Checking Computation of Numerical Functions by the Use of Functional Equations

F. Vainstein and C. Jones

Georgia Institute of Technology
feodor.vainstein@gtsav.gatech.edu

Abstract: Systematic use of functional equations for fault-tolerant computation of numerical functions have been introduced by M. Blum (1989) and then independently by F. Vainstein (1991). The later has introduced definition of polynomially checkable (PC) functions – the functions for which functional equations are polynomials, and proved that the class of PC functions is large and includes many commonly used functions. The functional equations that are used to check computations of numerical functions are called checking polynomials. In this paper we discuss an algorithm for computing coefficients of these polynomials. By using this algorithm we obtain checking polynomials for the commonly used functions.
Keywords: fault tolerance, algebraic methods, numerical functions, error checking, checking polynomials.

1. Introduction

Computers continue to take on more mission and safety critical operations in industrial, scientific, and consumer markets. Modern processors compute a wide range of numerical functions. Detecting and correcting errors due to numerical computations are critical aspects of processor design.

There have been numerous approaches to fault tolerant computation of numerical functions. These include hardware, information, time, and software redundancy methods (Lala 2001). However, each of these methods comes at a significant price to the system in space or time. And while the dimensions of chip technology are continually reduced the complexity of the systems placed on chips continues to rise.

The technique described here employs the algebraic concept of the transcendental degree of field extensions to exploit the structure of a specific numerical computation. This method requires significantly less hardware redundancy, offers good fault coverage, and has significant fault location capability (Vainstein 1993).

Algorithms used in numerical computations can be sophisticated and numerous implementations exist (Koren 2001; Muller 1997; Ercegovac, Lang et al 2000). Many considerations go into choosing a certain numerical algorithm based on specific application and design criteria. Contrary to the perception of many, the computation of numerical functions can be quite complex and susceptible to faults.

In order to give the flavor of an algorithm for computing an elementary function, consider Wong and Goto's algorithm for computing logarithms (Wong and Goto 1994). This description is based on the presentation of this algorithm given by Muller in (Muller 1997). See Muller's text for a complete description of the assumptions, details, and technical issues of the algorithm.

The notation

$$[z]_{a-b}$$

is the number obtained by zeroing all the bits of *z* but the bits *a* to *b*. For example, if $m = m_0.m_1m_2m_3m_4\ldots$, then

$$[m]_{1-3} = 0.m_1m_2m_3000\ldots$$

To compute the logarithm of a normalized IEEE-754 double precision floating-point number

$$x = m \times 2^{\exp onent}$$

We have to follow the steps below:

1. Obtain factor K_1 and $\ln(K_1)$ from tables.
2. Use a rectangular multiplier to multiply *m* by K_1. Then K_2 is chosen such that K_1K_2m is close to 1. And $[\ln(K_2)]_{1-56}$ is obtained from tables.
3. Use a rectangular multiplier to multiply (mK_1) by K_2. As in the previous step $[\ln(K_3)]_{1-56}$ is obtained from tables.
4. Use a rectangular multiplier to multiply (mK_1K_2) by K_3. The result is $1-\gamma$, where $0 \le \gamma < 2^{-24}$. This result is close enough to 1 that a degree-3 Taylor polynomial approximation will give good accuracy.
5. Then, full multiplication and tables are used to compute

$$\left[\frac{\gamma^2}{2}\right]_{1-56}$$

and

$$\left[\frac{[\gamma]^3_{25-33}}{3}\right]_{1-56}$$

6. And, finally,

$$\ln(x) \approx exponent \times \ln(2) - \ln K_1 - \ln K_2 - \ln K_3 - \gamma - \left[\frac{\gamma^2}{2}\right]_{1-56} - \left[\frac{[\gamma]^3_{25-33}}{3}\right]_{1-56}$$

As we see from this example, even "simple" numerical functions such as the logarithmic function can be quite sophisticated and thus susceptible to faults.

The method presented in this paper seeks to address the fault-tolerant needs of numerical algorithms with low processor overhead. To illustrate this method, let us consider an example.

Suppose we have to compute the function $f(x) = e^{-x}\sin 5x, x \in [0,10]$

Let $a_1, a_2 \in R$;

Denote by $f_0 = f(x+0) = e^{-x}\sin 5x,$

$$f_1 = f(x+a_1) = e^{-(x+a_1)}\sin 5(x+a_1),$$

$$f_2 = f(x+a_2) = e^{-(x+a_2)}\sin 5(x+a_2).$$

Denote by $p_1 = e^{-a_1}\cos 5a_1;\ q_1 = e^{-a_1}\sin 5a_1;\ p_2 = e^{-a_2}\cos 5a_2;\ q_2 = e^{-a_2}\sin 5a_2;$

$$A = p_1 q_2 - p_2 q_1;\ B = -q_2;\ C = q_1$$

Then $Af_0 + Bf_1 + Cf_2 = 0$

For every $x \in R$.

