Finding the ”truncated” polynomial that is closest to a function

Nicolas Brisebarre, Jean-Michel Muller

Laboratoire de l'Informatique du Parallélisme

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

Finding the "truncated" polynomial that is closest to a function

Nicolas Brisebarre
Jean-Michel Muller

Avril 2003

Research Report N° 2003-21

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`

Finding the “truncated” polynomial that is closest to a function

Nicolas Brisebarre
Jean-Michel Muller

Avril 2003

Abstract

When implementing regular enough functions (e.g., elementary or special functions) on a computing system, we frequently use polynomial approximations. In most cases, the polynomial that best approximates (for a given distance and in a given interval) a function has coefficients that are not exactly representable with a finite number of bits. And yet, the polynomial approximations that are actually implemented do have coefficients that are represented with a finite - and sometimes small - number of bits: this is due to the finiteness of the floating-point representations (for software implementations), and to the need to have small, hence fast and/or inexpensive, multipliers (for hardware implementations). We then have to consider polynomial approximations for which the degree i coefficient has at most m_i fractional bits (in other words, it is a rational number with denominator 2^{m_i}).We provide a method for finding the best polynomial approximation under this constraint.

Keywords: Computer arithmetic, polynomial approximations

Résumé

Lorsque l’on implante des fonctions suffisament régulières (par exemple des fonctions élémentaires ou spéciales) dans un système de calcul, on utilise souvent des approximations polynomiales. La plupart du temps, le polynôme qui approche le mieux (pour une distance et dans un intervalle donnés) une fonction a des coefficients qui ne sont pas représentables sur un nombre fini de bits. Cependant, les approximations polynomiales utilisées en pratique ont des coefficients écrits sur un nombre fini – souvent petit – de bits : ceci est dû à la finitude des représentations virgule flottante (pour les implantations logicielles) et au besoin d’avoir des circuits multiplieurs de petite taille, donc rapides et/ou peu coûteux (pour les implantations matérielles). Nous devons donc considérer des approximations polynomiales dont le $i$ème coefficient a au plus m_i bits fractionnaires (autrement dit, est un nombre rationnel de dénominateur 2^{m_i}). Nous proposons une méthode permettant d’obtenir le polynôme de meilleure approximation d’une fonction sous cette contrainte.

Mots-clés: Arithmétique des ordinateurs, approximations polynomiales

Finding the “truncated” polynomial that is closest to a function

Nicolas Brisebarre * and Jean-Michel Muller †

1st April 2003

Introduction

All the functions considered in this article are real valued functions of the real variable and all the polynomials have real coefficients.

After an initial *range reduction* step [3, 4, 5], the problem of evaluating a function φ in a large domain on a computer system is reduced to the problem of evaluating a possibly different function f in a small domain, that is generally of the form $[0, a]$. Polynomial approximations are among the most frequently chosen ways of performing this last approximation.

Two kinds of polynomial approximations are used: the approximations that minimize the “average error,” called *least squares approximations*, and the approximations that minimize the worst-case error, called *least maximum approximations*, or *minimax approximations*. In both cases, we want to minimize a distance $||p - f||$, where p is a polynomial of a given degree. For least squares approximations, that distance is:

$$||p - f||_{2,[0,a]} = \left(\int_0^a w(x) \left(f(x) - p(x) \right)^2 dx \right)^{1/2},$$

where w is a continuous *weight function*, that can be used to select parts of $[0, a]$ where we want the approximation to be more accurate. For minimax approximations, the distance is:

$$||p - f||_{\infty,[0,a]} = \max_{0 \leq x \leq a} |p(x) - f(x)|.$$

The least squares approximations are computed by a projection method using orthogonal polynomials. Minimax approximations are computed using an algorithm due to Remez [6, 7]. See [8, 9] for recent presentations of elementary function algorithms.

