

Optimizing polynomials for floating-point implementation *

Christoph Lauter, Florent de Dinechin
LIP, ENS-Lyon/INRIA/CNRS/UCBL, Université de Lyon
{Christoph.Lauter,Florent.de.Dinechin}@ens-lyon.fr

Abstract

The floating-point implementation of a function often reduces to a polynomial approximation on an interval. Remez algorithm provides the polynomial closest to the function, but the evaluation of this polynomial in floating-point may lead to catastrophic cancellations when the approximation interval contains zero and some of the polynomial coefficients are very small in magnitude with respects to others. To obtain cancellation-free polynomials while reducing operation count, an algorithm is presented that forces to zero the smaller coefficients thanks to a modified Remez algorithm targeting an incomplete monomial basis. This algorithm generalizes well-known techniques used for odd or even functions to a wider class of functions, and in a purely numerical way, the function being used as a numerical black box. This algorithm is demonstrated, within a larger polynomial implementation tool, on a range of examples, resulting in polynomials with less coefficients than those obtained the usual way.

Keywords: *Polynomial evaluation, floating-point, elementary functions.*

1 Introduction

Scientific and business applications need support for mathematical functions such as e^x , $\sin x$, $\log x$, $\operatorname{erf}x$, and many others. Current processors offer little or no hardware for the evaluation of such functions. What they offer is high performance floating-point hardware for basic operations such as addition and multiplication, or their combination as a fused multiply-and-add. Other mathematical functions are usually approximated on a small domain by polynomials, which can then be evaluated using the high-performance operators of the processor [15, 13, 6]. This article addresses the design of such polynomial approximations. The difficulty is to obtain floating-point implementations of high quality with respect to both performance and precision.

1.1 Obtaining approximation polynomials

Several textbooks [14, 6, 15] discuss techniques to obtain good approximation polynomials. One may use Taylor polynomials, Chebychev approximations, or a minimax approximations. The latter is preferred as it is theoretically the most accurate on a domain: The minimax approximation of a function f on a domain I is defined as the polynomial p that minimizes

*This work was supported by the ANR EVA-Flo project.

the infinite norm of the approximation error $\|p/f - 1\|_\infty^I$. Remez algorithm can be used for numerically computing such minimax approximations [16, 15]. Under some conditions, the algorithm eventually converges to the minimax polynomial.

Throughout this article we focus on relative errors, which are more relevant to floating-point implementations. However, a good implementation of Remez algorithm can accommodate any kind of error through the use of an additional weight function w : Remez algorithm will actually minimize $\|p \cdot w - f\|_\infty^I$ for any w provided by the user.

A few well-known tricks are used for certain classes of functions. For instance, for an odd/even function, a straightforward application of Remez algorithm, with a finite number of Remez iteration steps, will usually not provide an odd/even polynomial, although an odd/even approximation polynomial has several advantages. Firstly, it has twice as few coefficients as the minimax polynomial, and its evaluation will therefore be more economical. Secondly, the implementation will have the same symmetry properties as the function implemented. Thirdly, it is more stable numerically, as the sequel will show. The textbooks therefore advocate the use of an ad-hoc technique (which will be reviewed in Section 2.1) to compute, using standard Remez algorithm, a minimax approximation among the set of odd (resp. even) polynomials.

This article eventually offers a generalization of this ad-hoc technique to a much larger class of functions, but it didn't start this way. The initial motivation of this work was to avoid *cancellations* in the evaluation, which make the evaluation of some polynomials impractical using floating-point arithmetic.

1.2 Floating-point evaluation of a polynomial

Let us consider the Horner evaluation of a polynomial of degree d :

$$p(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + \dots) \dots) \quad .$$

It can be described as a recurrence:

$$\begin{cases} q_n &= a_n \\ q_i &= a_i + x \cdot q_{i+1} \quad \text{for } i = 0 \dots n-1 \\ p(x) &= q_0 \end{cases}$$

A cancellation happens when, for some values of x , the addition in $a_i + x \cdot q_{i+1}$ is effectively a subtraction and the two terms a_i and $x \cdot q_{i+1}$ are very close to each other.

