant donnée une base B du réseau, considérer la famille de Gram-Schmidt associée B^* ; changer la base B pour obtenir une base B' du réseau dont les vecteurs sont proches de ceux de B^*. Cette étape ne modifie pas la famille de Gram-Schmidt associée à la base du réseau ;
2. si un vecteur de la famille de Gram-Schmidt est trop petit par rapport au précédent, changer l'ordre des vecteurs de la base ; ceci change la famille de Gram-Schmidt associée et la rend plus proche des vecteurs du réseaux ;
3. si une permutation a été faite pendant l'étape (2), reprendre à l'étape (1).

À l'issue de l'algorithme, la base est à la fois assez proche de sa famille de Gram-Schmidt (grâce à l'étape 1) et formée de vecteurs assez courts (si l'étape 2 n'a pas donné lieu à une permutation, c'est que les vecteurs de Gram-Schmidt — et donc les vecteurs de la base — sont assez courts).

Plus formellement, l'étape 1 consiste à réduire faiblement la base B et l'étape 2 applique le test de Lovász aux vecteurs de la famille B^* :

Définition 2.3 (Réduction faible)**.** *Soit B une base d'un réseau et B^* la famille de Gram-Schmidt associée. B est dite* faiblement réduite *lorsque pour tout k et pour tout $j < k$,*

$$\left| \frac{\langle \overrightarrow{b_k}, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*}, \overrightarrow{b_j^*} \rangle} \right| \leq \frac{1}{2} \quad .$$

On dispose d'un algorithme simple pour transformer une base B en une base faiblement réduite B' de même famille de Gram-Schmidt (nous désignerons par $\lfloor x \rceil$ l'entier le plus proche de x ; si x est à égale distance entre deux entiers, on choisira le plus petit) :

Algorithme : WeaklyReduce
Data : une base B d'un réseau de rang p
Result : B' faiblement réduite
begin
  Construire la famille de Gram-Schmidt B^* associée à B ;
  $\overrightarrow{b'_1} = \overrightarrow{b_1}$;
  for *($k = 2$; $k \leq p$; k++)* **do**
    $\overrightarrow{t} = \overrightarrow{b_k}$;
    for *($j = k-1$; $j \geq 1$; j--)* **do**
      $\overrightarrow{t} = \overrightarrow{t} - \left\lfloor \frac{\langle \overrightarrow{t}, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*}, \overrightarrow{b_j^*} \rangle} \right\rceil \overrightarrow{b_j}$;
    end
    $\overrightarrow{b'_k} = \overrightarrow{t}$;
  end
end

Algorithme 2 : Réduction faible d'une base

Proposition 2.5 (Correction de l'algorithme de réduction faible)**.** *L'algorithme WeaklyReduce produit une base faiblement réduite.*

Démonstration. Voir annexe B. □

Définition 2.4 (Test de Lovász)**.** *Soit $\delta \in]1/4, 1[$. Soit B une base d'un réseau et B^* la famille de Gram-Schmidt associée. On dit que les vecteurs $\overrightarrow{b_i}$ et $\overrightarrow{b_{i+1}}$ satisfont le test de Lovász de paramètre δ si*

$$\left\| \overrightarrow{b_{i+1}^*} \right\|^2 \geq \left(\delta - \left(\frac{\langle \overrightarrow{b_{i+1}}, \overrightarrow{b_i^*} \rangle}{\langle \overrightarrow{b_i^*}, \overrightarrow{b_i^*} \rangle} \right)^2 \right) \left\| \overrightarrow{b_i^*} \right\|^2 \quad .$$

Si on utilise une base faiblement réduite et que tous les vecteurs de la base passent le test de Lovász (on dit alors que la base est LLL-réduite), on voit facilement que pour tout i, $\left\| \overrightarrow{b_{i+1}^*} \right\|^2 \geq (\delta - 1/4) \left\| \overrightarrow{b_i^*} \right\|^2$. C'est une contrainte de la forme désirée. En particulier, une récurrence évidente montre que pour tout i, $\left\| \overrightarrow{b_i^*} \right\|^2 \geq (\delta - 1/4)^{i-1} \left\| \overrightarrow{b_1^*} \right\|^2 \geq (\delta - 1/4)^{n-1} \left\| \overrightarrow{b_1^*} \right\|^2$ d'où il suit que

$$\lambda \geq \sqrt{(\delta - 1/4)^{n-1}} \left\| \overrightarrow{b_1} \right\| \quad .$$

Par exemple, pour $\delta = 3/4$ (qui est la valeur du paramètre courrament choisie pour exécuter LLL), le premier vecteur d'une base LLL est de norme inférieure à $2^{(n-1)/2}$ fois la norme du plus court vecteur non nul.

Cette majoration théorique est de bien mauvaise qualité par rapport aux excellents résultats pratiques de l'algorithme : il n'est pas rare que le plus court vecteur du réseau soit dans la base LLL.

L'algorithme LLL consiste donc à effectuer en boucle successivement une réduction faible, puis le test de Lovász sur chaque paire de vecteurs consécutifs de la base. Si le test échoue sur une paire $(i, i+1)$,

Algorithme : LLL

Data : une base B d'un réseau
Result : une base LLL-réduite

```
1  begin
2      while true do
3          B* = GramSchmidt(B) ;
4          B' = WeaklyReduce(B) ;
5          if ∀ i, b_i et b_{i+1} passent le test de Lovász then
6              goto 14 ;
7          else
8              i := indice du premier échec au test de Lovász ;
9              t := b_i ;
10             b_i := b_{i+1} ;
11             b_{i+1} := t ;
12         end
13     end
14     return B' ;
15 end
```

Algorithme 3 : Algorithme LLL

on échange les vecteurs $\overrightarrow{b_i}$ et $\overrightarrow{b_{i+1}}$ dans la base et on recommence. Ce qui n'est pas évident, c'est que ce processus termine. Sans rentrer dans les détails de la preuve (qui est assez technique et dont on pourra consulter les détails dans [6]) disons simplement que l'algorithme termine, et en temps polynomial qui plus est. Pour cela, on montre qu'à chaque fois que le test de Lovász échoue et qu'une permutation est réalisée, le produit des volumes fondamentaux des réseaux de base $(\overrightarrow{b_1})$, $(\overrightarrow{b_1}, \overrightarrow{b_2})$, $\dots$, $(\overrightarrow{b_1}, \overrightarrow{b_2}, \cdots, \overrightarrow{b_n})$ diminue d'un facteur au moins $1/\delta$.

2.4.1 Problème du plus proche vecteur

Le problème SVP admet assez naturellement une variante affine : il s'agit du problème du plus proche vecteur (généralement appelé CVP, de l'anglais *closest vector problem*). Étant donné un réseau de base B d'un espace vectoriel E, et un vecteur $\overrightarrow{v}$ de E, trouver un vecteur $\overrightarrow{t}$ dans le réseau minimisant $\|\overrightarrow{v} - \overrightarrow{t}\|$. Notons $\pi(\overrightarrow{v})$ la projection orthogonale de $\overrightarrow{v}$ sur l'espace vectoriel engendré par B. Le théorème de Pythagore indique que, pour $\overrightarrow{t} \in \mathrm{Vect}(B)$,

$$\|\overrightarrow{v} - \overrightarrow{t}\|^2 = \|\overrightarrow{v} - \pi(\overrightarrow{v})\|^2 + \|\pi(\overrightarrow{v}) - \overrightarrow{t}\|^2 \quad .$$

Par conséquent $\overrightarrow{t}$ minimise $\|\overrightarrow{v} - \overrightarrow{t}\|$ si et seulement s'il minimise aussi $\|\pi(\overrightarrow{v}) - \overrightarrow{t}\|$. On pourra donc supposer dans la suite que $\overrightarrow{v} \in \mathrm{Vect}(B)$.

Tout comme SVP (et pour les mêmes raisons), CVP admet au moins une solution, celle-ci n'étant pas nécessairement unique. Et de même qu'une base LLL-réduite nous donne une approximation du SVP, elle nous permet d'obtenir une approximation du CVP. On utilise un algorithme qui s'inspire fortement de l'algorithme de réduction faible ; on assure ainsi qu'en décomposant $\overrightarrow{v} - \overrightarrow{t}$ sur la famille B^*, les coefficients sont inférieurs à $1/2$ en valeur absolue.