In this paper, we are concerned with minimax approximations. Our approximations will be used in finite-precision arithmetic. Hence, the computed polynomial coefficients are usually rounded: the coefficient p_i of the minimax approximation

$$p(x) = p_0 + p_1 x + \cdots + p_n x^n$$

is rounded to, say, the nearest multiple of 2^{-m_i}. By doing that, we obtain a slightly different polynomial approximation $\hat{p}$. But *we have no guarantee that $\hat{p}$ is the best minimax approximation to f among the polynomials whose degree i coefficient is a multiple of 2^{-m_i}*. The aim of this paper is to give a way of finding this “best truncated approximation”. We have two goals in mind:

*LArAl, Université Jean Monnet, 23, rue du Dr P. Michelon, F-42023 Saint-Étienne Cedex, France and LIP / Arénaire (CNRS-ENS Lyon-INRIA-UCBL), 46 Allée d’Italie, F-69364 Lyon Cedex 07 FRANCE, Nicolas.Brisebarre@ens-lyon.fr

†LIP / Arénaire (CNRS-ENS Lyon-INRIA-UCBL), 46 Allée d’Italie, F-69364 Lyon Cedex 07 FRANCE, Jean-Michel.Muller@ens-lyon.fr

- rather low precision (say, around 15 bits), hardware-oriented, for specific-purpose implementations. In such cases, to minimize multiplier sizes (which increases speed and save silicon area), the values of m_i, for $i \geq 1$, should be very small. The degrees of the polynomial approximations are low. Typical recent examples are given in [10, 11]. Roughly speaking, what matters here is to reduce the cost (in terms of delay and area) without making the accuracy unacceptable;

- single-precision or double-precision, software-oriented, general-purpose implementations for implementation on current microprocessors. Using Table-driven methods, such as the ones suggested by Tang [13, 14, 15, 16], the degree of the polynomial approximations can be made rather low. Roughly speaking, what matters in that case is to get very high accuracy, without making the cost (in terms of delay and memory) unacceptable.

The outline of the paper is the following. We give an account of Chebyshev polynomials and some of their properties in Section 1. Then, in Section 2, we provide a general method that finds the "best truncated approximation" of a function f over a compact interval $[0, a]$. Sometimes, the cost of our method is too big. Thus, we end this section by a remark that explains how to get in a faster time a "good truncated approximation". Eventually, we deal with two examples, one using our general method and another that uses the remark.

Our method is implemented in Maple programs that can be downloaded from

```
http://www.ens-lyon.fr/~nbriseba/trunc.html
```

We plan to prepare a C version of these programs which should be much faster.

1 Some reminders on Chebyshev polynomials

Definition 1 (Chebyshev polynomials) *The Chebyshev polynomials can be defined either by the recurrence relation*

$$\left\{ \begin{array}{lcl} T_0(x) & = & 1 \\ T_1(x) & = & x \\ T_n(x) & = & 2xT_{n-1}(x) - T_{n-2}(x); \end{array} \right. \tag{1}$$

or by

$$T_n(x) = \left\{ \begin{array}{ll} \cos\left(n \cos^{-1} x\right) & (|x| \leq 1) \\ \cosh\left(n \cosh^{-1} x\right) & (x > 1). \end{array} \right. \tag{2}$$

The first Chebyshev polynomials are listed below.

$$\begin{array}{lcl} T_0(x) & = & 1, \\ T_1(x) & = & x, \\ T_2(x) & = & 2x^2 - 1, \\ T_3(x) & = & 4x^3 - 3x, \\ T_4(x) & = & 8x^4 - 8x^2 + 1, \\ T_5(x) & = & 16x^5 - 20x^3 + 5x. \end{array}$$

An example of Chebyshev polynomial (T_7) is plotted in Fig. 1.

These polynomials play a central role in approximation theory. Among their many properties, the following ones will be useful in the sequel of this paper. A presentation of the Chebyshev polynomials can be found in [1] and especially in [12].

Property 1 *For $n \geq 0$, we have*

$$T_n(x) = \frac{n}{2} \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \frac{(n-k-1)!}{k!(n-2k)!} (2x)^{n-2k}.$$

Hence, T_n has degree n and its leading coefficient is 2^{n-1}. It has n real roots, all strictly between -1 and 1.