Cancellations in polynomial evaluation should be avoided for two reasons.

- Although a cancelling subtraction is an exact operation, in the sense that it never involves a rounding, some digits of the result are no longer significant. Consider for instance, in a decimal format with 4-digit significand, the subtraction $1.234 - 1.232$. It is an exact subtraction that cancels three digits and returns $2.000 \cdot 10^{-3}$: the three zeroes do not correspond to significant information from the initial numbers. If this result is then multiplied by another value –which is precisely the case in Horner evaluation– this loss of information propagates to the result of the multiplication. As a side effect, error analysis becomes very difficult: Useful theorems about the accuracy of Horner evaluation [1] have hypotheses that ensure that cancellation will not occur.
- Fast double-double [11] and triple-double [12] operations can only be used if the absence of cancellation can be proven beforehand. Otherwise, much slower versions have to be used [10].

The second reason is a motivation to avoid any cancellation, *i.e.* ensure that $x \cdot q_{i+1}$ remains smaller than $a_i/2$ or that both always have the same sign – see Section 3 for the actual criterion used.

Other evaluation schemes exist and are better suited to current superscalar machines, from the family of Estrin schemes [6, 15, 7] to Knuth/Eve transformations [11, 17]. The issue of cancellation is relatively independent of the choice of evaluation scheme, since it is mostly related to the orders of magnitude of the coefficients. More specifically, these other schemes can also be expressed as a tree of operations involving additions and partial polynomials. The cancellation criterion typically translates to a criterion on the respective magnitude of two partial polynomials $x \cdot q_1$ and $x \cdot q_2$. This can be expressed by base changes in the approximation process. The techniques developed for Horner should hence adapt to other schemes as well.

1.3 Contributions

We present an algorithm that detects the possibility of cancellations in a polynomial evaluation scheme. In the presence of such possible cancellations, a new polynomial is computed using a modification to Remez algorithm that forces the offending coefficients to zero, *i.e.* avoids the operation completely.

It turns out that this algorithm numerically discovers the zero coefficients of odd/even functions, but also in many more cases when textbook recipes do not apply – examples will be given in Section 2.1. Even for odd/even functions, it widens the implementation space, as it explores implementations that are not strictly odd/even.

This algorithm is purely numerical: all it requires is a black-box implementation of the function and its first two derivatives. It may therefore be applied to arbitrary compound functions, but also to functions defined by integrals or iterative processes.

This algorithm may even be used for a class of applications studied by Remez [16], then Dunham [9], where a function is defined by a large set of points obtained from some physical experiment. A linear interpolation between these points allows one to recover the function, but a more compact polynomial implementation may be obtained by the technique presented here, provided the set of points is large and accurate enough.

1.4 A complete polynomial implementation chain

The algorithm presented in this article has been implemented and integrated in a larger Sollya¹ program that automatically generates certified C code for Horner’s scheme, using expansion arithmetic from double to triple-double precision [7].

Sollya uses multiple-precision interval arithmetic (based on the MPFR² library) to evaluate expressions with faithful rounding to any precision. It also provides several variants of infinite norm [5], the modified Remez algorithm described in Section 2, and much more.

The final step is to transform the polynomial with real coefficients obtained here into a polynomial with machine-representable coefficients which is the minimax among such machine polynomials [2]. This step is out of the scope of this article, but it has to be initialized with a good approximation polynomial, the one obtained by our algorithm fills this role. Some of the examples provided in Section 4 have gone through this final step, which is also automatic.

¹<http://sollya.gforge.inria.fr/>

²<http://www.mpfr.org/>

This recipe is more or less the state of the art for finding an approximation polynomial – the main improvement over it has consisted in obtaining minimax approximations over the set of polynomials with machine-representable coefficients [2]. Let us now try to apply it to more complex functions.

