Efficient polynomial L^∞-approximations

Nicolas Brisebarre, Sylvain Chevillard

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Efficient polynomial L^∞-approximations

Nicolas Brisebarre — Sylvain Chevillard

N° 6060

December 2006

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--6060--FR+ENG

INRIA
RHÔNE-ALPES

Efficient polynomial L^∞-approximations

Nicolas Brisebarre*†, Sylvain Chevillard†

Thème SYM — Systèmes symboliques
Projet Arénaire

Rapport de recherche n° 6060 — December 2006 —11 pages

Abstract: We address the problem of computing good floating-point-coefficient polynomial approximation to a function, with respect to the supremum norm. This is a key step in most processes of evaluation of a function. We present a fast and efficient method, based on lattice basis reduction, that often gives the best polynomial possible and most of the time returns a very good approximation.

Key-words: Efficient polynomial approximation, floating-point arithmetic, absolute error, L^∞ norm, lattice basis reduction, closest vector problem, LLL algorithm

* LaMUSE, Univ. J. Monnet, 23, rue du Dr P. Michelon, F-42023 Saint-Étienne, France
† LIP, UMR INRIA-CNRS-ENS Lyon-UCB Lyon 5668, École Normale Supérieure de Lyon, 46, Allée d'Italie, F-69364 Lyon CEDEX 07, France

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Approximation polynomiale minimax efficace

Résumé : Nous nous intéressons au problème du calcul efficace d'approximations polynomiales à coefficients flottants minimax. C'est un point crucial dans la plupart des procédés d'évaluation d'une fonction. Nous présentons une méthode rapide et efficace, à base de réduction des réseaux, qui donne souvent le meilleur polynôme possible et renvoie la plupart du temps une très bonne approximation.

Mots-clés : Approximation polynomiale efficace, arithmétique virgule flottante, erreur absolue, norme infinie, réduction de réseaux, CVP, problème du vecteur le plus proche, algorithme LLL

1 Introduction

To evaluate a mathematical function on a computer, in software or hardware, one frequently replaces the function with good polynomial approximations of it. This is due to the fact that floating-point (FP) addition, subtraction and multiplication are carefully and efficiently implemented on modern processors and also because one may take advantage of existing efficient schemes of polynomial evaluation. Such polynomial approximations are used in several recent elementary function evaluation algorithms [13, 15, 19, 5].

The first natural goal for someone who tries to evaluate a function f is thus to obtain a polynomial p which is sufficiently close to f for the approximation required by the application.

Hence, we are naturally led to consider the problem of getting for a given continuous real-valued function f, a real interval $[a, b]$, and a degree $n \in \mathbb{N}$, the polynomial $p \in \mathbb{R}_n[X]$ that approximates f the best way, where $\mathbb{R}_n[X]$ is the $\mathbb{R}$-vector space of the polynomials with real coefficients and degree lesser or equal to n. The optimal polynomial depends on the way used to measure the quality of the approximation. The most usual choices are the supremum norm (or L^∞ norm or absolute error)

$$||p - f||_{\infty,[a,b]} = \sup_{a \le x \le b} |p(x) - f(x)|,$$

or the relative error

$$||p - f||_{\text{rel},[a,b]} = \sup_{a \le x \le b} \frac{1}{|f(x)|} |p(x) - f(x)|$$

or least squares approximations norm

$$||p - f||_{2,[a,b]} = \left(\int_a^b w(x) \left(f(x) - p(x)\right)^2 dx \right)^{1/2},$$

where w is a continuous *weight function*.

The method proposed in this paper aims at minimizing the absolute error between f and polynomials with FP coefficients. In practice, people are mostly interested in minimizing the relative error. This second problem is in general more difficult than the first one (even when searching real coefficient polynomials) and we are currently working on this issue. But we can remark that there are numerous situations where the two problems differ not so much. For example, there is a lot of applications where the domain of definition of the function is firstly cut into small intervals where the order of magnitude of the function is constant. In these cases, the absolute and relative errors become almost proportional. Hence minimizing the absolute error instead of the relative one is in general quite satisfying.