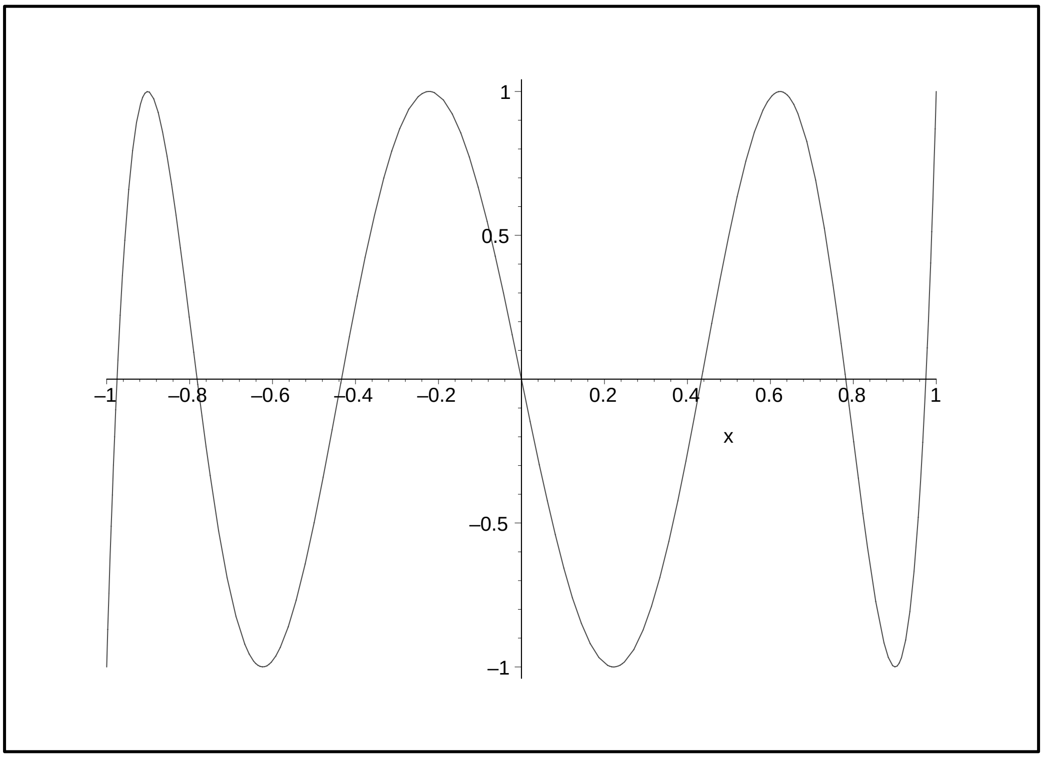

Figure 1: Graph of the polynomial $T_7(x)$.

Property 2 *There are exactly $n+1$ values $x_0, x_1, x_2, \ldots, x_n$ such that*

$$-1 = x_0 < x_1 < x_2 < \cdots < x_n = 1$$

such that

$$T_n(x_i) = (-1)^{n-i} \max_{x \in [-1,1]} |T_n(x)| \quad \forall i,\ i = 0, \ldots,\ n.$$

That is, the maximum absolute value of T_n is attained at the x_i's, and the sign of T_n alternates at these points.

We recall that a *monic* polynomial is a polynomial whose leading coefficient is 1.

Property 3 (Monic polynomials of smallest norm) *Let $a,\ b \in \mathbb{R}$, $a \leq b$. The monic degree-n polynomial having the smallest $||.||_{\infty,[a,b]}$ norm in $[a,b]$ is*

$$\frac{(b-a)^n}{2^{2n-1}} T_n\left(\frac{2x-b-a}{b-a}\right).$$

The central result in polynomial approximation theory is the following theorem, due to Chebyshev.