On a alors

$$\|\overrightarrow{v} - \overrightarrow{t}\|^2 \leq \frac{1}{4} \sum_{j=1}^{n} \|\overrightarrow{b_j^*}\|^2 \leq \frac{1}{4} \sum_{j=1}^{n} \frac{1}{(\delta - 1/4)^{n-j}} \left\|\overrightarrow{b_n^*}\right\|^2$$

car la base est LLL-réduite. De là, en exprimant la somme d'une suite géométrique et en simplifiant, on obtient

$$\|\overrightarrow{v} - \overrightarrow{t}\|^2 \leq \frac{1}{4} (\delta - 1/4) \frac{(\delta - 1/4)^{-n} - 1}{5/4 - \delta} \left\|\overrightarrow{b_n^*}\right\|^2 \quad .$$

Finalement, on obtient la majoration

$$\|\overrightarrow{v} - \overrightarrow{t}\|^2 \leq \left(\frac{\delta - 1/4}{5/4 - \delta}\right) (\delta - 1/4)^{-n} \frac{\left\|\overrightarrow{b_n^*}\right\|^2}{4} \quad .$$

Algorithme : ApproximatedCVP
Data : une base LLL-réduite B ; un vecteur $\overrightarrow{v}$
Result : une approximation du CVP de $\overrightarrow{v}$
begin
  $\overrightarrow{t} = \overrightarrow{v}$;
  for *(*$j = n$; $j \geq 1$; j--*)* **do**
    $\overrightarrow{t} = \overrightarrow{t} - \left\lfloor \frac{\langle \overrightarrow{t}, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*}, \overrightarrow{b_j^*} \rangle} \right\rceil \overrightarrow{b_j}$;
  end
  return $\overrightarrow{v} - \overrightarrow{t}$;
end

Algorithme 4 : Approximation de CVP

Considérons une solution exacte $\overrightarrow{\tau}$ de CVP et supposons par souci de simlicité que les composantes suivant $\overrightarrow{b_n}$ de $\overrightarrow{\tau}$ et de $\overrightarrow{t}$ sont différentes (en cas d'égalité, on considérerait le plus grand indice s pour lequel les composantes sont différentes). Elles diffèrent d'une certaine valeur entière k. Alors, les composantes α_n et a_n de $\overrightarrow{v} - \overrightarrow{\tau}$ et de $\overrightarrow{v} - \overrightarrow{t}$ suivant $\overrightarrow{b_n^*}$ diffèrent aussi de k. Or $a_n \in [-\frac{1}{2}, \frac{1}{2}[$, donc $|\alpha_n| \geq 1/2$.

Il suit que $\|\overrightarrow{v} - \overrightarrow{\tau}\|^2 \geq \alpha_n^2 \left\|\overrightarrow{b_n^*}\right\|^2 \geq \frac{\left\|\overrightarrow{b_n^*}\right\|^2}{4}$. De là, on déduit finalement que

$$\|\overrightarrow{v} - \overrightarrow{t}\|^2 \leq \left(\frac{\delta - 1/4}{5/4 - \delta}\right) (\delta - 1/4)^{-n} \|\overrightarrow{v} - \overrightarrow{\tau}\|^2 \quad .$$

Par exemple, pour $\delta = 3/4$, l'approximation obtenue est au pire $2^{n/2}$ fois plus grande que l'optimal. Là encore, les résultats pratiques sont de bien meilleure qualité que la majoration théorique.

Comme la base LLL est formée de vecteurs assez courts et assez orthogonaux, on peut en outre rayonner efficacement autour de $\overrightarrow{t}$ pour trouver d'autres vecteurs proches de $\overrightarrow{v}$. Ainsi les $\overrightarrow{t} \pm \overrightarrow{b_i}$ sont d'autres bonnes approximations du CVP. Cette remarque a une importance pratique : par la suite, nous utiliserons les algorithmes LLL et ApproximatedCVP pour résoudre CVP de manière approchée. Ceci nous contraint à utiliser la norme euclidienne puisque c'est elle qui sous-tend ces algorithmes. Mais en pratique, ce n'est pas la norme euclidienne qui nous intéressera et nous cherchons en fait à résoudre CVP pour une autre norme N. Puisque nous sommes en dimension finie, toutes les normes sont équivalentes, c'est-à-dire qu'il existe deux réels positifs α et β tels que pour tout vecteur $\overrightarrow{x}$, $\alpha\|\overrightarrow{x}\| \leq N(\overrightarrow{x}) \leq \beta\|\overrightarrow{x}\|$. En pratique, cela signifie que si α et β sont suffisament proches de 1 on a des chances de trouver le CVP pour N dans le voisinage proche du CVP de la norme euclidienne ; on voit donc l'intérêt de posséder une base formée de vecteurs courts et presque orthogonaux.

3 Présentation de notre méthode

3.1 Résolution approchée de systèmes d'équations linéaires

Comme nous l'avons mentionné, les réseaux sont une structure très pratique et forment un bon cadre pour la modélisation de nombreux problèmes. Nous allons décrire à présent un procédé qui utilise la théorie des réseaux pour résoudre des systèmes d'équations linéaires de façon approchée. Ce procédé sera au cœur de notre méthode ; c'est pourquoi nous allons prendre le temps de le détailler précisément.

Nous nous intéressons à la résolution d'un système linéaire $AX = Y$ où A est une matrice et Y un vecteur colonne. Pour ne pas compliquer inutilement la discussion, nous supposerons que A est carrée

inversible de taille p mais on pourrait tenir le même genre de raisonnements pour une matrice quelconque dont les colonnes forment une famille libre.

Lorsqu'on est amené à résoudre de tels systèmes, généralement A est une matrice représentant un opérateur linéaire entre un espace de paramètres (où X prend ses valeurs) et un espace de quantités observables (où Y prend ses valeurs). On cherche les valeurs qu'il faut donner aux paramètres X pour obtenir, via l'opérateur linéaire A, l'effet désiré Y. C'est ce qui amène à poser et à résoudre le système $AX = Y$.

Cependant, dans la réalité, on n'a pas la liberté de fixer les valeurs des paramètres dans un ensemble continu : X ne peut prendre que des valeurs discrètes et la véritable question n'est pas de trouver X tel que $AX = Y$ (ce qui sera sans doute impossible) mais plutôt de trouver X de telle sorte que AX soit le plus *proche* possible de Y, sachant que X ne prend que des valeurs discrètes. La notion de *proximité* entre AX et Y peut se définir formellement en utilisant une norme d'espace vectoriel. En pratique, le choix d'une norme particulière plutôt qu'une autre dépendra de ce qu'on cherche effectivement à minimiser : veut-on s'assurer que tous les coefficients de AX sont proches de Y ou bien seulement que la moyenne des écarts entre les coefficients de AX et de Y est faible, par exemple ?

Supposons pour modéliser le problème que X prenne ses valeurs dans $\mathbb{Z}^p$. On se donne une norme N et notre problème est désormais le suivant :

$$\text{trouver } X \in \mathbb{Z}^p \text{ qui minimise } N(Y - AX).$$

Une première idée pourrait être la suivante : calculer la solution réelle $\overline{X}$ du système, puis arrondir cette solution aux entiers les plus proches. Cette méthode, quoique naturelle, peut être très mauvaise. Notons $\widehat{X}$ la solution arrondie et $V = \overline{X} - \widehat{X}$ l'erreur d'arrondi. On a $A\widehat{X} = A(\overline{X} - V) = Y - AV$. Par conséquent, l'erreur que cette méthode produit vis-à-vis de Y est égale à AV. Or, l'opérateur A peut amplifier considérablement l'erreur V. Peut-être qu'on aurait pu arrondir $\overline{X}$ moins précisément mais dans une direction V' telle que $N(AV') \ll N(AV)$. La valeur approchée de Y ainsi obtenue serait de bien meilleure qualité.

Mais à bien y regarder, le problème que nous nous posons n'est rien d'autre que la recherche d'un vecteur proche de Y dans un réseau : c'est ce que nous allons détailler à présent. Pour cela, voyons A comme la donnée de p vecteurs colonnes de $\mathbb{R}^p$: $(\overrightarrow{a_1}, \cdots, \overrightarrow{a_p})$. De même Y est l'expression dans la base canonique de $\mathbb{R}^p$ d'un vecteur colonne $\overrightarrow{y}$. Notons $X = {}^t(x_1, \cdots, x_p)$. On peut réécrire AX sous la forme

$$AX = \begin{pmatrix} \\ \overrightarrow{a_1}, \cdots, \overrightarrow{a_p} \\ \\ \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} = x_1\overrightarrow{a_1} + x_2\overrightarrow{a_2} + \cdots + x_p\overrightarrow{a_p}$$

Si les x_i sont entiers, il s'agit de l'expression d'un certain vecteur dans le réseau de base $(\overrightarrow{a_1}, \cdots, \overrightarrow{a_p})$. Notons Γ ce réseau. Notre problème se reformule donc sous la forme suivante : trouver un vecteur $\overrightarrow{v}$ de Γ tel que $N(\overrightarrow{y} - \overrightarrow{v})$ soit minimal. Autrement dit : trouver dans Γ un plus proche vecteur $\overrightarrow{v}$ de $\overrightarrow{y}$ relativement à la norme N. Il s'agit donc bien d'une instance de CVP.

Il faut bien remarquer que ce qui nous intéresse réellement n'est pas tant le vecteur $\overrightarrow{v}$ mais plutôt ses coordonnées $(x_1, \cdots, x_p)$ dans la base $(\overrightarrow{a_1}, \cdots, \overrightarrow{a_p})$ de Γ.

À ce point, le lecteur est peut-être un peu confus : dans l'espace vectoriel que nous considérons, il y a plusieurs bases :

- la base canonique de $\mathbb{R}^p$: c'est dans cette base qu'on écrit les vecteurs $\overrightarrow{a_1}, \cdots, \overrightarrow{a_p}$ ainsi que $\overrightarrow{y}$. C'est la base ambiante dans laquelle s'effectuent tous les calculs.
- la base $(\overrightarrow{a_1}, \cdots, \overrightarrow{a_p})$: il s'agit de la base du réseau Γ. C'est elle qui nous intéresse réellement car ce sont les coordonnées de $\overrightarrow{v}$ dans cette base qui fournissent la solution $(x_1, \cdots, x_p)$ à notre problème.
- par la suite, nous allons réduire cette base avec l'algorithme LLL en une autre base du réseau $(\overrightarrow{a'_1}, \cdots, \overrightarrow{a'_p})$: cette base sera bien adaptée pour se promener dans le réseau et pour résoudre CVP de manière approchée. Naturellement, $\overrightarrow{v}$ nous sera donné par son expression dans cette base $\overrightarrow{v} = x'_1\overrightarrow{a'_1} + x'_2\overrightarrow{a'_2} + \cdots + x'_p\overrightarrow{a'_p}$ mais cette expression n'aura aucun intérêt pour nous : il faudra que nous la retraduisions en une expression dans la base $(\overrightarrow{a_1}, \cdots, \overrightarrow{a_p})$.