From now on, we will focus on the supremum norm that we shall write $||\cdot||_{[a,b]}$ instead of $||\cdot||_{\infty,[a,b]}$ in the sequel. Among important theoretical works by various mathematicians (like Bernstein, Weierstrass, Lagrange for example), the work of Chebyshev in this area of function L^∞-approximation, is especially remarkable. He showed in particular that when p runs among $\mathbb{R}_n[X]$, $||f - p||_\infty$ reaches a minimum at a unique polynomial and gave a very precize characterization of this best polynomial approximant (see [4] for example). From that characterization, Remez [18] designed an algorithm that makes it possible to compute the best L^∞-polynomial approximation, also called minimax approximation (see [4] for a proof of the algorithm) in a fairly short time (see [22] for a proof of its quadraticity). Therefore, we see that the general situation for L^∞ approximation by real polynomials can be considered quite satisfying. The problem for the scientist that implements in software or hardware such approximations is that he uses finite-precision arithmetic and unfortunately, most of the time, the minimax approximation given by Chebyshev's theorem and computed by Remez' algorithm has coefficients which are transcendental (or at least irrational) numbers, hence not exactly representable with a finite number of bits.

Thus, the coefficients of the approximation usually need to be rounded according to the requirements of the application targeted (for example, in current software implementations, one often uses FP numbers in IEEE single or double precision for storing the coefficients of the polynomial approximation). But this rounding, if carelessly done, may lead to an important loss of accuracy. For instance, if we choose to round to the nearest each coefficient of the minimax approximation to the required format (this yields a polynomial that we will call *rounded minimax* in the sequel of the paper), the quality of the approximation

we get can be very poor. To illustrate that problem, let us look at the following simple example. We want to approach the function $f : x \mapsto \sqrt{2} + \pi x + ex^2$ on the interval $[2, 4]$ by a degree-2 polynomial with IEEE double precision FP coefficients. The minimax approximation is the function f itself. If we round the coefficients of f to the closest `double`, we obtain the polynomial

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

and we have $\|f - \widehat{P}\|_\infty \simeq 2.70622 \cdot 10^{-15}$. But the best degree-2 polynomial with double precision coefficients is

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

for which $\|f - P^*\|_\infty \simeq 2.2243 \cdot 10^{-16}$. Therefore, there is a factor bigger than 10 between the optimum and the rounded polynomial.

In 2005, Brisebarre, Muller and Tisserand proposed in [2] a first general method for tackling this problem. Given $m_0, m_1, \ldots, m_n$ a fixed finite sequence of natural integers, they search for the polynomials of the form

$$p(x) = \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}}x + \cdots + \frac{a_n}{2^{m_n}}x^n \text{ with } a_i \in \mathbb{Z}$$

that minimize $||f - p||_\infty$. The approach given in [2] consists in, first, constructing with care a certain rational polytope containing all the $(a_0, \ldots, a_n) \in \mathbb{Z}^{n+1}$ solution and as few as possible other elements of $\mathbb{Z}^{n+1}$ and, in a second time, efficiently scanning all the points with integer coordinates that lie inside this polytope.

This approach, that currently makes it possible to obtain polynomials of degree up to 10, has two major drawbacks. First of all, its complexity is not precisely estimated but it might be exponential in the worst case, since the scanning of the integer points of the polytope is done using linear rational programming. The other problem is that the efficiency of the method relies on the relevance of the estimation beforehand of the optimal error $\|f - p\|_\infty$. If this error is overestimated, then the polytope may contain too many integer points and the scanning step may become intractable. If this error is underestimated, then the polytope contains no solution. Hence, we were led to design a tool that could give us a better insight of the value of the optimal error, in order to speed up the method of [2]. To do so, we developed a new approach based on Lattice Basis Reduction and in particular on the LLL algorithm. But, indeed, that tool proved to be far more useful than expected: it gives most of the time an excellent approximant (if not the best) and this is done quickly and at low memory cost. The goal of that paper is to present this new approach.

The outline of the paper is the following. In the second section, we recall basic facts about lattices, about the closest vector problem (CVP), an algorithmic problem related to lattices and that appears naturally in our approach, and some algorithm that solves an approximated version of the CVP. Then we present our new approach in Section 3 and give some worked examples in Section 4, before giving a brief conclusion in last section.