- $\cos(x^2)$ has a Taylor approximation of the form $1 - 1/2x^4 + 1/24x^8$ with only coefficients for powers of x^4 . The recipe above can be accommodated to such cases.
- $\sin(x) + \cos(x^2)$ has the following Taylor approximation:
 $1 + x - 1/6x^3 - 1/2x^4 + 1/120x^5 - 1/5040x^7 + 1/24x^8 + 1/362880x^9 \dots$ Here the zero coefficients – which will probably lead to cancelling coefficients in minimax approximation – are the coefficients of degree $2 + 4k$ for k integer. This structure can be predicted fairly straightforwardly from the sums of the Taylor series of $\sin(x)$ and $\cos(x^2)$. However, the textbook recipe does not extend straightforwardly to this function: providing a minimax polynomial with this coefficient structure is not obvious. One may apply the recipe to $\sin(x)$, then to $\cos(x^2)$, then add the resulting polynomials coefficient-wise. This will provide a polynomial with the wanted coefficient structure, but there is no reason why it should be the one with minimal error.
- $e^{\sin(x)+\cos(x^2)}$ has a Taylor series with a single zero coefficient at degree 3. Again, this can be predicted analytically or using computer algebra, but it is getting less and less intuitive. And again, can the reader provide a change of variable allowing to compute a minimax polynomial for this function among the set of polynomial whose degree-3 coefficient is zero?

Looking back at the Taylor series for $\sin(x) + \cos(x^2)$, we also remark that even its non-zero coefficients will present risks of cancellation as the degree augments. Any methodology leading to an implementation has to study these cancellation issues at some point. Our approach is to attack the cancellation issues first by removing possibly cancelling operations. This translates to imposing the value 0 to some coefficients. In practice, it happens that such a methodology will, as a side effect, zero out the coefficients which are zero in the Taylor series, and thus minimize the number of operations required for the evaluation, just as the textbook recipe, but for more functions. The precise, mathematical link between the zero coefficients of Taylor polynomials and small coefficients – finally zeroed out by our approach – is currently unclear.

The core of this technique is a modified Remez algorithm computing a minimax polynomial on an incomplete monomial basis, which we present now.

2.2 Modifications to Remez algorithm

Refer to [15] for a good introduction to Remez' algorithm [16, 4]. Modifying this algorithm for an incomplete monomial basis $\{x^{i_0}, x^{i_2}, \dots, x^{i_n}\}$ is fairly straightforward. The core of Remez algorithm solves a linear interpolation problem on $n + 2$ points. In the general case, the matrix of this linear system is a Vandermonde matrix. For an incomplete basis, the matrix simply only contains columns corresponding to the given monomials. This possibility is already present in the initial formulation of the problem by Remez [16].

2.3 Theoretical issues

In order to ensure its convergence, the classical Remez algorithm requires the so-called Haar condition [16, 4]: the determinant of the formal matrix of the linear system does not vanish for

any choice of approximation points in the interval I . This Haar condition is always satisfied for Vandermonde matrices [16], hence for approximations on the complete monomial basis.

Unfortunately, the Haar condition is not fulfilled in general for incomplete bases, in particular if the interval comprises zero [16, 4]. Therefore the convergence of our modified Remez is not proven. Furthermore, when it converges, there is currently no proof that it converges to the expected minimax polynomial. There are indeed counter-examples where the algorithm converges to a polynomial far greater in error than the minimax polynomial.

It is still possible to ensure the Haar condition in some cases. Of practical importance is the case when the function is odd/even and the interval I is symmetrical around zero. In this case, the condition is ensured for an odd/even monomial basis on the positive (or negative) half of the interval.

These issues are under investigation. To the best of our knowledge, there exists some literature addressing them, sometimes dating back to Remez himself [16, 18], but no recent work on a multiple-precision minimax implementation without Haar condition.