Il n'y a pas d'algorithme connu pour résoudre efficacement (et même de manière approchée) le CVP en temps raisonnable pour une norme quelconque N. Cependant, comme nous l'avons vu page 15, nous

pouvons utiliser l'algorithme ApproximatedCVP qui fonctionne avec la norme euclidienne : il ne nous donnera sans doute pas un vecteur optimal pour la norme N mais ce vecteur devrait tout de même être assez proche de $\overrightarrow{y}$ en norme N. Nous avons vu en outre qu'on peut utiliser les vecteurs de la base LLL-réduite pour explorer le voisinage immédiat de $\overrightarrow{v}$ à la recherche de vecteurs plus proches de $\overrightarrow{y}$ en norme N.

En pratique, donc, on commence par calculer une base LLL-réduite $A' = (\overrightarrow{a'_1}, \cdots, \overrightarrow{a'_p})$. Comme nous l'avons vu, nous aurons besoin de traduire des expressions de la base A' dans la base A : il nous faut donc l'expression de chaque vecteur de A' dans la base A :

$$\left\{\begin{array}{rcl} \overrightarrow{a'_1} & = & m_{11}\overrightarrow{a_1} + m_{21}\overrightarrow{a_2} + \cdots + m_{p1}\overrightarrow{a_p} \\ \overrightarrow{a'_2} & = & m_{12}\overrightarrow{a_1} + m_{22}\overrightarrow{a_2} + \cdots + m_{p2}\overrightarrow{a_p} \\ & \vdots & \\ \overrightarrow{a'_p} & = & m_{1p}\overrightarrow{a_1} + m_{2p}\overrightarrow{a_2} + \cdots + m_{pp}\overrightarrow{a_p} \end{array}\right.$$

En notant M la matrice $(m_{ij})_{i,\,j\in[\![1,\,p]\!]}$, on peut réécrire ce système sous forme matricielle : $A' = AM$. Cette matrice de passage nous permet de traduire facilement une expression de la base A' dans la base A : si $\overrightarrow{v} = x'_1\overrightarrow{a'_1} + x'_2\overrightarrow{a'_2} + \cdots + x'_p\overrightarrow{a'_p}$, on peut écrire

$$V = A' \begin{pmatrix} x'_1 \\ \vdots \\ x'_p \end{pmatrix}$$

d'où il vient $V = (AM)X' = A(MX')$. Il suit que $X = MX'$: une simple multiplication matricielle nous permet de retrouver l'expression de $\overrightarrow{v}$ dans la base A. On remarquera que puisque M est la matrice de passage entre deux bases d'un réseau, c'est une matrice entière et la multiplication MX' est faisable en arithmétique entière, donc sans erreurs d'arrondi. Les coefficients m_{ij} sont tous calculés pendant l'exécution de l'algorithme LLL et on peut donc fournir la matrice M sans aucun surcoût.

Après avoir calculé la base réduite A', on l'utilise avec l'algorithme ApproximatedCVP pour produire une bonne approximation $\overrightarrow{v}$ du plus proche vecteur de $\overrightarrow{y}$ en norme euclidienne. Là aussi, ce qui nous intéresse n'est pas l'expression de $\overrightarrow{y}$ dans la base canonique de $\mathbb{R}^n$ mais son expression X' dans la base A'. Là encore, il n'est pas difficile d'adapter ApproximatedCVP pour fournir le vecteur X' sans surcoût.

Finalement, on renvoie le vecteur $X = MX'$. Si on veut utiliser la base réduite pour explorer le voisinage immédiat du vecteur $\overrightarrow{v}$ il faut exprimer par exemple $\overrightarrow{v} + \overrightarrow{a'_1}$ dans la base A. On a $V = A(MX')$ et $A'_1 = AM_1$ où le 1 en indice indique qu'on prend la première colonne de la matrice. On en déduit que $V = A(MX' + M_1)$: l'expression dans la base A est donc $(MX' + M_1)$.

3.2 Première approche

À présent que nous comprenons l'intérêt et l'efficacité de la réduction des réseaux, voyons comment nous pouvons appliquer cette théorie au problème qui nous occupe. On se donne donc $n+1$ entiers relatifs $m_0, \cdots, m_n$ et une fonction f continue sur un intervalle $[a, b]$. Nous essayons de trouver $n+1$ entiers relatifs $a_0, \cdots, a_n$ tels que $\|f - p\|_\infty$ soit le plus petit possible, où

$$p = \sum_{i=0}^{n} \frac{a_i}{2^{m_i}} X^i \quad . \qquad (*)$$

Contrairement à l'approche certifiée proposée dans [3] qui fournit la liste de tous les polynômes optimaux, nous ne cherchons pas forcément le meilleur polynôme. On souhaite bien sûr obtenir un polynôme qui approche f du mieux possible, mais en insistant sur la rapidité de l'algorithme permettant d'obtenir un tel polynôme.

Basiquement, l'idée de notre méthode est d'utiliser l'interpolation pour approcher la fonction. Comme nous l'avons vu, interpoler revient à résoudre un système linéaire. Grâce aux réseaux, nous pouvons trouver une solution approchée à coefficients entiers à ce système.

Plus formellement, choisissons $n+1$ points $x_0, \cdots, x_n$ dans l'intervalle $[a, b]$. Si nous cherchions le polynôme réel interpolant exactement f aux points x_k et se présentant sous la forme

$$\sum_{i=0}^{n} \frac{a_i}{2^{m_i}} X^i \quad (a_i \in \mathbb{R}) \quad ,$$

nous aurions à résoudre le système suivant :

$$\begin{pmatrix} \frac{1}{2^{m_0}} & \frac{x_0}{2^{m_1}} & \frac{x_0^2}{2^{m_2}} & \cdots & \frac{x_0^n}{2^{m_n}} \\ \frac{1}{2^{m_0}} & \frac{x_1}{2^{m_1}} & \frac{x_1^2}{2^{m_2}} & \cdots & \frac{x_1^n}{2^{m_n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{2^{m_0}} & \frac{x_n}{2^{m_1}} & \frac{x_n^2}{2^{m_2}} & \cdots & \frac{x_n^n}{2^{m_n}} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix} \quad . \tag{S}$$

En réalité, nous ne cherchons pas le polynôme réel interpolant exactement f : nous cherchons un polynôme pour lequel les a_k sont entiers, et qui interpole presque f aux points x_k. Il s'agit donc de résoudre de façon approchée un système linéaire à l'aide d'un vecteur à coefficients entiers. C'est le problème que nous avons décrit un peu plus haut. Nous le résolvons en utilisant la méthode que nous avons déjà détaillée.

Afin de mieux comprendre le fonctionnement de la méthode, nous allons traiter un exemple complet. Mais avant cela, arrêtons-nous sur une remarque importante.

Nous n'avons pas précisé comment choisir les points d'interpolation. Comme nous l'avons vu, lorsqu'on utilise l'interpolation, le choix de mauvais points peut conduire à des résultats désastreux (c'est le phénomène de Runge, décrit en annexe A). Cependant, nous avons remarqué qu'en choisissant les points adaptés, l'interpolation pouvait fournir exactement le polynôme de meilleure approximation.

Il est naturel que ce phénomène se reproduise lorsqu'on résoud le système d'interpolation de façon approchée. De fait, considèrons la fonction où se manifeste les phénomènes de Runge ($f : x \mapsto 1/(1 + 25x^2)$). Posons tous les $m_k = 50$ et cherchons un polynôme de degré 10 ayant la forme désirée et approchant f sur l'intervalle $[-1, 1]$. Si on utilise 11 points équirépartis, le polynôme p obtenu est de très mauvaise qualité comme on peut le voir sur la figure 6 en annexe A.

Tout comme dans le cas de l'interpolation exacte, le phénomène ne fait que s'agraver si on augmente le degré et le nombre de points. Par exemple, pour le degré 20, $\|f - p\|_\infty$ est de l'ordre de 60 alors que si on arrondit les coefficients du polynôme fourni par l'algorithme de Remes, on obtient un polynôme q tel que $\|f - q\|_\infty \simeq 9 \cdot 10^{-3}$.

3.3 Choix des points

Pour résoudre ce problème, on s'appuie sur la constatation que nous avions faite dans le cas de l'interpolation exacte : d'après le corollaire 2.1, il existe $n+1$ points pour lesquels le polynôme d'interpolation exact coïncide avec le polynôme de meilleure approximation. Cette remarque s'appuie sur le théorème de Tchebychev qui affirme que le polynôme de meilleure approximation oscille au moins $n+2$ fois autour de la fonction. Or le polynôme optimal à coefficients de la forme $\mathbb{Z}/2^{m_i}$ n'oscille pas nécessairement autour de la fonction.

Néanmoins, lorsque les m_i sont suffisamment grands, le polynôme optimal à coefficients de la forme $\mathbb{Z}/2^{m_i}$ n'est sans doute pas très éloigné du polynôme de meilleure approximation. C'est pourquoi nous avons adopté la stratégie suivante : nous calculons d'abord le polynôme de meilleure approximation réel, puis nous cherchons les points x_i où il croise la fonction.