2 A reminder on Lattice Basis Reduction and the LLL Algorithm

Let $x = (x_1, \ldots, x_\ell) \in \mathbb{R}^\ell$. We set

$$||x||_2 = (x|x)^{1/2} = (x_1^2 + \cdots + x_\ell^2)^{1/2} \text{ and } ||x||_\infty = \max_{1 \le i \le \ell} |x_i|.$$

A lattice is a discrete algebraic object that is encountered in several domains of various sciences, such as Mathematics, Computer Science or Chemistry. It is a rich and powerful modelling tool thanks to the deep and numerous theoretical, algorithmic or implementation available works (see [3, 9, 12, 20] for example). The lattice structure is ubiquitous in our approach.

Definition 2.1. Let L be a nonempty subset of $\mathbb{R}^\ell$, L is a lattice iff there exists a set of vectors $b_1, \ldots, b_d$ $\mathbb{R}$-linearly independent such that

$$L = \mathbb{Z}.b_1 \oplus \cdots \oplus \mathbb{Z}.b_d.$$

The family $(b_1, \ldots, b_d)$ is a basis of the lattice L and d is called the rank of the lattice L.

In the sequel, we'll have to deal with the following algorithmic problem.

Problem 2.2. Closest vector problem (CVP) *Let $||\cdot||$ a norm on $\mathbb{R}^\ell$. Given a basis of a lattice $L \subset \mathbb{Q}^\ell$ of rank k, $k \leq \ell$, and $x \in \mathbb{R}^\ell$, find $y \in L$ s.t. $||x-y|| = \operatorname{dist}(x, L)$.*
Associated approximation problem: find $y \in L$ s.t. $||x-y|| \leq \gamma \operatorname{dist}(x, L)$ where $\gamma \in \mathbb{R}$ is fixed.

Let us recall some of the complexity results known about this problem. In 1981, van Emde Boas [21] proved that CVP is NP-hard (see also [14]). On the other hand, Goldreich and Goldwasser showed (but their proof is nonconstructive) in [8] that approximating CVP, in the euclidean norm case (CVP_2), to a factor $\sqrt{d/\log d}$ is not NP-hard. Regarding to the infinity norm (CVP_∞), they also show that approximating CVP to a factor $d/\log d$ is not NP-hard[1]. Unfortunately, no polynomial algorithm is known for approximating CVP to a polynomial factor.

Though, if we relax the constraint on the factor, the situation becomes far better.

In 1982, Lenstra, Lenstra and Lovász [11] gave an algorithm that allows one to get "pretty" short vectors in polynomial time. Among many important applications of that algorithm, Babai [1] proposed a polynomial algorithm for solving CVP with an exponential approximation factor.

Algorithm: ApproximatedCVP
Data: An LLL-reduced basis $(b_i)_{1\leq i\leq d}$; its Gram-Schmidt orthogonalization $(b_i^*)_{1\leq i\leq d}$; a vector $\overrightarrow{v}$
Result: An approximation to the factor $2^{d/2}$ of the CVP_2 of $\overrightarrow{v}$
begin
  $\overrightarrow{t} = \overrightarrow{v}$;
  for *($j = d$; $j \geq 1$; 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

Algorithm 1: Babai's nearest Plane algorithm

Remark 2.3. In practice, the result of Babai's algorithm is of better quality and given faster than expected.

For a practical exact CVP, see the work of Kannan [10] in which the given algorithm is super-exponential.

3 Our approach

In many applications, scientist and engineers desire to find the best (or very good) polynomial among those who have FP coefficients and this is the problem we address in that paper. First, let us make the following important remark: to solve that problem, it is sufficient to be able to solve the following problem:

Problem 3.1. *Let f be a continuous real-valued function defined on a given real interval $[a, b]$, let $n \in \mathbb{N}$, $(m_i)_{0\leq i\leq n}$ a finite sequence of rational integers, give one polynomial of the form*

$$p^\star = \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} X + \cdots + \frac{a_n}{2^{m_n}} X^n$$

where the $a_i \in \mathbb{Z}$, that minimizes (or at least makes very small) $\|f - p\|_{[a,\,b]}$.

The problem we address can be reduced to this one by the following heuristic.