Theorem 1 (Chebyshev) *Let $a,\ b \in \mathbb{R}$, $a \leq b$. The polynomial p is the minimax degree-n approximation to a continuous function f on $[a,b]$ if and only if there exist at least $n+2$ values*

$$a \leq x_0 < x_1 < x_2 < \ldots < x_{n+1} \leq b$$

such that:

$$p(x_i) - f(x_i) = (-1)^i \left[p(x_0) - f(x_0)\right] = \pm ||f-p||_{\infty,[a,b]}.$$

Throughout the paper, we will make frequent use of the polynomials

$$T_n^*(x) = T_n(2x-1).$$

The first polynomials T_n^* are given below. We have (see [2, Chap. 3] for example) $T_n^*(x) = T_{2n}(x^{1/2})$, hence all the coefficients of T_n^* are non zero integers.

$$\begin{array}{rcl} T_0^*(x) &=& 1, \\ T_1^*(x) &=& 2x-1, \\ T_2^*(x) &=& 8x^2-8x+1, \\ T_3^*(x) &=& 32x^3-48x^2+18x-1, \\ T_4^*(x) &=& 128x^4-256x^3+160x^2-32x+1, \\ T_5^*(x) &=& 512x^5-1280x^4+1120x^3-400x^2+50x-1. \end{array}$$

Theorem 2 (Polynomial of smallest norm with degree-k coefficient equal to 1.) *Let $a \in (0,+\infty)$, define*

$$\beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n = T_n^*\left(\frac{x}{a}\right).$$

Let k be an integer, $0 \le k \le n$, the polynomial of degree at most n with a degree-k coefficient equal to 1 that has the smallest $||.||_{\infty,[0,a]}$ norm in $[0,a]$ is

$$\frac{1}{\beta_k} T_n^*\left(\frac{x}{a}\right).$$

That norm is $|1/\beta_k|$.

Proving this theorem first requires the following results.

Proposition 1 *Let $(\delta_i)_{i=0,\ldots,n}$ be an increasing sequence of non negative integers and*

$$P(x) = a_0 x^{\delta_0} + \cdots + a_n x^{\delta_n} \in \mathbb{R}[x],$$

then either $P = 0$ or P has at most n zeros in $(0,+\infty)$.

Proof. By induction on n. For $n = 0$, it is obvious. Now we assume that the property is true until the rank n. Let $P(x) = a_0 x^{\delta_0} + \cdots + a_n x^{\delta_n} + a_{n+1} x^{\delta_{n+1}} \in \mathbb{R}[x]$ with $0 \le \delta_0 < \cdots < \delta_{n+1}$ and $a_0 a_1 \ldots a_{n+1} \neq 0$. Assume that P has at least $n+2$ zeros in $(0,+\infty)$. Then $P_1 = P/x^{\delta_0}$ has at least $n+2$ zeros in $(0,+\infty)$. Thus, the non zero polynomial $P_1'(x) = (\delta_1-\delta_0)a_1 x^{\delta_1-\delta_0} + \cdots + (\delta_{n+1}-\delta_0)a_{n+1}x^{\delta_{n+1}-\delta_0}$ has, from Rolle's Theorem, at least $n+1$ zeros in $(0,+\infty)$, which contradicts the induction hypothesis. $\Box$

Corollary 1 *Let $(\delta_i)_{i=0,\ldots,n}$ be an increasing sequence of non negative integers and*

$$P(x) = a_0 x^{\delta_0} + \cdots + a_n x^{\delta_n} \in \mathbb{R}[x].$$

If P has at least $n+1$ zeros in $[0,+\infty)$ and at most a simple zero in 0, then $P = 0$.