Lorsqu'on choisit ces points la solution réelle du système (S) est le polynôme de meilleure approximation. En particulier, si le polynôme de meilleure approximation a des coefficients de la forme $\mathbb{Z}/2^{m_i}$, la résolution du système en nombres entiers nous fournira bien le polynôme optimal. C'est donc ces points là que nous choisissons pour poser le système.

Effectivement, les résultats obtenus en choisissant ces points sont excellents. Dans les quelques cas où nous connaissions explicitement le polynôme optimal, nous avons pu constaté que notre algorithme le trouvait souvent. Lorsque ce n'est pas le cas, le polynôme optimal s'écrit très simplement dans la base réduite et on peut l'obtenir en visitant avec la base réduite le voisinage du polynôme renvoyé par notre algorithme. Dans les cas où le polynôme optimal ne peut pas être calculé en temps raisonnable par l'algorithme décrit dans [3], notre algorithme fournit un polynôme d'une qualité si proche de celle du polynôme de meilleure approximation qu'on peut parfaitement se contenter de cette approximation dans les applications pratiques : la recherche du polynôme réellement optimal n'apporterait aucune amélioration tangible.

Algorithme : GoodPoly
Data : une fonction f ; un intervalle $[a,\, b]$; un entier n
begin
  Calculer $p^* = \text{Remes}(f, [a,\, b],\, n))$;
  Trouver les $n+1$ solutions x_i de l'équation $p^*(x) = f(x)$;
  Construire la matrice $A = (x_{i-1}^{j-1}/2^{m_{j-1}})_{i,\, j \in [\![1,\, n]\!]}$;
  Calculer $A' = \text{LLL}(A)$ et la matrice de passage M : $A' = AM$;
  Construire le vecteur $Y = (f(x_{i-1}))_{i \in [\![1,\, n]\!]}$;
  Calculer ApproximatedCVP(A', Y) ; noter V ses coordonnées dans A';
  return MV ;
  (le cas échéant, renvoyer d'autres vecteurs proches $MV \pm M_j$) ;
end

Algorithme 5 : Schéma de notre algorithme

Il faut bien remarquer que le choix des points x_i conditionne à la fois le système (S) (puisque la matrice est définie par les x_i) mais aussi la norme euclidienne qui sera utilisée par LLL : en effet, la norme utilisée par LLL est relative à la base canonique ambiante de $\mathbb{R}^n$: si Z est un vecteur colonne, sa norme euclidienne est $\sqrt{z_1^2 + z_2^2 + \cdots + z_n^2}$. Lorsqu'on utilise LLL pour résoudre approximativement le CVP euclidien, on cherche à minimiser $\|A\alpha - Y\|$: en notant $A_0, \cdots, A_n$ les colonnes de A et $a_0, \cdots, a_n$ les coefficients de α, cette expression peut se réécrire : minimiser $\|a_0 A_0 + \cdots + a_n A_n - Y\|$. Dans le cas où la matrice A est celle du système (S), le j-ème vecteur colonne de A représente le monôme $X^{j-1}/2^{m_{j-1}}$ évalué aux points $x_0, \cdots, x_n$. En notant

$$P(X) = \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} X + \cdots + \frac{a_n}{2^{m_n}} X^n \quad ,$$

nous pouvons réécrire le problème sous la forme

$$\text{minimiser} \quad \left\| \begin{pmatrix} P(x_0) \\ \vdots \\ P(x_n) \end{pmatrix} - Y \right\| \quad .$$

Finalement, en prenant le Y du système (S), il s'agit de minimiser

$$\sqrt{(P(x_0) - f(x_0))^2 + \cdots + (P(x_n) - f(x_n))^2} \quad .$$

Autrement dit, l'algorithme LLL va chercher un polynôme qui minimise les moindres carrés pris aux points x_i. Ce problème est *a priori* bien distinct de celui de minimiser $\|f - P\|_\infty$ sur tout l'intervalle ; mais en choisissant les x_i où f croise p^*, nous pouvons en première approximation confondre la méthode d'interpolation et de Remes et le remplacement de la norme L_∞ par les moindres carrés est sans gravité.

On peut craindre qu'à force d'approximations successives, on dénature le problème au point que la solution approchée obtenue n'ait plus rien à voir avec le problème initial. En effet, on procède à plusieurs approximations : tout d'abord on discrétise le problème aux points x_i, ensuite, on utilise la norme euclidienne qui ne correspond pas exactement à la minimisation en norme L_∞ qu'on recherche ; enfin, pour rester en temps polynomial, on se contente d'une solution approchée au problème du plus proche vecteur. Néanmoins, la méthode est extrêmement efficace en pratique.

Prenons un exemple : on cherche à approcher cos sur l'intervalle $[0,\ \pi/4]$ par un polynôme de degré 3 de la forme

$$\frac{a_0}{4096} + \frac{a_1}{1024}X + \frac{a_2}{64}X^2 + \frac{a_3}{16}X^3 \quad .$$

En choisissant 4 points équirépartis dans l'intervalle, notre algorithme renvoie le polynôme optimal.

Parfois, l'algorithme trouve un vecteur V qui n'est pas exactement l'optimal, mais ce dernier s'écrit sous la forme $V + \sum \mu_j A'_j$ avec de très petites valeurs de μ_j. Par exemple, pour exp sur l'intervalle $[0,\ 1/2]$ le polynôme optimal de degré 3 de la forme

$$\frac{a_0}{32768} + \frac{a_1}{16384}X + \frac{a_2}{4096}X^2 + \frac{a_3}{1024}X^3$$

s'écrit $V + A'_1 + A'_3$ en choisissant des points équirépartis. En prenant les points d'intersection entre le polynôme fourni par Remes et la fonction, l'expression du polynôme optimal est encore plus simple : $V + A'_4$.

3.4 Exemple d'utilisation

Nous allons à présent traiter un exemple complet qui montrera comment on règle quelques petits problèmes techniques en pratique. Cet exemple, tiré d'une situation réelle, manipule des polynômes de petit degré, ce qui va nous permettre de le détailler complètement. Avec un petit degré tel que celui-ci, nous ne pourrons évidemment pas gagner beaucoup de précision ; nous développons ce petit exemple uniquement pour familiariser le lecteur avec les possibilités de notre méthode. Nous citerons ensuite quelques cas où notre méthode a permis d'améliorer notablement ce qui existait.

Nous allons nous intéresser à la fonction exp sur l'intervalle

$$I = \left[-\frac{1+2^{-18}}{2^{13}}\ln(2),\ \frac{1+2^{-18}}{2^{13}}\ln(2)\right] \simeq [-0.000085\,;\,0.000085] \quad .$$

Nous avons beaucoup travaillé en collaboration avec les développeurs de la librairie `crlibm`, en particulier C. Lauter. Cette librairie (disponible à l'adresse `http://lipforge.ens-lyon.fr/www/crlibm/`) a pour but de fournir une implémentation efficace [§] des fonctions élémentaires avec arrondi correct au format `double` IEEE-754 : il faut implémenter chaque fonction élementaire (exp, cos, sin, arccos, etc.) de telle sorte que lorsque l'utilisateur demande le calcul de $f(x)$, le résultat soit celui que l'on obtiendrait si on calculait la valeur exacte de $f(x)$ et qu'on l'arrondissait ensuite à 53 bits de mantisse.

L'exemple que nous allons traiter ici correspond à un vrai sous-cas de l'algorithme de calcul de la fonction exp dans `crlibm`. C. Lauter nous a indiqué que pour des raisons d'efficacité, il cherche à approcher exp par un polynôme de la forme

$$1 + X + X^2/2 + (a_{0h} + a_{0\ell})X^3 + (a_{1h} + a_{1\ell})X^4 + a_2X^5 + a_3X^6 + a_4X^7$$

où a_{0h}, $a_{0\ell}$, a_{1h}, $a_{1\ell}$, a_2, a_3 et a_4 sont des `doubles` IEEE-754 (c'est-à-dire qu'ils ont 53 bits de mantisse).

Nous observons là deux contraintes que nous n'avons pas modélisées : tout d'abord, certains coefficients du polynôme sont fixés par avance ; ensuite, certains coefficients sont stockés sous la forme d'une somme non évaluée de deux nombres flottants. Il s'agit d'un format utilisé en interne de la librairie `crlibm` et qui permet de simuler une grande précision. Ce format porte le nom de double-`double` ; de même, on peut avoir besoin d'utiliser le format triple-`double`. Pour ces coefficients, nous avons donc *a priori* deux inconnues à gérer.

Pour résoudre le premier problème, nous utilisons une petite astuce : nous allons poser $m_0 = m_1 = 0$ et $m_2 = 1$. Ainsi, nous recherchons un polynôme qui approche exp sur un petit intervalle centré en zéro, dont les deux premiers coefficients sont des entiers et le troisième coefficient est un multiple de $1/2$. Cette contrainte est tellement forte que le polynôme n'aura d'autre de choix que de commencer par $1 + X + X^2/2$.