Let's denote by P the best polynomial with FP coefficients and by R the minimax. For the sake of simplicity, we will assume here that all the coefficients of P have the same precision t (but the arguments would be the same with several different precisions for the coefficients). In many cases (in particular if

[1] As it is noticed in [16], it seems that the CVP problem when the norm is the infi nity norm is more diffi cult.

t is big enough), the order of magnitude of the i-th coefficient of P is the same as the corresponding coefficient of R. Hence, we can suppose, in a first time, that the exponents of these coefficients are the same. From Remez' algorithm, we know the i-th coefficient of R and thus its exponent e. Since the i-th coefficient of P is a FP number with precision t and since we may suppose that its exponent is e, we can write it $1.u_1 \dots u_{t-1} \cdot 2^e$ where $1u_1 \dots u_{t-1}$ is the unknown binary mantissa of the coefficient. This can be rewritten $\frac{1u_1 \dots u_{t-1}}{2^{t-1-e}}$ and we can set $m_i = t - 1 - e$.

However the guessed exponent may slightly differ from the real optimal one. In that case, the computed coefficient of P will be of the form $a_i/2^{m_i}$ but where a_i needs more than t bits to be written. Our heuristic is then to use this computed coefficient as a new insight for the order of magnitude of the optimal one. We note e' its exponent, and we set m_i to $t - 1 - e'$ and we try again until we reach a fixed point.

This is just a heuristic and we have no proof of convergence for that process. However, in every practical case we met, this procedure was observed to converge in at most two or three steps. In particular, we (sometimes, but rarely) encountered situations where the initial assumption that the coefficients of P and R have the same order of magnitude were completely wrong. But in those cases, this heuristic also converged in two or three steps and let us find a polynomial with FP coefficients which was really good.

Now that we reduced our general problem to Problem 3.1, we start explaining how we solve it.

The first idea is to discretize the continuous problem: let $\ell \geq n+1$, let $x_1, \cdots, x_\ell$ in $[a, b]$, we want $\frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} x_i + \cdots + \frac{a_n}{2^{m_n}} x_i^n$ to be as close as possible to $f(x_i)$ for all $i = 1, \dots, \ell$. That is to say we want the vectors

$$\begin{pmatrix} \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} x_1 + \cdots + \frac{a_n}{2^{m_n}} x_1^n \\ \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} x_2 + \cdots + \frac{a_n}{2^{m_n}} x_2^n \\ \vdots \\ \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}} x_\ell + \cdots + \frac{a_n}{2^{m_n}} x_\ell^n \end{pmatrix} \text{ and } \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_\ell) \end{pmatrix}$$

to be as close as possible, with respect to the $||\cdot||_\infty$ norm. This can be rewritten as: we want the vectors

$$a_0 \underbrace{\begin{pmatrix} \frac{1}{2^{m_0}} \\ \frac{1}{2^{m_0}} \\ \vdots \\ \frac{1}{2^{m_0}} \end{pmatrix}}_{\overrightarrow{v_0}} + a_1 \underbrace{\begin{pmatrix} \frac{x_1}{2^{m_1}} \\ \frac{x_2}{2^{m_1}} \\ \vdots \\ \frac{x_\ell}{2^{m_1}} \end{pmatrix}}_{\overrightarrow{v_1}} + \cdots + a_n \underbrace{\begin{pmatrix} \frac{x_1^n}{2^{m_n}} \\ \frac{x_2^n}{2^{m_n}} \\ \vdots \\ \frac{x_\ell^n}{2^{m_n}} \end{pmatrix}}_{\overrightarrow{v_n}} \text{ and } \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_\ell) \end{pmatrix}}_{\overrightarrow{y}}$$

to be as close as possible. Hence, we have to find $(a_0, \dots, a_n) \in \mathbb{Z}^{n+1}$ that minimize $||a_0\overrightarrow{v_0} + \cdots + a_n\overrightarrow{v_n} - \overrightarrow{y}||_\infty$: this is an instance of CVP_∞.

Two problems arise at this moment: first, how to choose the points x_i and then how do we deal with the CVP_∞ associated.