2.4 The modified Remez algorithm in practice

In practice, the modified Remez algorithm implemented in the Sollya tool provides useful results. It typically runs even without Haar condition until it converges to some polynomial, or until it can no longer exhibit $n + 2$ appropriate interpolation points. It is possible to verify a posteriori whether or not this polynomial satisfies the target approximation error bound, which is enough to validate the implementation of a function. Experimentally, comparisons with minimax on the complete basis suggest that the polynomial is not far from the optimal. In principle, the optimality could be proven a posteriori using a modification of the polytope exploration described in [3], but this is very costly and out of scope of this paper.

3 Computing cancellation-free approximation polynomials

The complete algorithm is sketched below. It first tests if the approximation polynomial in the complete basis is cancellation-free. If the test fails, all monomials on which cancellations occur are removed from the basis. An approximation polynomial is then computed in the resulting, incomplete basis. The algorithm iterates on increasing degrees until the target error bound is fulfilled or an iteration limit is reached.

The `guessdegree` Sollya function is based on the first iteration of the Remez algorithm combined with a bisection search.

The assignments $q_i \leftarrow p_i^* + x \cdot q_{i+1}$ at lines 15 and 18 of the algorithm must be understood as a symbolic multiplication of the (symbolic) polynomial q_{i+1} by a free variable x and a symbolic addition of p_i^* .

All computations of infinite norms, infima and suprema occurring in the algorithm can use fast but uncertified numerical algorithms. This permits to use a black-box implementation of f . Certification of the error of the final polynomial can be performed a posteriori [5].

The iteration limit l ensures the termination of the algorithm on degenerated problems that may exist. Currently we can neither prove the convergence nor exhibit an example on which it would not terminate. In mathematical terms, the question is whether the set of cancellation-free polynomials of arbitrary degree is dense for any function and approximation interval.

Algorithm 1: The complete approximation algorithm

Input: a function $f \in \mathcal{C}^2$, a domain I , a target error $\bar{\varepsilon} \in \mathbb{R}^+$, an iteration limit $l \in \mathbb{N}$

Output: a polynomial p , cancellation-free in Horner's scheme, such that $\|p/f - 1\|_{\infty}^I \leq \bar{\varepsilon}$

\perp if no such polynomial can be found

```
1  $n \leftarrow \text{guessdegree}(f, I, \bar{\varepsilon}) - 1$ ;           /* Compute initial guess of the required degree */
2 repeat                                           /* Iterate until required degree  $n$  is found */
3    $n \leftarrow n + 1$ ;
4    $p^* \leftarrow \text{remez}(f, I, \{x^0, \dots, x^n\})$ ; /* Compute polynomial in the complete basis */
5    $\varepsilon \leftarrow \|p^*/f - 1\|_{\infty}^I$ ;       /* Compute bound on approximation error */
6 until  $\varepsilon \leq \bar{\varepsilon}$ ;
7  $k \leftarrow 1$ ;                                   /* Iteration count */
8 repeat                                           /* Iterate until appropriate cancellation-free polynomial found */
9    $B \leftarrow \{x^n\}$ ;                             /*  $B$  will hold basis of non-cancelling additions */
10  cancellationfree  $\leftarrow$  true;   $q_n \leftarrow p_n^*$ ;
11  for  $i \leftarrow n - 1$  to 0 do                 /* Statically simulate steps of Horner's scheme */
12     $\bar{\alpha} \leftarrow \sup \{x \cdot q_{i+1}(x) \mid x \in I\}$ ;   $\underline{\alpha} \leftarrow \inf \{x \cdot q_{i+1}(x) \mid x \in I\}$ ;   $\alpha \leftarrow \max(|\underline{\alpha}|, |\bar{\alpha}|)$ ;
13    if  $(\alpha \leq 1/2 \cdot |p_i^*|)$  or  $(\underline{\alpha}, \bar{\alpha}, p_i^*$  have the same sign) then /* No cancellation? */
14       $B \leftarrow B \cup \{x^i\}$ ;                 /* No cancellation, add monomial to basis */
15       $q_i \leftarrow p_i^* + x \cdot q_{i+1}$ ;       /* Statically simulated step in Horner's scheme */
16    else
17      cancellationfree  $\leftarrow$  false;           /* Cancellation occurs */
18       $q_i \leftarrow x \cdot q_{i+1}$ ;             /* Statically simulated step in Horner's scheme */
19  if cancellationfree then return  $p^*$ ;         /* Polynomial in full basis is cancellation-free */
20  else
21     $p \leftarrow \text{remez}(f, I, B)$ ;               /* Compute approx. polynomial in incomplete basis  $B$  */
22     $\varepsilon \leftarrow \|p/f - 1\|_{\infty}^I$ ;     /* Compute associated error bound */
23    if  $\varepsilon \leq \bar{\varepsilon}$  then  $p^* \leftarrow p$ ; /* Error of approx. polynomial in incomplete basis okay? */
24    else
25       $n \leftarrow n + 1$ ;                         /* Increase the degree  $n$  by 1 */
26       $p^* \leftarrow \text{remez}(f, I, \{x^0, \dots, x^n\})$ ; /* Compute polynomial in complete basis */
27       $\varepsilon \leftarrow \|p^*/f - 1\|_{\infty}^I$ ;   /* Compute associated error bound */
28   $k \leftarrow k + 1$ ;
29 until  $k > l$ ;
30 return  $\perp$ ;                                     /* No polynomial has been found in  $l$  iterations */
```