Le second problème est plus délicat. Pour le résoudre, nous remplaçons la recherche de double-`doubles` par la recherche d'un nombre flottant de $2 \times 53 = 106$ bits de mantisses. En effet, un nombre flottant de $2k$ bits de mantisse peut toujours se représenter par une somme de deux nombres flottants de k bits de

§. c'est-à-dire avec des temps moyens d'utilisation de la librairie du même ordre de grandeur que ceux des librairies mathématiques usuelles, qui ne fournissent pas l'arrondi correct.

mantisse de la façon suivante : soit x un flottant de $2k$ bits dont la mantisse est l'entier μ ; on note, μ_i les β-chiffres de μ. Alors x s'écrit comme la somme $x_h + x_\ell$ suivante :

$$x = s \cdot \underbrace{\mu_{2k-1}\mu_{2k}\dots\mu_k}_{h}\underbrace{\mu_{k-1}\dots\mu_1\mu_0}_{\ell} \cdot \beta^e = \underbrace{(s \cdot h \cdot \beta^{e+k})}_{x_h} + \underbrace{(s \cdot \ell \cdot \beta^e)}_{x_\ell} \quad .$$

Ainsi, en cherchant les coefficients sous la forme de flottants de 106 bits est-on sûr d'obtenir des double-`doubles`. Le défaut de la méthode vient du fait que la réciproque n'est pas vraie : il existe des double-`doubles` non représentables sur un format de 106 bits. Par exemple, le nombre x suivant est représentable par un double-`double` mais nécessite 116 bits de mantisse :

$$x = \underbrace{\mu_{2k-1}\mu_{2k}\dots\mu_k}_{h}\underbrace{0000000000}_{10\text{ zéros}}\underbrace{1\mu_{k-2}\dots\mu_1\mu_0}_{\ell} \cdot \beta^e = \underbrace{(h \cdot \beta^{e+k+10})}_{x_h} + \underbrace{(\ell \cdot \beta^e)}_{x_\ell} \quad .$$

Notre méthode peut donc rater des coefficients qui seraient meilleurs que ceux qu'on peut obtenir avec 106 bits de mantisse. Nous disposons d'une piste pour régler automatiquement ce petit problème mais elle est encore largement expérimentale et nous ne la présenterons pas ici.

À ce stade, nous avons ramené le problème à la recherche d'un polynôme approchant la fonction exp et de la forme $c_0 + c_1X + c_2X^2 + c_3X^3 + c_4X^4 + c_5X^5 + c_6X^6 + c_7X^7$ où c_0 et c_1 sont des entiers, c_2 est un demi-entier, c_3 et c_4 sont des flottants de 106 bits et c_5, c_6 et c_7 sont des flottants de 53 bits. Pour se ramener au cas d'un nombre de la forme $\mathbb{Z}/2^{m_i}$ il suffit de connaître par avance l'ordre de grandeur des c_i. En effet, si on connait l'ordre de grandeur de c_i, on peut déterminer quel sera son exposant e, et par conséquent, on remplacera la recherche de c_i par la recherche d'un entier (correspondant à la mantisse de c_i) divisé par 2^{-e}. L'expérience montre que la plupart du temps, les coefficients d'un polynôme optimal à coefficients de la forme $\mathbb{Z}/2^{m_i}$ sont du même ordre de grandeur que ceux du polynôme de meilleure approximation réel. Par conséquent, il suffit de regarder l'ordre de grandeur des coefficients fournis par l'algorithme de Remes pour connaître e.

Évidemment, on n'est pas à l'abri d'une mauvaise surprise et le polynôme p fourni par notre méthode peut finalement avoir des coefficients d'un ordre de grandeur différent. Dans ce cas, les coefficients de p ne sont pas tout à fait représentables dans le format flottant demandé. Ce genre de situation se produit lorsque la contrainte qu'on impose en demandant des coefficients de la forme $\mathbb{Z}/2^{m_i}$ est trop forte : dans ce cas, pour optimiser l'erreur, le polynôme p s'éloigne du polynôme de meilleure approximation et les coefficients des deux polynômes peuvent ne pas être du même ordre de grandeur. Mais sans doute le polynôme p donne-t-il alors une meilleure estimation des coefficients du polynôme optimal. On peut donc relancer notre méthode en utilisant l'ordre de grandeur fourni par le polynôme p. Et ainsi de suite si cela ne convient toujours pas. Nous n'avons pas de preuve que ce procédé finit par s'arrêter ; néanmoins la pratique montre qu'en une ou deux étapes, on se stabilise finalement sur des coefficients représentables dans le format flottant désiré.

Dans l'exemple qui nous intéresse, l'algorithme de Remes fournit un polynôme P^* tel que $\| \exp - P^* \|_\infty \simeq 5.0906 \cdot 10^{-40}$. En tronquant naïvement ce polynôme, on obtient une erreur d'environ $1.6790 \cdot 10^{-38}$. Il n'y a pas lieu d'être surpris de cette importante perte de qualité puisqu'on applique une contrainte très forte sur les trois premiers coefficients. C. Lauter a contourné le problème comme il a pu et a réussi à obtenir pour `crlibm` un polynôme satisfaisant les contraintes et donnant une erreur d'environ $4.5738 \cdot 10^{-39}$ ce qui est déjà bien mieux. On peut montrer que le meilleur polynôme réel de degré 7 commençant par $1 + X + X^2/2$ réalise une erreur d'environ $1.2336 \cdot 10^{-39}$. Nous allons voir que notre méthode fournit un polynôme satisfaisant les contraintes et atteignant quasiment cet optimal à coefficients réels.

Pour cela, on calcule donc les solutions de l'équation $\exp(x) = P^*(x)$ et on trouve les 8 solutions suivantes :

$$\begin{array}{lllllllll} x_0 & \simeq & -8.29872 \cdot 10^{-5} & ; & x_1 & \simeq & -7.03531 \cdot 10^{-5} & ; & x_2 & \simeq & -4.70084 \cdot 10^{-5} \\ x_3 & \simeq & -1.65071 \cdot 10^{-5} & ; & x_4 & \simeq & 1.65073 \cdot 10^{-5} & ; & x_5 & \simeq & 4.70085 \cdot 10^{-5} \\ x_6 & \simeq & 7.03532 \cdot 10^{-5} & ; & x_7 & \simeq & 8.29872 \cdot 10^{-5} \, . & & & & \end{array}$$

En outre, les exposants des coefficients de P^* sont -108, -110, -59, -62 et -65. On pose donc le système (S) en utilisant les x_i ci-dessus et en prenant les m_i égaux aux opposés des exposants ci-dessus ; on construit le vecteur $\overrightarrow{y}$ en évaluant exp aux points x_i. On calcule alors une base LLL-réduite puis on résout le CVP en prenant soin de garder trace de la matrice de passage à chaque étape. Enfin, on affiche

les polynômes correspondants au vecteur $\overrightarrow{v}$ fourni par ApproximatedCVP ainsi que ceux correspondants aux vecteurs $\overrightarrow{v} + \overrightarrow{a_j'}$ et $\overrightarrow{v} - \overrightarrow{a_j'}$ (pour $j = 0 \cdots 7$).

Si on note $(\overrightarrow{a_0^*}, \cdots, \overrightarrow{a_7^*})$ la famille de Gram-Schmidt associée à la base LLL-réduite, on observe que

$$\begin{cases} \|\overrightarrow{a_0^*}\| & \simeq & 5.8401 \cdot 10^{-50} \\ \|\overrightarrow{a_1^*}\| & \simeq & 1.0895 \cdot 10^{-48} \\ \|\overrightarrow{a_2^*}\| & \simeq & 8.0735 \cdot 10^{-46} \\ \|\overrightarrow{a_3^*}\| & \simeq & 1.4414 \cdot 10^{-44} \\ \|\overrightarrow{a_4^*}\| & \simeq & 2.4054 \cdot 10^{-40} \\ \|\overrightarrow{a_5^*}\| & \simeq & 5.5236 \cdot 10^{-10} \\ \|\overrightarrow{a_6^*}\| & \simeq & 9.2320 \cdot 10^{-6} \\ \|\overrightarrow{a_7^*}\| & \simeq & 5.3452 \cdot 10^{-1}. \end{cases}$$

Leurs normes sont donc croissantes ; comme le plus court vecteur du réseau est de norme supérieure à $\min \|\overrightarrow{a_i^*}\|$ (proposition 2.4) et que $\overrightarrow{a_0^*} = \overrightarrow{a_0'}$ est un vecteur du réseau, on en déduit que $\overrightarrow{a_0'}$ est un plus court vecteur du réseau pour la norme euclidienne.

De même, les normes euclidiennes de $(\overrightarrow{v} - \overrightarrow{y})$, $(\overrightarrow{v} \pm \overrightarrow{a_0^*} - \overrightarrow{y})$, $(\overrightarrow{v} \pm \overrightarrow{a_1^*} - \overrightarrow{y})$, $(\overrightarrow{v} \pm \overrightarrow{a_2^*} - \overrightarrow{y})$ et $(\overrightarrow{v} \pm \overrightarrow{a_3^*} - \overrightarrow{y})$ sont toutes comprises entre $1.962225628 \cdot 10^{-39}$ et $1.962225630 \cdot 10^{-39}$ et $\|\overrightarrow{v} - \overrightarrow{y}\|_2$ est la plus petite d'entre elles. On peut légitimement penser que $\overrightarrow{v}$ est bien un plus proche vecteur de $\overrightarrow{y}$ en norme euclidienne. Nous voyons donc bien sur cet exemple la très grande qualité pratique de l'algorithme LLL.

Si on regarde les polynômes correspondant à ces quelques vecteurs, et qu'on observe les valeurs de $\|f - p\|_\infty$ correspondantes, on observe le même phénomène : les normes L_∞ sont comprises entre $1.3026286 \cdot 10^{-39}$ et $1.3026346 \cdot 10^{-39}$. Mais c'est le polynôme qui correspond à $\overrightarrow{v} - \overrightarrow{a_3'}$ et non pas celui correspondant à $\overrightarrow{v}$ qui est le meilleur. Cela correspond à la petite différence qu'il y a entre la minimisation en norme L_∞ dans l'espace $\mathbb{R}_n[X]$ et en norme euclidienne dans l'espace discrétisé. D'ailleurs, cette différence est mineure puisque les erreurs ne diffèrent qu'à partir de la cinquième décimale.