3.1 Choice of the points x_i

This is a critical step in our approach. We want that a small value of $\|a_0\overrightarrow{v_0} + \cdots + a_n\overrightarrow{v_n} - \overrightarrow{y}\|_\infty$ means a small value of the supremum norm $||p - f||_{[a,b]}$. More precisely, requiring $\|a_0\overrightarrow{v_0} + \cdots + a_n\overrightarrow{v_n} - \overrightarrow{y}\|_\infty$ to be small can be viewed as an approximate interpolation problem and it is well known that, if the points x_i are carelessly chosen, one may find a polynomial that coincides with the function on the points x_i but is pretty far from it on the whole $[a, b]$ (one may consider the classical Runge's example [7, Chap. 2] for instance).

Our choice of the points relies on the observation that when the imposed precision of the coefficients is big enough (i.e. with m_i big enough, which is the case with `double` FP coefficients for instance), the good polynomial approximants to the function will be close to the minimax approximation. Hence, we generally choose the points where f and the degree-n minimax polynomial R are the closest possible, i.e. when $f - R$ cancels. The following classical result (see [4, Chap. 3] for a more general statement) tells us that there are exactly $n + 1$ such points.

Theorem 3.2 (Chebyshev). *A polynomial $p \in \mathbb{R}_n[X]$ is the best approximation of f in $[a,\, b]$ iff there exist $a \leq y_0 < y_1 < \cdots < y_{n+1} \leq b$ such that*

$$\begin{cases} \forall i \in \{0, \cdots, n+1\},\ |f(y_i) - p(y_i)| = \|f - p\|_\infty, \\ \forall i \in \{0, \cdots, n\},\ f(y_{i+1}) - p(y_{i+1}) = -(f(y_i) - p(y_i)). \end{cases}$$

Another possible choice are the Chebyshev points [7, Chap. 2]. They are known to be good choices in some approximation problems (for example, they are the starting points in Remez' algorithm). They proved to be a pretty valuable choice in the experiments we made, especially when the size of the m_i is low.

3.2 Solving of the CVP$_\infty$

Kannan's algorithm [10] makes it possible to solve either CVP$_2$ or CVP$_\infty$ but its complexity is super-exponential. Since the euclidean and infinity norms have the same order of magnitude (remember that, in $\mathbb{R}^\ell$, $||\cdot||_\infty \leq ||\cdot||_2 \leq \sqrt{\ell}\,||\cdot||_\infty$), we preferred to use Babai's algorithm that solves CVP$_2$ with, in theory, an exponential approximation factor but whose performance, in time, space and quality of the result, is directly related to LLL's one, hence very good in practice for n, say, no greater than 50.

We may then consider the approximation of CVP$_2$ provided by Babai's algorithm as the approximation of CVP$_\infty$ searched. But we can also refine that result in the following way. We can use the LLL-reduced basis to explore the neighborhood of the computed approximated CVP$_2$. Let's denote by v the vector given by Babai's nearest plane algorithm and denote by $(\varepsilon_0, \cdots, \varepsilon_n)$ the LLL-reduced basis. LLL gives pretty short vectors in the lattice: for small values of n (say up to 50), ε_0 is often the shortest nonzero vector in the lattice (in norm $||\cdot||_2$) and the other vectors of the basis are also short. Since $||\cdot||_\infty \leq ||\cdot||_2 \leq \sqrt{\ell}||\cdot||_\infty$ in $\mathbb{R}^\ell$, they are also pretty short vectors for $||\cdot||_\infty$. If v is not the exact CVP$_\infty$, it is maybe not so far from it: the exact CVP$_\infty$ is probably of the form $v + a_0\varepsilon_0 + \cdots + a_n\varepsilon_n$ where the a_i are rational integers with small absolute value. We can explore the neighborhood of v, for example by looking at all vectors of the form $v \pm \varepsilon_i$, and test if the corresponding $||\cdot - \overrightarrow{y}||_\infty$ is smaller than $||v - \overrightarrow{y}||_\infty$. If we find such a vector, it improves the quality of the result. If not, we may think that the quality of v is quite good.

4 Examples

We will now present two examples that illustrate the behavior of our algorithm in real situations. Our first example shows the possible benefit that may come from the use of our method in the implementation of a function for the Intel Itanium. The second example well illustrates the gap between an (almost) optimal polynomial and the *rounded minimax*. This example comes from the implementation of the function $\arcsin$ in the library `CRlibm` [17].