4 Examples

4.1 From a function to a C implementation

Let us first demonstrate the full implementation chain on the function $f(x) = e^{\sin x - \cos x^2}$, to be approximated by a polynomial in the domain $I = [-2^{-8}; 2^{-8}]$ with a relative error less than 2^{-90} . Our algorithm chooses the monomial basis $\{x^0, x^1, x^2, x^4, x^5, x^6, x^7, x^8, x^9\}$, leaving out the monomial x^3 . The final polynomial has the following coefficients:

$$\begin{array}{ll} p_0 = 119383704169626743428469396878343 \cdot 2^{-108} & p_6 = 6516674741954513 \cdot 2^{-56} \\ p_1 = 29845926042406685857117349204375 \cdot 2^{-106} & p_7 = 589077943038783 \cdot 2^{-57} \\ p_2 = 119383704169626743428436621385363 \cdot 2^{-109} & p_8 = 5559725200690211 \cdot 2^{-59} \\ p_4 = 4970345142530923 \cdot 2^{-55} & p_9 = 5320394595779079 \cdot 2^{-58} \\ p_5 = 358969371405011 \cdot 2^{-51} & \end{array}$$

Its approximation error is $\|p/f - 1\|_{\infty}^I < 2^{-90.4}$. Besides, it can be evaluated using Horner's scheme with IEEE 754 double and double-double arithmetic with a round-off error of less than $2^{-93.6}$. This bound has been shown with the Gappa tool [8], and in the process Gappa formally shows that no cancellation may occur.

4.2 Adaptation of the monomial basis to the target accuracy

Here is an example of evolution of the monomial basis chosen by our algorithm, depending on the target approximation error $\bar{\varepsilon}$. The function is $f(x) = e^{\cos x^2 + 1}$ in the domain $I = [-2^{-8}; 2^{-5}]$.