Finalement, on obtient donc un polynôme satisfaisant les contraintes et dont l'erreur en norme L_∞ vis-à-vis de la fonction est environ $1.30263 \cdot 10^{-39}$. Notre méthode a donc permis d'améliorer la précision du polynôme de `crlibm` d'un facteur 3.5 environ. Précisons que ce polynôme (sans doute quasi optimal) est obtenu en quelques secondes alors que celui utilisé par `crlibm` a été obtenu grâce au savoir faire de ses développeurs qui, en travaillant au cas par cas, ont pu sur cet exemple obtenir un polynôme déjà bien meilleur que le polynôme de meilleure approximation naïvement arrondi. Par rapport à ce polynôme naïvement arrondi, c'est presqu'un facteur 13 que nous gagnons instantanément.

Dans ce cas précis, le gain est trop faible pour améliorer les performances de `crlibm`. En revanche, nous avons rencontré des situations où nous avons pu améliorer considérablement le polynôme utilisé par `crlibm`. Par exemple, nous avons étudié le cas de la fonction arcsin sur l'intervalle [0.79, 1] approché par un polynôme de degré 28. Sur cet exemple, nous avons pu gagner un facteur 2^{16} par rapport au polynôme naïvement arrondi. Ainsi, nous avons pu abaisser le degré du polynôme à 21 et n'utiliser que deux triple-`doubles` et huit double-`doubles` et obtenir néanmoins la précision requise par `crlibm`. Cela permet de gagner 25 % de temps d'exécution et réduit d'un tiers la taille de la table utilisée par `crlibm`. De même, à l'occasion de la venue d'ingénieurs d'Intel dans nos locaux, nous avons travaillé sur une approximation de la fonction erf sur l'intervalle [1, 2]. Le polynôme que nous avons obtenu devrait permettre théoriquement de diviser presque par deux le temps de chargement des tables utilisées pour l'évaluation du polynôme, par rapport à ce qu'ils utilisent actuellement.

3.5 Justification théorique de la méthode

La diversité des outils utilisés (interpolation, théorie de l'approximation polynomiale, algorithme LLL, équivalence des normes) rend la justification théorique de la méthode assez délicate. En effet, notre algorithme fait plusieurs approximations successives : discrétisation du problème aux points x_i, utilisation de la norme euclidienne dans l'espace discrétisé, approximation enfin sur la solution du CVP fournie par ApproximatedCVP.

La seule de ces approximations pour laquelle on possède déjà une estimation quantitative concerne ApproximatedCVP : en effet, nous avons vu que le vecteur $\overrightarrow{v}$ fourni par ApproximatedCVP est au plus dans un facteur $2^{n/2}$ avec la solution exacte. Mais cette borne est déjà très mauvaise par rapport aux résultats effectifs de l'algorithme. Elle ne rend donc pas compte de la réelle efficacité de la méthode.

L'approximation commise en utilisant la norme euclidienne est difficile à quantifier. En effet, elle dépend du choix des points x_i (puisqu'en réalité la norme euclidienne représente les moindres carrés évalués aux point x_i) ; on peut peut-être théoriquement borner la norme euclidienne d'un vecteur en fonction des x_i et de la norme L_∞ du polynôme correspondant, mais nous n'avons pas dégagé de piste claire pour l'instant et, de toutes façons, obtenir une estimation effective d'une telle borne est un problème encore plus délicat.

Enfin, l'erreur due à la discrétisation aux points x_i représente la partie la plus délicate. En effet, comme nous l'avons, vu, le corollaire 2.1 indique que la discrétisation n'entraînerait aucune approximation si nous cherchions un polynôme à coefficients réels : notre choix des points x_i est fait de telle sorte que le polynôme réel interpolant f en ces points soit le polynôme de meilleure approximation. Nous nous appuyons sur l'idée suivant laquelle il existe un polynôme optimal de la forme $\sum_i (a_i/2^{m_i})X^i$ approchant f en norme L_∞ et dont les coefficients soient proches de ceux du polynôme de meilleure approximation réel p^*.

Cette idée semble assez bien vérifiée dans la pratique. Néanmoins, rien ne nous permet de l'affirmer théoriquement pour l'instant. Il est vraisemblable qu'une condition sur les m_i soit nécessaire pour que cette hypothèse soit vérifiée : en effet, dans le cas limite où tous les m_i sont égaux à 0, c'est-à-dire où l'on cherche un polynôme à coefficients entiers pour approcher f, il est évident que cette hypothèse pourra être mise en défaut.

En outre, pour quantifier l'erreur de discrétisation, il faudrait pouvoir quantifier l'écart entre les points x_i et ceux où un polynôme optimal à coefficients dans $\mathbb{Z}/2^{m_i}$ hypothétiquement proche de p^* croise la fonction f. Cette tâche semble pour l'instant totalement hors de portée.

Nous en sommes donc réduit actuellement à constater les excellents résultats pratiques de la méthode et à nous réjouir du fait qu'elle permet déjà d'obtenir des polynômes de qualité parfois bien meilleure que le polynôme de meilleure approximation naïvement tronqué. Par ailleurs, nous continuons de perfectionner la méthode en nous appuyant sur l'intuition et les constatations pratiques.

C'est d'ailleurs là que réside l'espoir d'une future justification théorique : à mesure que nous perfectionnons la méthode, nous simplifions les approximations, nous cernons mieux les problèmes et nous les englobons dans des considérations plus générales. Il est bien possible qu'à force de simplifications, la méthode finisse par être justifiée et quantifiée.

4 Conclusion et perspectives

Nous avons vu que la méthode donne déjà des résultats très prometteurs. L'implantation qui en a été faite n'est pour l'instant qu'un prototype pour tester l'idée générale de l'algorithme et montrer ses possibilités ; à terme, on pourrait imaginer d'intégrer cet algorithme dans une bibliothèque d'aide au développement de fonctions sur machine.

Néanmoins, en l'état, ce prototype a permis de démontrer aux développeurs de `crlibm` que des améliorations tangibles de leur code pourraient être faites dans un avenir proche, tant en terme de rapidité d'exécution de leurs programmes qu'en terme de mémoire utilisée.

En outre, lorsque nous avons présenté ce travail à des ingénieurs d'Intel, ils se sont montrés très intéressés ; aussi, nous envisageons de nouer rapidement un partenariat avec un ou plusieurs industriels.

Bien sûr, beaucoup de questions restent largement ouvertes. Au premier plan, évidemment, une amélioration de la méthode de façon à ce qu'elle fonctionne plus systématiquement (il reste encore une grande part de savoir faire dans l'utilisation de celle-ci) ; à l'arrière-plan, une justification théorique de l'algorithme.

Il est difficile à l'heure actuelle de valider précisément notre algorithme : à part dans quelques cas en faible dimension, on ne possède pas explicitement de polynôme optimal à coefficients dans $\mathbb{Z}/2^{m_i}$ (et pour cause : c'est pour cette raison que nous avons fait tout ce travail !) Du coup, nous ne pouvons pas savoir précisément si notre algorithme se contente de faire mieux en général — mais sans plus — que l'arrondi naïf du polynôme de meilleure approximation ; ou bien s'il fournit un polynôme quasiment optimal. Il est possible que ce problème diminue d'importance à l'avenir car la méthode proposée dans [3]

et qui fournit la liste des polynômes optimaux va bénéficier de notre travail : nous sommes désormais en mesure de donner une meilleure estimation de l'erreur optimale et cela devrait accélérer leur méthode. Ainsi disposera-t-on peut-être bientôt de plus de données concernant les polynômes optimaux ce qui nous permettra d'évaluer plus précisément la qualité de notre méthode.

Nous avons vu qu'en utilisant une petite astuce, nous pouvions fixer des contraintes sur certains coefficients du polynôme. Il est vraisemblable que cette astuce ne fonctionnera pas systématiquement car les contraintes sur le polynôme peuvent nous éloigner considérablement du polynôme de meilleure approximation, mettant en défaut l'hypothèse selon laquelle le polynôme de meilleure approximation et le polynôme à coefficients de la forme $\mathbb{Z}/2^{m_i}$ sont assez proches. La recherche directe de polynômes contraints par une méthode générale constitue donc un de nos futurs axes de recherche.

Par ailleurs, minimiser l'erreur $\|f - p\|_\infty$ n'est pas idéal en pratique ; dans la plupart des situations pratiques (par exemple dans le développement de `crlibm`) ce qui est intéressant, c'est de trouver un polynôme p approchant bien f en erreur relative, c'est-à-dire tel que $\|p/f - 1\|_\infty$ soit minimal. Souvent, les deux problèmes sont quasiment identiques car la fonction f est à peu près constante et ne s'annule pas sur l'intervalle considéré et les erreurs relatives et absolues sont proportionnelles. Du coup, minimiser l'une est équivalent à minimiser l'autre. Cependant, nous avons rencontré des situations pratiques où la fonction f n'est pas constante et où notre méthode ne permet donc pas, en l'état, de donner des résultats pertinents. Nous ne nous sommes pas encore penchés cette question car le bagage théorique existant était plus conséquent pour le problème de minimisation de $\|f - p\|_\infty$ que pour celui de minimisation de $\|p/f - 1\|_\infty$. Mais il est évident que, à terme, l'enjeu se situera sur ce dernier problème.