Proof. If $P(0) \neq 0$, then P has at least $n+1$ zeros in $(0,+\infty)$, hence $P = 0$ from Proposition 1. Suppose now that $P(0) = 0$. We can rewrite P as $P(x) = \sum_{\substack{j=1 \\ j\neq k}}^{n} e_j x^j$. As P has at least $n-1$ zeros in $(0,+\infty)$, it must yet vanish identically from Proposition 1. $\Box$

Proof of Theorem 2. We give the proof (which follows step by step the proof of Theorem 2.1 in [12]) in the case $a = 1$ (the general case is a straightforward generalization). Denote $T_n^*(x) = \sum_{k=0}^{n} a_k x^k$. From Property 2, there exist $0 = \eta_0 < \eta_1 < \cdots < \eta_n = 1$ such that

$$a_k^{-1} T_n^*(\eta_i) = a_k^{-1}(-1)^{n-i}\,\|T_n^*\|_{\infty,[0,1]} = a_k^{-1}(-1)^{n-i}.$$

Let $q(x) = \sum_{\substack{j=0\\ j\neq k}}^{n} c_j x^j \in \mathbb{R}[x]$ satisfy $\|x^k - q(x)\|_{\infty,[0,1]} \leq |a_k^{-1}|$. We suppose that $x^k - q \neq a_k^{-1} T_n^*$. Then the polynomial $P(x) = a_k^{-1} T_n^*(x) - (x^k - q(x))$ has the form $\sum_{\substack{j=0\\ j\neq k}}^{n} d_j x^j$ and is not identically zero. Hence there exist i and j, $0 \leq i \leq j \leq n$, such that $P(\eta_0) = \cdots = P(\eta_{i-1}) = 0$, $P(\eta_i) \neq 0$ and $P(\eta_j) \neq 0$, $P(\eta_{j+1}) = \cdots = P(\eta_n) = 0$ (otherwise, the polynomial q would have at least $n+1$ distinct roots in $[0,1]$ which contradicts Corollary 1). Let l such that $P(\eta_l) \neq 0$ then sgn $P(\eta_l) =$ sgn $a_k^{-1} T_n^*(\eta_l) = (-1)^{n-l}$ sgn a_k^{-1}. Let m such that $P(\eta_l) \neq 0$, $P(\eta_{l+1}) = \cdots = P(\eta_{l+m-1}) = 0$, $P(\eta_{l+m}) \neq 0$: P has at least $m-1$ zeros in $[\eta_l, \eta_{l+m}]$. We distinguish two cases :

- If m is even, we have sgn $P(\eta_l) =$ sgn $P(\eta_{l+m})$ and thus, P must have an even number of zeros (counted with multiplicity) in $[\eta_l, \eta_{l+m}]$.
- If m is odd, we have sgn $P(\eta_l) = -$ sgn $P(\eta_{l+m})$ and thus, P must have an odd number of zeros (counted with multiplicity) in $[\eta_l, \eta_{l+m}]$.

In both cases, we conclude that P has at least m zeros in $[\eta_l, \eta_{l+m}]$.

Then P has at least $j-i$ zeros in $[\eta_i, \eta_j]$. Finally, P has no less than $i + (j-i) + n - j = n$ zeros in $[0,1]$ (P has i zeros in $[\eta_0, \eta_i)$ and P has $n-j$ zeros in $(\eta_j, \eta_n]$). Note that we also obtained that P has no less than $n-1$ zeros in $(0,1]$. Hence, we deduce from Corollary 1 that P vanishes identically. □

2 Getting the "truncated" polynomial that is closest to a function in $[0, a]$.

Let $a \in (0, +\infty)$, let f be a function defined on $[0, a]$ and m_0, m_1, ..., m_n be $n+1$ integers. Define $\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$ as the set of the polynomials of degree less than or equal to n whose degree-i coefficient is a multiple of 2^{-m_i} for all i between 0 and n (we will call these polynomials "truncated polynomials").