One remarks that for some accuracy targets, our algorithm happens to use more monomials than for slightly higher accuracies. Still, such polynomial may be preferred when their evaluation exhibits more parallelism.

target $\bar{\varepsilon}$	monomial basis
2^{-40}	$\{x^0, x^4\}$
2^{-50}	$\{x^0, x^4, x^8\}$
2^{-60}	$\{x^0, x^4, x^8\}$
2^{-70}	$\{x^0, x^4, x^8, x^{12}\}$
2^{-80}	$\{x^0, x^4, x^8, x^{12}\}$
2^{-90}	$\{x^0, x^4, x^8, x^{12}\}$
2^{-100}	$\{x^0, x^4, x^8, x^{12}, x^{13}, x^{14}, x^{15}\}$
2^{-110}	$\{x^0, x^4, x^8, x^{12}, x^{16}\}$
2^{-120}	$\{x^0, x^4, x^8, x^{12}, x^{16}, x^{17}, x^{18}\}$

4.3 An example of black-box function

The following example illustrates that the algorithm works well on black-box functions. The function $\text{argerf} = \text{erf}^{-1}$ is not currently available in the MPFR library, which the Sollya tool uses for numerics. It was implemented – in a inefficient way – using a Newton-Raphson-iteration on the function erf , its first two derivatives being evaluated based on the code for argerf itself, and dynamically linked to Sollya. Thus the tool had no formal knowledge that it is an odd function.

Nevertheless, for $I = [-1/4; 1/4]$ and $\bar{\varepsilon} = 2^{-60}$, our algorithm finds the odd monomial basis $\{x^1, x^3, x^5, x^7, x^9, x^{11}, x^{13}, x^{15}, x^{17}, x^{19}\}$, with the following coefficients:

$$\begin{array}{ll}
 p_1 = 71899270015270848535577833907197 \cdot 2^{-106} & p_{11} = 7455281238343373 \cdot 2^{-57} \\
 p_3 = 37646369746407330411070885976913 \cdot 2^{-107} & p_{13} = 3086390951797773 \cdot 2^{-56} \\
 p_5 = 2297847774298601 \cdot 2^{-54} & p_{15} = 5269462590206135 \cdot 2^{-57} \\
 p_7 = 3118369096730189 \cdot 2^{-55} & p_{17} = 8758767795225423 \cdot 2^{-58} \\
 p_9 = 2340416807028733 \cdot 2^{-55} & p_{19} = 5369190506948897 \cdot 2^{-57}
 \end{array}$$

Approximation error is bounded by $2^{-62.9}$. The polynomial can be evaluated using Horner's scheme, in double and double-double arithmetic, with an evaluation error bounded by $2^{-62.4}$.

4.4 Illustration of the limitations

If the function is difficult to approximate on the given domain, the proposed algorithm will do no miracle. It is the case for example of $f = \log 1 + x$ on the large domain $I = [-1/2; 1/2]$. The table below shows the monomial bases chosen by our algorithm when it works. The irregularities in these bases are not related to the symmetry of the function, and the algorithm fails to find a polynomial for target accuracies higher than 50 bits – indeed, it is unclear that a cancellation-free polynomial exists in these cases.

accuracy target	monomial basis
2^{-20}	$\{x^1, \dots, x^8, x^{11}, x^{12}\}$
2^{-30}	$\{x^1, \dots, x^{12}, x^{15}, x^{16}\}$
2^{-40}	$\{x^1, \dots, x^{15}, x^{18}, x^{19}, x^{20}, x^{22}, x^{23}\}$
2^{-45}	$\{x^1, \dots, x^{17}, x^{20}, x^{21}, x^{22}, x^{24}, x^{25}\}$
2^{-50}	$\{x^1, \dots, x^{20}, x^{23}, \dots, x^{28}\}$
2^{-55}	failure

Our algorithm also fails to find a cancellation-free polynomial for the (ad-hoc constructed) function f defined by $f(x) = \cos(\pi + 1/16 + x)$ in the domain $I = [-1/4; 1/4]$. As pointed by one of the reviewers, this may be due to the fact that its first derivative f' vanishes in the interval I but that this zero is not at the origin.

5 Conclusion and future works

Obtaining a floating-point C implementation of an arbitrary function has never been so automatic. The tools presented in this article are able to provide a high quality implementation of reasonably complex functions in a few seconds. Some of the a-posteriori validation steps may require many hours, though.