To implement our method, we have used Maple and GP[2]. Maple lets us compute the minimax polynomial R (Remez' algorithm is available in Maple with the function `minimax` of the package `numapprox`) which is used to determine the points x_i by solving the equation $R(x) = f(x)$. Then, we use GP to compute a LLL-reduced basis and to solve the approximated CVP$_2$ by Babai's nearest-plane algorithm. We thus get a polynomial P and we finally use Maple to compute $||P - f||_{[a,b]}$ (with Maple's `numapprox[infnorm]`) and compare it to $||R - f||_{[a,b]}$.

Applying our method to an Itanium implementation of a function

The processor Intel Itanium uses a particular FP format called `extended double`: it is a FP format with a 64 bit mantissa. An `extended double` allows more accurate computations, but this gain in accuracy has a cost: `extended doubles` must be loaded in cache one after the other, whereas two regular `doubles` may be loaded at the same time. Moreover, the latency of such an operation is 6 cycles (when a multiplication costs 4 cycles). Thus the loading time is quite critical in such a processor and it is very interesting to replace an `extended double` with a `double` when the accuracy doesn't require it. See [5] and [6] for more informations about the Itanium architecture.

[2] GP is an interactive calculator for number theory algorithms. In particular, it implements LLL algorithm. It is distibuted under GPL at `http://pari.math.u-bordeaux.fr/`

Intel developed and included in the `glibc` a mathematical library optimized for the Itanium. Let us consider an example inspired by the implementation of the function erf in this library. The function erf is defined by $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \mathrm{d}t$ for all $x \in \mathbb{R}$. The domain is cut into several intervals where the function can be approximated by polynomials with reasonable degree (say less than 30). This example deals with erf on the interval $[1; 2]$. The goal is to provide a polynomial, with `extended` and/or `regular double` coefficients, which approximates the function with a relative error bounded by 2^{-64}. The domain is first translated to the interval $[0, 1]$. The problem is thus reduced to the following: find a polynomial $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ such that $||p(x) - \text{erf}(x+1)||_{\text{rel},[0,1]} < 2^{-64}$.

The minimax of degree 18 gives an error of $2^{-61.19}$. *A fortiori*, a polynomial of degree 18 with FP coefficients can't provide the required accuracy. The minimax of degree 19 gives an error of $2^{-66.92}$. Can we find a satisfying polynomial of degree 19 ?

A common strategy consists, when 0 belongs to the definition interval, in using a bigger precision for the first coefficients and a smaller precision for the last ones. Setting the first 8 coefficients to be `extended doubles` and the other to be `doubles`, and rounding each coefficient of the minimax to the nearest in the corresponding format, we obtain an error of $2^{-61.13}$: it is not sufficient. However, using 9 `extended doubles` and using the same classical procedure, we obtain an error of $2^{-64.74}$ which is enough.

When we use our method, we obtain a polynomial with an error of $2^{-64.74}$ and only two `extended doubles`. We saved 7 `extended doubles` thanks to our method. The computing time is the time necessary to Maple for computing minimax approximation since the other steps are instantaneously performed.

Second example: an example from `CRlibm`

The first example showed that our method makes it possible to reduce the size of the coefficients given a target accuracy. This can be also very interesting in hardware applications where each single bit may count.

Another possibility offered by our method is to improve the accuracy provided by the polynomial approximation (compared to the *rounded minimax* for example) while keeping the same precision for the coefficients. Our second example comes from the implementation of the function $\arcsin$ in `CRlibm`. `CRlibm` is a mathematical library which provides correct rounding: the value returned by `CRlibm` when evaluating a function is the exact mathematical value rounded to the nearest `double`. To achieve this goal, the developers of `CRlibm` must use a bigger precision than the standard double precision provided by the processor. They use some special formats such as double-`double` and triple-`double` which are unevaluated sums of two or three regular `doubles`. In particular, they approximate functions by polynomials with double-`double` and triple-`double` coefficients (and also regular `doubles`).

Basically, a double-`double` gives the same precision as a FP number with 106 bits mantissa. However, a number of the form $1.x_1x_2\ldots x_{52}000000001y_1y_2\ldots y_{52} \cdot 2^e$ is representable by the double-`double` $(1.x_1x_2\ldots x_{52} \cdot 2^e + 1.y_1y_2\ldots y_{52} \cdot 2^{e-8-53})$ and is not representable in a 106 bits format. This limitation is not a problem in general: we can often prove *a priori* that to achieve a given precision, the coefficients of a polynomial are so constrained that the 53 first bits are fixed. Thus, one of the two `doubles` of the double-`double` is known and we are back to the problem of searching a regular `double` (that is a FP number with at most 53 bits). At worst, we can't prove that the first 53 bits are fixed and we just search for numbers with 106 bits: the resulting number is representable with a double-`double` (it may just be nonoptimal).