We are looking for a truncated polynomial $p^\star \in \mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$ such that

$$||f - p^\star||_{\infty,[0,a]} = \min_{q \in \mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}} ||f - q||_{\infty,[0,a]}. \tag{3}$$

Let p be the minimax approximation of f on $[0, a]$. Define $\hat{p}$ as the polynomial whose degree-i coefficient is obtained by rounding the degree-i coefficient of p to the nearest multiple of 2^{-m_i} (with an arbitrary choice in case of a tie) for $i = 0, \ldots, n$: the polynomial $\hat{p}$ is an element of $\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$. It should be noticed that $\hat{p}$ is not necessarily equal to $p^\star$. Also define ϵ and $\hat{\epsilon}$ as

$$\epsilon = ||f - p||_{\infty,[0,a]} \text{ and } \hat{\epsilon} = ||f - \hat{p}||_{\infty,[0,a]}.$$

In the following, we compute bounds on the coefficients of a polynomial q such that if q is not within these bounds, then

$$||f - q||_{\infty,[0,a]} > \epsilon + \hat{\epsilon}.$$

Knowing these bounds will allow an exhaustive searching of $p^\star$. To do that, consider a polynomial q whose degree-i coefficient is $p_i + \delta_i$. Let us see how close can q be to p. We have

$$(q - p)(x) = \delta_i x^i + \sum_{j \neq i} (q_j - p_j) x^j.$$

Hence, $||q - p||_{\infty,[0,a]}$ is minimum implies that

$$||x^i + \frac{1}{\delta_i} \sum_{j \neq i} (q_j - p_j) x^j||_{\infty,[0,a]}$$

is minimum.

Hence, we have to find the polynomial of degree n, with fixed degree-i coefficient, whose norm is smallest. This is given by Theorem 2. Therefore, we have

$$||x^i + \frac{1}{\delta_i}\sum_{j\neq i}(q_j - p_j)x^j||_{\infty,[0,a]} \geq \frac{1}{|\beta_i|},$$

where β_i is the non-zero degree-i coefficient of $T_n^*(x/a)$.

$$||q-p||_{\infty,[0,a]} \geq \frac{\delta_i}{|\beta_i|}.$$

Now, since $\hat{p} \in \mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$, if a polynomial is at a distance greater than $\hat{\epsilon}$ from p, it cannot be $p^\star$. Therefore, if there exists i, $0 \leq i \leq n$, such that

$$|\delta_i| > (\epsilon + \hat{\epsilon})|\beta_i|$$

then

$$||q-f|| \geq ||q-p|| - ||p-f|| > \hat{\epsilon}:$$

the polynomial q cannot be the element of $\mathcal{P}_n^{[m_0,m_1,\ldots,m_n]}$ that is closest to f. Hence, the i-th coefficient of $p^\star$ necessarily lies in the interval $[p_i - \hat{\epsilon}|\beta_i|, p_i + \hat{\epsilon}|\beta_i|]$. Thus we have

$$\lceil 2^{m_i}p_i - (\epsilon+\hat{\epsilon})|\beta_i|\rceil \leq 2^{m_i}p_i^\star \leq \lfloor 2^{m_i}p_i + (\epsilon+\hat{\epsilon})|\beta_i|\rfloor. \tag{4}$$

Remark. As it can be seen in the examples, the number of polynomials to test given by the conditions (4) may be too large to produce in a "reasonable time" the optimal polynomial. And yet, we can perform a partial search which will not necessarily give the best truncated polynomial but one better than $\hat{p}$. To do so, we are going to search for, among the truncated polynomials closer than $\hat{p}$ to the minimax polynomial p, the one that is closest to f. This polynomial will be denoted $p^\times$. We proceed as follows.

Define η as

$$\eta = ||\hat{p} - p||_{\infty,[0,a]}.$$

Now, we compute bounds on the coefficients of a polynomial q such that if q is not within these bounds, then

$$||p-q||_{\infty,[0,a]} > \eta.$$

Knowing these bounds will allow an exhaustive searching of $p^\times$. To do that, consider a polynomial q whose degree-i coefficient is $p_i + \delta_i$. Now, as in the previous section, we obtain that, if there exists i, $0 \leq i \leq n$, such that

$$|\delta_i| > \eta|\beta_i|$$

then q cannot be $p^\times$. Hence, the i-th coefficient of $p^\times$ necessarily lies in the interval $[p_i - \eta|\beta_i|, p_i + \eta|\beta_i|]$. Thus we have

$$\lceil 2^{m_i}p_i - \eta|\beta_i|\rceil \leq 2^{m_i}p_i^\times \leq \lfloor 2^{m_i}p_i + \eta|\beta_i|\rfloor.$$