Enfin, la méthode pourrait sans doute être adaptée à d'autres sortes d'approximations : approximation par des fractions rationnelles par exemple, mais aussi approximation par des polynômes trigonométriques (c'est-à-dire par des sommes de la forme $\sum_k a_k \cos(kx/2)$). Ce sont des objets très utilisés en théorie du signal (filtres RIF) et il semble que le même problème de représentation des coefficients par des nombres flottants se pose dans ce domaine.

Références

[1] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In *STOC*, pages 10–19, 1998.

[2] S. N. Bernstein. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Comm. Soc. Math. Kharkov*, 13, 1912/13.

[3] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32(2), June 2006.

[4] J.-Y. Cai. Some recent progress on the complexity of lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(006), 1999.

[5] E. W. Cheney. *Introduction to approximation theory.* AMS Chelsea Publishing, second edition, 1982.

[6] H. Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993.

[7] R. Descombes. *Éléments de théorie des nombres.* PUF, 1986.

[8] A. K. Lenstra, H. W. Lenstra Jr., and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261 :515–534, 1982.

[9] D. Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6) :2008–2035, March 2001. Preliminary version in FOCS 1998.

[10] P. Nguyen and D. Stehlé. Low-dimensional lattice basis reduction revisited. In *Proceedings of the 6th Algorithmic Number Theory Symposium (ANTS VI)*, volume 3076 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, 2004.

[11] P. Q. Nguyen and J. Stern. The two faces of lattices in cryptology. *Lecture Notes in Computer Science*, 2146 :146–180, 2001.

[12] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *PSAM : Proceedings of the 42th Symposium in Applied Mathematics, American Mathematical Society*, 1991.

[13] E. Ya. Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximations. *Comptes rendus hebdomadaires des scéances de l'académie des sciences, Paris*, 198 :2063–2065, 1934.

[14] D. Stehlé, V. Lefèvre, and P. Zimmermann. Worst cases and lattice reduction. In *Proceedings of the 16th Symposium on Computer Arithmetic (ARITH'16)*, pages 142–147. IEEE Computer Society Press, 2003.

[15] J. Stern. Lattices and cryptography : An overview. *Lecture Notes in Computer Science*, 1431 :50–53, 1998.

[16] G. H. Stewart. *Afternotes goes to Graduate School, Lectures on Advanced Numerical Analysis.* SIAM, 3600 University City Science Center – Philadeplhia, PA 19104 – 2688, 1998.

[17] L. Veidinger. On the numerical determination of the best approximations in the Chebyshev sense. *Numerische Mathematik*, 2 :99–105, 1960.

[18] K. Weierstrass. *Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen.* 1885.

Annexe A : le phénomène de Runge

Lorsqu'on choisit mal les points lors d'une interpolation (typiquement, lorsqu'on choisit des points équirépartis dans l'intervalle), le polynôme interpolateur peut osciller très fortement autour de la fonction et par conséquent l'approcher très mal en norme L_∞.

Plus précisément, si on note p_n le polynôme interpolateur de f en $n+1$ points équirépartis dans l'intervalle $[a, b]$, il se peut que la suite $\|p_n - f\|_\infty$ tende vers $+\infty$. Autrement dit, non seulement la suite p_n n'approche pas bien la fonction f, mais plus on prend de points plus le polynôme est mauvais.

La figure 5 illustre ce phénomène : la courbe en gras est la fonction $x \mapsto 1/(1+25x^2)$; la courbe en trait plein est le polynôme d'interpolation de degré 5 aux points équirépartis ; la courbe en pointillés est le polynôme d'interpolation de degré 9 aux points équirépartis. Les oscillations sont d'ampleur croissante avec le degré et la suite des polynômes interpolateurs diverge.

La figure 6 montre que le même phénomène d'oscillations se produit lorsqu'on utilise les points équirépartis pour notre méthode. C'est ce qui nous conduit à utiliser d'autres points que les points équirépartis.

FIGURE 5 – Phénomène de Runge : la courbe en gras est la fonction $1/(1+25x^2)$; les courbes en trait plein et en pointillés sont les polynômes interpolateurs de degré 5 et 9.

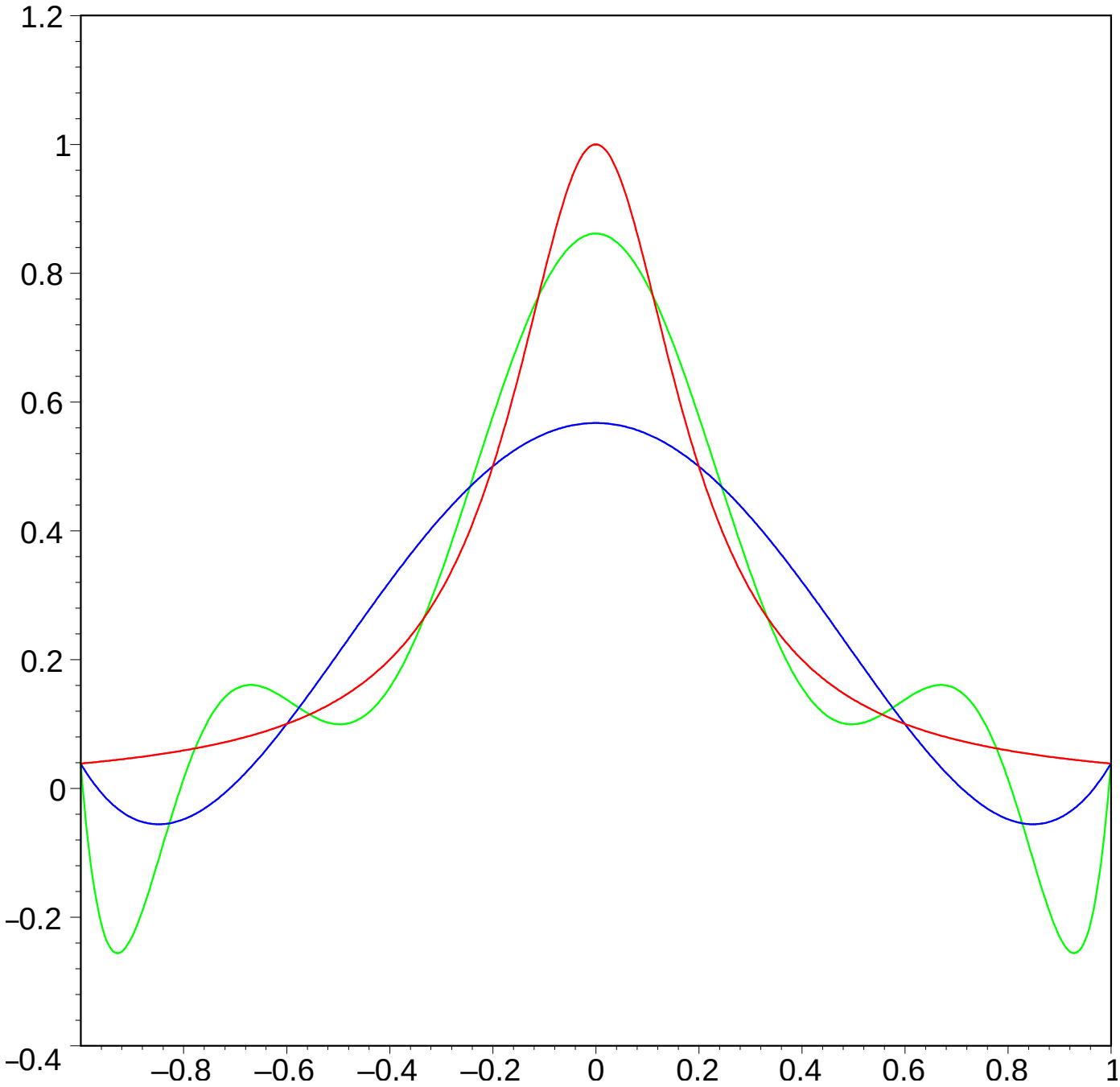
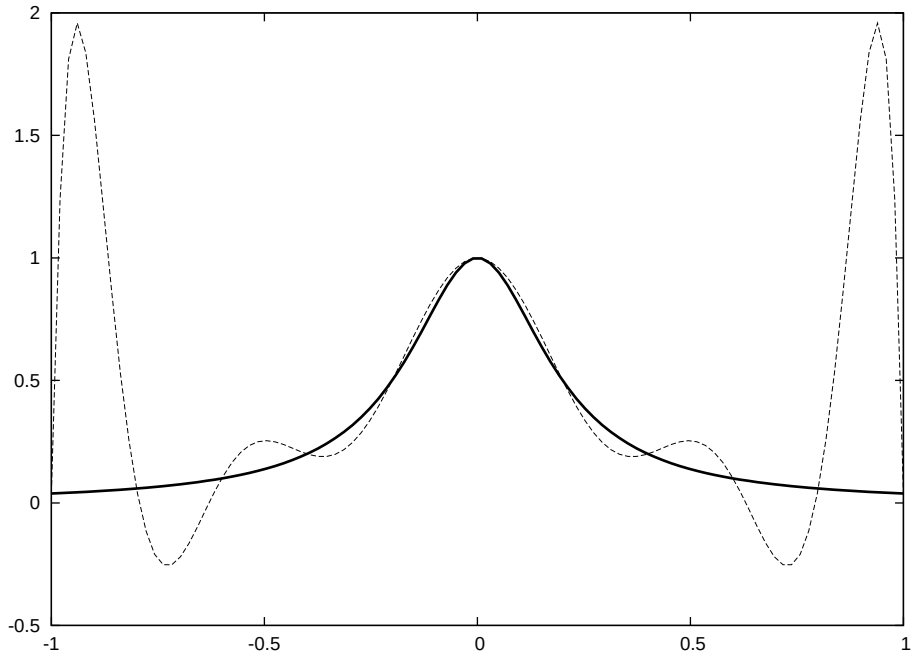


FIGURE 6 – La courbe en gras est la fonction $x \mapsto 1/(1+25x^2)$ et la courbe en pointillés est le polynôme de degré 10 à coefficients de la forme $a_i/2^{50}$ obtenu par interpolation approchée avec des points équirépartis.