We focus here on the approximation of $\arcsin$ on the interval $[0.79; 1]$. It is a real-life example we worked on with C. Lauter (who is one of the developers of `CRlibm`). After an argument reduction we obtain the problem to approximate

$$g(z) = \frac{\arcsin(1 - (z+m)) - \frac{\pi}{2}}{\sqrt{2 \cdot (z+m)}}$$

where $-0.110 \lesssim z \lesssim 0.110$ and $m \simeq 0.110$. The developers know from previous computations that they need an accuracy of more than 119 bits to achieve correct rounding. The minimax of degree 21 provides this accuracy (see Figure 1). Our method gives a satisfying polynomial with two triple-`doubles`, eight double-

Figure 1: binary logarithm of the absolute error of several approximants

Target	-119
Minimax	-119.83
Rounded minimax	-103.31
Our polynomial	-119.77

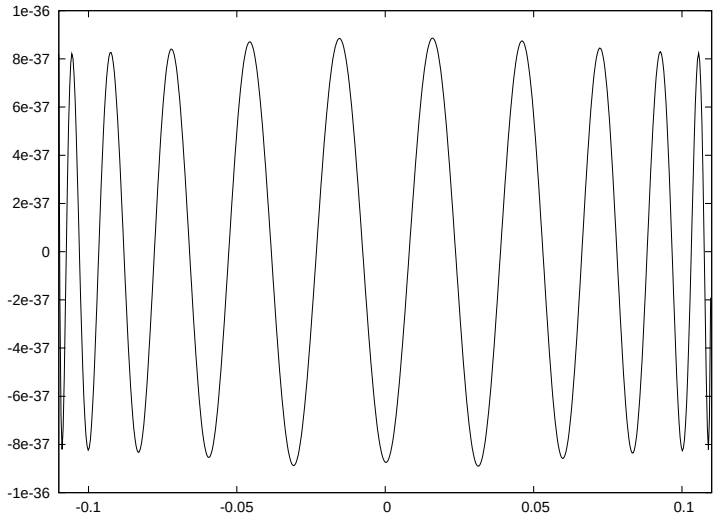
`doubles` and twelve regular `doubles` (here again, the computing time is the time needed by Remez algorithm for computing the minimax approximation). More precisely, we searched for a polynomial of the form

$$\underbrace{p_0}_{173} + \underbrace{p_1}_{159} x + \underbrace{p_2}_{106} x^2 + \underbrace{\cdots}_{\cdots} + \underbrace{p_9}_{106} x^9 + \underbrace{p_{10}}_{53} x^{10} + \underbrace{\cdots}_{\cdots} + \underbrace{p_{21}}_{53} x^{21}$$

where the p_i are FP numbers whose mantissa has the corresponding *size* bits. Note that the first coefficient is actually searched with the method described above: we prove that the first 53 bits are fixed and we then search for a FP number with 106 bits for the two remaining `doubles`. The number 173 indicated under p_0 only means that the 159 bits of this triple-`double` are nonconsecutive.

The accuracy provided by our polynomial is very close to the one given by the minimax. However, without our method, we would have to use the *rounded minimax* which gives only 103 bits of accuracy: it would not be enough to provide correct rounding. This example illustrates the gap between the rounded minimax and the optimal polynomial with FP coefficients. Here, 16 precious bits are lost by the rounding of minimax' coefficients. This loss can be seen very well if the absolute errors are plotted: Theorem 3.2 indicates that a polynomial is optimal if and only if the absolute error oscillates $n + 2$ times between its extrema. Our polynomial almost satisfies this theorem (see Figure 2). On the other hand, one can see on Figure 3 that the *rounded minimax* does not satisfy this theorem at all.