3 Examples

3.1 Cosine function in $[0, \pi/4]$ with a degree-3 polynomial.

In $[0, \pi/4]$, the distance between the cosine function and its best degree-3 minimax approximation is 0.00011. This means that such an approximation is not good enough for single-precision implementation of the cosine function. It can be of interest for some special-purpose implementations.

```
m := [12,10,6,4]:polstar(cos,Pi/4,3,m);

  "minimax = ", .9998864206

         + (.00469021603 + (-.5303088665 + .06304636099 x) x) x


            "Distance between f and p = ", .0001135879209

                              3    17   2
              "hatp = ", 1/16 x   - -- x   + 5/1024 x + 1
                                   32


           "Distance between f and hatp = ", .0006939707768

degree 0: 6 possible values between 4093/4096 and
    2049/2048
degree 1: 38 possible values between -7/512 and
    23/1024
degree 2: 8 possible values between -37/64 and
    -15/32
degree 3: 1 possible values between 1/16 and
    1/16
1824 polynomials need be checked

                              3    17   2                4095
             "pstar = ", 1/16 x   - -- x   + 3/512 x + ----
                                   32                    4096


           "Distance between f and pstar =", .0002441406250


                  "Time elapsed (in seconds)", 8.080
```

3.2 Exponential function in $[0, \log(1 + 1/2048)]$ with a degree-3 polynomial.

In $[0, \log(1 + 1/2048)]$, the distance between the exponential function and its best degree-3 minimax approximation is around 1.8×10^{-17}, which should be sufficienty for a faithfully rounded double precision implementation with much care in the polynomial implementation. Unfortunately, the bounds given to get $p^\star$ are too large (there are 18523896 polynomials to test). Hence, we will only try to determine the polynomial $p^\times$.

```
Digits:=30: m := [56,45,33,23]: poltimes(exp,log(1.+1./2048),3,m);

"minimax = ", .999999999999999981509827946165
+ (1.00000000000121203815619648271 + (.499999987586063030320493910112
 + .166707352549861488779274879363 x) x) x

                                                                   -16
         "Distance between f and p = ", .1849017208895 10
```

```
             1398443  3   4294967189  2   35184372088875
  "hatp = ", ------- x  + ---------- x  + -------------- x
             8388608      8589934592      35184372088832

           72057594037927935
         + -----------------
           72057594037927936

                                                                                      -16
  "Distance between f and hatp = ", .236242209693262352294431493060 10

                                                                                      -17
  "Distance between p and hatp = ", .531982124948018688509983966915 10
```

```
degree 0: 1 possible values between 72057594037927935/72057594037927936
and 72057594037927935/72057594037927936
degree 1: 14 possible values between 8796093022217/8796093022208 and
351843720888
degree 2: 18 possible values between 4294967181/8589934592 and
2147483599/4294967296
degree 3: 24 possible values between 1398431/8388608 and
699227/4194304
6048 polynomials need be checked
```

```
               1398443  3   2147483595  2   35184372088873
  "ptimes = ", ------- x  + ---------- x  + -------------- x
               8388608      4294967296      35184372088832

           72057594037927935
         + -----------------
           72057594037927936
                                                                                       -16
  "Distance between f and ptimes =", .202462803670964701822850663822 10


          "Time elapsed (in seconds) =", 1970.699
```