In the process of developing these tools, much insight has been gained in the issue of cancellation in polynomial evaluation, and its relationship to the monomial basis in which polynomials should be searched. The proposed modifications to Remez algorithm are simple in principle but complex in the details, and in the supporting theory. More solid mathematical foundations remain to be built, or rediscovered from ancient literature – the reference books on the subject date back to the '60s. Meanwhile, we have demonstrated an implementation that is very satisfying in practice.

Of course, expertise remains necessary in the implementation of an elementary function. A clever argument reduction can bring in accuracy and performance improvements by orders of magnitude above polynomial approximation alone, and this is totally function-dependent. Still, even in this case, the presented tool will be very helpful in the hands of the expert. Modern table-based argument reduction techniques are often parameterized, and lead to wide trade-offs. Choosing the best set of parameters may be very tedious, and performance-wise, it will be very dependent on the target machine. The tools presented here will help the designer navigate these trade-offs quickly. They have been used (at various stages of maturity) to build some of the functions in the CRLibm library³.

Another, longer-term application of this kind of tool is the construction at compile-time of ad-hoc polynomial evaluators for composite functions appearing in application code.

References

- [1] S. Boldo and M. Daumas. A simple test qualifying the accuracy of Horner's rule for polynomials. *Numerical Algorithms*, 37(1-4):45–60, 2004.
- [2] N. Brisebarre and S. Chevillard. Efficient polynomial L^∞ -approximations. In *18th IEEE Symposium on Computer Arithmetic*, pages 169–176. IEEE, 2007.

³<http://lipforge.ens-lyon.fr/www/crlibm/>

- [3] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32(2):236–256, 2006.
- [4] E. W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, New York, 1966.
- [5] S. Chevillard and C. Lauter. A certified infinite norm for the implementation of elementary functions. In *Int'l Conference on Quality Software*, pages 153–160. IEEE, 2007.
- [6] M. Cornea, J. Harrison, and P.T.P Tang. *Scientific Computing on Itanium-based Systems*. Intel Press, 2002.
- [7] F. de Dinechin, C. Lauter, and J.-M. Muller. Fast and correctly rounded logarithms in double-precision. *RAIRO, Theoretical Informatics and Applications*, 41:85–102, 2007.
- [8] F. de Dinechin, Ch. Q. Lauter, and G. Melquiond. Assisted verification of elementary functions using Gappa. In *Symposium on Applied Computing - MCMS Track*, volume 2, pages 1318–1322. ACM, April 2006.
- [9] C. B. Dunham. Fitting approximations to the Kuki-Cody-Waite form. *International Journal of Computer Mathematics*, 31:263–265, 1990.
- [10] C. Finot-Moreau. *Preuves et algorithmes utilisant l'arithmétique flottante normalisée IEEE*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, July 2001.
- [11] D. Knuth. *The Art of Computer Programming, vol.2: Seminumerical Algorithms*. Addison Wesley, 3rd edition, 1997.
- [12] C. Lauter. Basic building blocks for a triple-double intermediate format. Technical Report RR2005-38, LIP, September 2005.
- [13] R.-C. Li, P. Markstein, J. P. Okada, and J. W. Thomas. The libm library and floating-point arithmetic for HP-UX on Itanium. Technical report, Hewlett-Packard company, april 2001.
- [14] P. Markstein. *IA-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.
- [15] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhäuser, 2nd edition, 2006.
- [16] E. Y. Remez. *Osnovy chislennykh metodov chebyshevskogo priblizheniya*. Akademiya Nauk Ukrainskoy SSR, Institut Matematiki, Naukova Dumka, Kiev, 1969.
- [17] G. Revy. Analyse et implantation d'algorithmes rapides pour l'évaluation polynomiale sur les nombres flottants. Master's thesis, École Normale Supérieure de Lyon, 2006.
- [18] R. Schaback and D. Braess. Eine Lösungsmethode für die lineare Tschebysheff-Approximation bei nicht erfüllter Haarscher Bedingung. *Computing*, 6:289–294, February 1970.