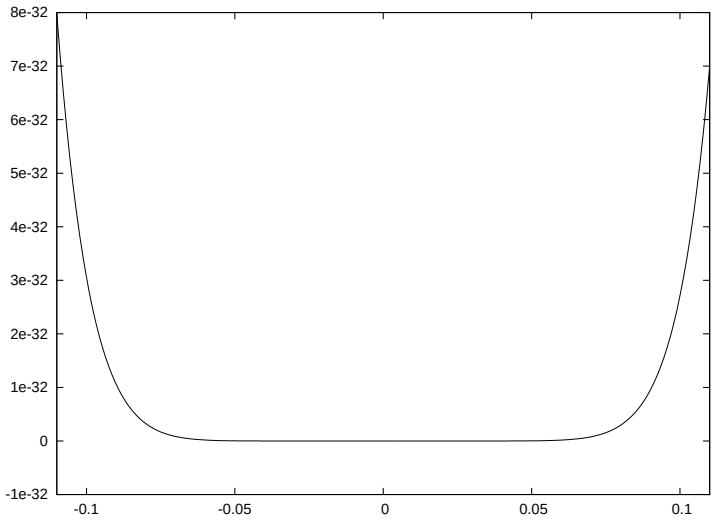
Figure 2: Absolute error between g and our polynomial



5 Conclusion

We have presented a new method for computing efficient polynomial approximation with machine-number coefficients and in particular FP coefficients. It improves the results provided by existing Remez' based

Figure 3: Absolute error between g and rounded minimax



method. The method, based on lattice basis reduction, is much faster and more efficient than the one given in [2] and gives a very good approximant. Moreover, it may considerably speed up the polytope-based method of [2] that gives the best polynomial possible, by providing a relevant (indeed often a very good) estimate of the L^∞ approximation error.

This method can be adapted to several kind of coefficients: fixed-point format, multi-`double` or classical floating point arithmetic with several precision formats.

For some applications, one may want to set the value of certain coefficients of the approximant. For example, one may search for approximants whose first terms match the corresponding one in the Taylor expansion of the approximated function. We should be able to deal with this additional constraint in a close future.

We are currently working on an adaptation of that method to the problem of minimizing the relative error and we plan to extend it to the problem of approximation by sums of cosines that arise in the field of Signal Processing and more specially in FIR filters.

References

[1] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

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

[3] J. W. S. Cassels. *An introduction to the geometry of numbers*. Classics in Mathematics. Springer-Verlag, Berlin, 1997. Corrected reprint of the 1971 edition.

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

[5] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium-Based Systems*. Intel Press, 2002.

[6] Intel corporation. Intel Itanium 2 processor reference manual for software development and optimization. Technical Report 251110-003, Intel, 2004.

[7] W. Gautschi. *Numerical analysis*. Birkhäuser Boston Inc., Boston, MA, 1997. An introduction.

[8] O. Goldreich and S. Goldwasser. On the limits of non-approximability of lattice problems. In *Proceedings of 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–9. ACM, May 1998.

[9] P. M. Gruber and C. G. Lekkerkerker. *Geometry of numbers*, volume 37 of *North-Holland Mathematical Library*. North-Holland Publishing Co., Amsterdam, second edition, 1987.

[10] R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[11] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Annalen*, 261:515–534, 1982.

[12] L. Lovász. *An algorithmic theory of numbers, graphs and convexity*, volume 50 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1986.

[13] P. Markstein. *IA-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000.

[14] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Trans. Inform. Theory*, 47(3):1212–1215, 2001.

[15] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, Boston, 1997.

[16] P. Q. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of CALC '01, Lecture Notes in Computer Science*, volume 2146, pages 146–180. Springer Verlag, 2001.

[17] The Arénaire Project. CRlibm, Correctly Rounded mathematical library, July 2006. `http://lipforge.ens-lyon.fr/www/crlibm/`.

[18] E. Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Acad. Sci. Paris*, 198:2063–2065, 1934.

[19] S. Story and P. T. P. Tang. New algorithms for improved transcendental functions on IA-64. In Koren and Kornerup, editors, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pages 4–11, Los Alamitos, CA, April 1999. IEEE Computer Society Press.

[20] V. Shoup. NTL, a library for doing number theory, version 5.4. `http://shoup.net/ntl/`, 2005.

[21] P. van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Mathematische Instituut, University of Amsterdam, 1981.

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

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399