Appendix: Maple program that computed the polynomial $p^{\star}$

```
with(numapprox);with(orthopoly);

polstar := proc(f,a,n,m)
local p, i, hatp, poltronq, hatepsilon, epsilon, beta, prod,
ecart, coeffp, temps;
global pstar, minpstar, smallest, largest, mm, aa;
temps:=time():
mm:=m; aa:=a;
p := minimax(f(x),x=0..a,[n,0],1,'epsilon');
print("minimax = ",p);
print("Distance between f and p = ",epsilon);
for i from 0 to n do
  hatp[i] := round(2^m[i+1]*coeff(p,x,i))/2^m[i+1];
od;
poltronq := add(hatp[i]*x^i,i=0..n);
print("hatp = ",sort(poltronq));
```

```
hatepsilon := infnorm(poltronq-f(x),x=0..a);
print("Distance between f and hatp = ",hatepsilon);
beta := T(n,2*(x/a)-1); prod := 1;
for i from 0 to n do
   ecart := abs((epsilon+hatepsilon)*coeff(beta,x,i));
   coeffp := coeff(p,x,i);
   smallest[i] := ceil((coeffp-ecart)*2^m[i+1]);
   largest[i] := floor((coeffp+ecart)*2^m[i+1]);
   printf("degree %a: %a possible values between %a and
   %a\n",i,largest[i]-smallest[i]+1,smallest[i]*2^(-m[i+1]),
   largest[i]*2^(-m[i+1]));
   prod := prod*(largest[i]-smallest[i]+1)
od;
printf("%a polynomials need be checked",prod);print();
pstar:=poltronq;
minpstar:=hatepsilon;
selpolstar(n,f,0);
print("pstar = ",sort(pstar));
print("Distance between f and pstar =",minpstar);
print("Time elapsed (in seconds)", time() - temps);
end:

selpolstar:=proc(k,f,P)
local i, reste;
global minpstar, pstar;
if k = -1 then reste:= infnorm(f(x)-P,x=0..aa);
               if reste < minpstar
                        then  minpstar:= reste;
                              pstar:= P;
               fi
          else for i from smallest[k] to largest[k]
                          do selpolstar(k-1,f,P+i*x^k/2^mm[k+1])
                          od
fi
end:
```

References

[1] P. Borwein and T. Erdélyi, *Polynomials and Polynomials Inequalities*, Graduate Texts in Mathematics, **161**, Springer-Verlag, 1995.

[2] L. Fox and I. B. Parker, *Chebyshev Polynomials in Numerical Analysis*, Oxford Mathematical Handbooks, Oxford University Press, 1972.

[3] M. Payne and R. Hanek. Radian reduction for trigonometric functions. *SIGNUM Newsletter*, 18:19–24, 1983.

[4] K. C. Ng. Argument reduction for huge arguments: Good to the last bit (can be obtained by sending an e-mail to the author: kwok.ng@eng.sun.com). Technical report, SunPro, 1992.

[5] M. Daumas, C. Mazenc, X. Merrheim, and J.- M. Muller. Modular range reduction: A new algorithm for fast and accurate computation of the elementary functions. *Journal of Universal Computer Science*, 1(3):162–175, March 1995.

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

[7] J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. G. Thacher, and C. Witzgall. *Computer Approximations*. Wiley, New York, 1968.

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

[9] P. Markstein. *IA-64 and Elementary Functions: Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.

[10] B. Wei, J. Cao and J. Cheng. High-performance architectures for elementary function generation. In Burgess and Ciminiera, editors, *Proc. of the 15th IEEE Symposium on Computer Arithmetic (Arith-15)*. IEEE Computer Society Press, 2001.

[11] J.A. Pineiro, J.D. Bruguera, and J.-M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In Burgess and Ciminiera, editors, *Proc. of the 15th IEEE Symposium on Computer Arithmetic (Arith-15)*. IEEE Computer Society Press, 2001.

[12] T. J. Rivlin. *Chebyshev polynomials. From approximation theory to algebra and number theory.* Second edition. Pure and Applied Mathematics. John Wiley & Sons, Inc., New York, 1990.

[13] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989.

[14] P. T. P. Tang. Table-driven implementation of the logarithm function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 16(4):378–400, December 1990.

[15] P. T. P. Tang. Table lookup algorithms for elementary functions and their error analysis. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 232–236, Grenoble, France, June 1991. IEEE Computer Society Press, Los Alamitos, CA.

[16] P. T. P. Tang. Table-driven implementation of the expm1 function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 18(2):211–222, June 1992.