Annexe B : preuves

Proposition (2.1). *Le polynôme de meilleure approximation est unique.*

Démonstration. Supposons que p_1 et p_2 soient deux polynômes de meilleure approximation. Alors

$$\begin{aligned} \left\| \frac{p_1+p_2}{2} - f \right\|_\infty &= \left\| \frac{p_1-f}{2} + \frac{p_2-f}{2} \right\|_\infty \\ &\leq \left\| \frac{p_1-f}{2} \right\|_\infty + \left\| \frac{p_2-f}{2} \right\|_\infty \\ &\leq \frac{\mu}{2} + \frac{\mu}{2} = \mu \quad . \end{aligned}$$

Il suit que le polynôme $q = (p_1+p_2)/2$ (qui est aussi de degré inférieur ou égal à n) est un polynôme de meilleure approximation. Notons $x_0, \cdots, x_{n+1}$ les points associés par le théorème de Tchebychev à ce polynôme. En x_i, $|q(x_i) - f(x_i)| = \mu$ et les inégalités précédentes deviennent donc des égalités et on a $|p_1(x_i) - f(x_i)| = \mu$ et $|p_2(x_i) - f(x_i)| = \mu$. Or par définition des x_i on a aussi $|q(x_i) - f(x_i)| = \mu$. On a alors nécessairement $p_1(x_i) - f(x_i) = p_2(x_i) - f(x_i)$. Par conséquent, p_1 et p_2 sont deux polynômes de degré au plus n coïncidant en $n+2$ points : ils sont donc égaux. □

Corollaire (2.1). *Si p est le polynôme de meilleure approximation de f sur $[a, b]$, $f - p$ possède au moins $n+1$ zéros distincts : $z_0, \cdots, z_n$.*

Démonstration. Si $\|f-p\|_\infty = 0$, $f = p$ et n'importe quels points conviennent.

Sinon, posons $x_0 < x_1 < \cdots < x_{n+1}$ les points du théorèmes de Tchebychev. Soit $i \in [\![0, n]\!]$. Puisque le signe de $f(x_i) - p(x_i)$ est différent du signe de $f(x_{i+1}) - p(x_{i+1})$ et puisque $f - p$ est continue, le théorème des valeurs intermédiaire nous indique qu'il existe $z_i \in [x_i, x_{i+1}]$ tel que $f(z_i) - p(z_i) = 0$.

Or $|f(x_i) - p(x_i)| = |f(x_{i+1}) - p(x_{i+1})| = \|f-p\|_\infty \neq 0$. Donc $z_i \in]x_i, x_{i+1}[$. Par suite, les z_i sont distincts. □

Proposition (2.4). *Soit B une base d'un réseau et B^* la famille de Gram-Schmidt associée. Si λ désigne la taille du plus court vecteur non nul du réseau, on a $\lambda \geq \min_i \left\|\overrightarrow{b_i^*}\right\|$.*

Démonstration. Soit $\overrightarrow{v}$ un vecteur du réseau de taille λ. En écrivant $\overrightarrow{v}$ dans la base B, on a $\overrightarrow{v} = a_1\overrightarrow{b_1} + \cdots + a_n\overrightarrow{b_n}$ avec $a_i \in \mathbb{Z}$. La famille $(\overrightarrow{b_j^*}/\|\overrightarrow{b_j^*}\|)$ étant orthonormale, on a

$$\|\overrightarrow{v}\|^2 = \left\| \sum_{j=1}^{n} \left\langle \overrightarrow{v},\, \frac{\overrightarrow{b_j^*}}{\left\|\overrightarrow{b_j^*}\right\|} \right\rangle \frac{\overrightarrow{b_j^*}}{\left\|\overrightarrow{b_j^*}\right\|} \right\|^2$$

d'où

$$\lambda^2 = \sum_{j=1}^{n} \frac{\left\langle \overrightarrow{v},\, \overrightarrow{b_j^*} \right\rangle^2}{\left\|\overrightarrow{b_j^*}\right\|^2} \geq \frac{\left\langle \overrightarrow{v},\, \overrightarrow{b_i^*} \right\rangle^2}{\left\|\overrightarrow{b_i^*}\right\|^2}$$

où i est tel que $a_i \neq 0$ et $a_{i+1} = \cdots = a_n = 0$. Alors

$$\langle \overrightarrow{v},\, \overrightarrow{b_i^*} \rangle = \sum_{j=1}^{n} a_j \langle \overrightarrow{b_j},\, \overrightarrow{b_i^*} \rangle \quad .$$

Or $a_j = 0$ pour $j > i$ et $\langle \overrightarrow{b_j},\, \overrightarrow{b_i^*} \rangle = 0$ pour $j < i$. Il suit que

$$\lambda^2 \geq a_i^2 \frac{\langle \overrightarrow{b_i},\, \overrightarrow{b_i^*} \rangle^2}{\left\|\overrightarrow{b_i^*}\right\|^2} \quad .$$

Or a_i est un entier non nul, donc $a_i^2 \geq 1$. Par ailleurs

$$\langle \overrightarrow{b_i},\, \overrightarrow{b_i^*} \rangle = \left\langle \left(\overrightarrow{b_i^*} + \sum_{k=1}^{i-1} \frac{\left\langle \overrightarrow{b_i},\, \overrightarrow{b_k^*} \right\rangle}{\left\langle \overrightarrow{b_k^*},\, \overrightarrow{b_k^*} \right\rangle} \overrightarrow{b_k^*} \right),\, \overrightarrow{b_i^*} \right\rangle = \langle \overrightarrow{b_i^*},\, \overrightarrow{b_i^*} \rangle + \sum_{k=1}^{i-1} \frac{\left\langle \overrightarrow{b_i},\, \overrightarrow{b_k^*} \right\rangle}{\left\langle \overrightarrow{b_k^*},\, \overrightarrow{b_k^*} \right\rangle} \langle \overrightarrow{b_k^*},\, \overrightarrow{b_i^*} \rangle \quad .$$

Or pour $k < i$, $\langle \overrightarrow{b_k^*},\, \overrightarrow{b_i^*} \rangle = 0$. De là, $\langle \overrightarrow{b_i},\, \overrightarrow{b_i^*} \rangle = \langle \overrightarrow{b_i^*},\, \overrightarrow{b_i^*} \rangle = \|\overrightarrow{b_i^*}\|^2$. On en déduit que $\lambda^2 \geq \|\overrightarrow{b_i^*}\|^2 \geq \min_k \|\overrightarrow{b_k^*}\|^2$ d'où la conclusion. □

Proposition (2.5). *L'algorithme WeaklyReduce produit une base faiblement réduite.*

Démonstration. Si on déroule la boucle d'indice j de l'algorithme, l'expression de $\overrightarrow{b'_k}$ est la suivante :

$$\overrightarrow{b'_k} = \overbrace{\underbrace{\underbrace{\overrightarrow{b_k}}_{\overrightarrow{t_{k-1}}} - \left\lfloor \frac{\langle \overrightarrow{t_{k-1}},\, \overrightarrow{b^*_{k-1}} \rangle}{\langle \overrightarrow{b^*_{k-1}},\, \overrightarrow{b^*_{k-1}} \rangle} \right\rceil \overrightarrow{b^*_{k-1}}}_{\overrightarrow{t_{k-2}}} - \left\lfloor \frac{\langle \overrightarrow{t_{k-2}},\, \overrightarrow{b^*_{k-2}} \rangle}{\langle \overrightarrow{b^*_{k-2}},\, \overrightarrow{b^*_{k-2}} \rangle} \right\rceil \overrightarrow{b^*_{k-2}} - \cdots - \left\lfloor \frac{\langle \overrightarrow{t_1},\, \overrightarrow{b_1^*} \rangle}{\langle \overrightarrow{b_1^*},\, \overrightarrow{b_1^*} \rangle} \right\rceil \overrightarrow{b_1^*}}^{\overrightarrow{t_1}} \quad .$$

Il suit que

$$\langle \overrightarrow{b'_k},\, \overrightarrow{b_j^*} \rangle = \left\langle \left(\overrightarrow{t_j} - \left\lfloor \frac{\langle \overrightarrow{t_j},\, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle} \right\rceil \overrightarrow{b_j^*} \right),\, \overrightarrow{b_j^*} \right\rangle$$

d'où

$$\langle \overrightarrow{b'_k},\, \overrightarrow{b_j^*} \rangle = \langle \overrightarrow{t_j},\, \overrightarrow{b_j^*} \rangle - \left\lfloor \frac{\langle \overrightarrow{t_j},\, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle} \right\rceil \langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle \quad .$$

Enfin

$$\left| \frac{\langle \overrightarrow{b'_k},\, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle} \right| = \left| \frac{\langle \overrightarrow{t_j},\, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle} - \left\lfloor \frac{\langle \overrightarrow{t_j},\, \overrightarrow{b_j^*} \rangle}{\langle \overrightarrow{b_j^*},\, \overrightarrow{b_j^*} \rangle} \right\rceil \right| \leq \frac{1}{2}$$

et l'algorithme produit donc bien une base faiblement réduite. □