It is very important that A, B and C do not depend on x and depend only on a_1 and a_2 Taking (1) into consideration we can consider the following method for error detection.

Denote the computed values of function f at the points $x, x+a_1, x+a_2$ by $\tilde{f}_0, \tilde{f}_1, \tilde{f}_2$ respectively. Then if the computation is correct

$A\tilde{f}_0 + B\tilde{f}_1 + C\tilde{f}_2 = 0$ (Independently of *x*!) (2)

For error correction (single error in this case) we can proceed as follows.

Consider $a_1 = a;\ a_2 = 2a$ and let $A\tilde{f}_0 + B\tilde{f}_1 + C\tilde{f}_2 \neq 0$ (3)

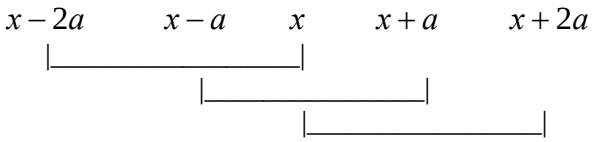
Because one of $\tilde{f}_i \neq f_i;\ i = 0, 1, 2.$

Suppose for example that $\tilde{f}_0 \neq f_0;\quad \tilde{f}_1 = f_1;\ \tilde{f}_2 = f_2$

Then the correct value is given by the formula

$$f_0 = -\frac{B}{A}\tilde{f}_1 - \frac{C}{A}\tilde{f}_2 \quad (4)$$

Location of the error can be obtained by using (2) for the following triples:



It should be taken into consideration that computations are done in practice with a certain level of accuracy. Hence the formula 2 should be substituted by the formula

$$| \ A\tilde{f}_0 + B\tilde{f}_1 + C\tilde{f}_2 \ | \le \delta, \qquad (2')$$

where δ is a small positive number specified by the precision of the computation.

2. Polynomial checking

For the readers convenience let us present the following definitions and results from the field extension theory (Lang 1992).

Definition 1. Let $K \subset L$ be a field extension and $K[T_1,....,T_n]$ be the set of all polynomials in $T_1,...,T_n$ over *K*. The elements $a_1,....a_n \in L$ are called algebraically dependent over *K*, if there exists a polynomial $P \in K[T_1,...,T_n], P \neq 0$, such that $P(a_1,...a_n) = 0.$ The elements $a_1,...,a_n \in L$ are called algebraically independent over *K*, if they are not algebraically dependent. By $K(T_1,...,T_n)$ we denote the quotient field of the ring $K[T_1,...,T_n]$.

Example 2. Consider the field extension $Q \subset R.$ Then the numbers $\sqrt{2}$ and $\sqrt{3} \in R$ are algebraically dependent over *Q*. $P(T_1,T_2) = T_1^2 + T_2^2 - 5.$ The numbers 1, $\pi \in R$ are algebraically independent over *Q*.

Definition 2. Let $K \subset L$ be a field extension. Transcendental degree (Tr.deg.) of this extension is by definition the maximum possible number of elements from *L* algebraically independent over *K*.

If Tr.deg. of $K \subset L$ is equal to *n* and *m>n,* the any subset $\{ a_1,...,a_m \} \subset L$ is algebraically dependent.

Example 3. Tr.deg. of $R \subset R(T)$ equals to 1.

Tr.deg. of $R \subset R(x,e^x)$ equals to 2.

Tr.deg. of $R(x,\sin x,e^x) \subset R(x,\sin x,\cos x,e^x)$ equals to 0.

Definition 3. A field *L* is called algebraically closed if any polynomial $P \in L[T]$ has a root in *L.*

Example 4. *R* is not algebraically closed. $P(T) = T^2 + 1$ does not have roots in *R*. *C* is algebraically closed.

Definition 4. A field $\overline{K}$ is called an algebraic closure of field *K* is

1. $\overline{K}$ is algebraically closed.

2. Tr.deg. of $K \subset \overline{K}$ is equal to 0.

Theorem 1. For any field *K* its algebraic closure $\overline{K}$ exists and is unique up to isomorphism.

Definition 5. A function $f{:}R \to R$ is called polynomially checkable (PC) if there exists an integer *k*, such that for any $a_1,...,a_k \in R$ the functions $f_0(x) = f(x), f_1(x) = f(x + a_1),..., f_k(x) = f(x + a_k)$ are algebraically dependent, i.e. there exists a polynomial $P \in R[T_0,...,T_k]$, such that $P(f_0,...,f_k) = 0$ (for any $x \in R$). The polynomial *P* is called a *checking polynomial* of the function *f*.

The computation of a PC function can be readily verified. For a given value of *x*, denote by $\tilde{f}_0, \tilde{f}_1,...., \tilde{f}_k$ the values of *f* at the points $x, x + a_1,..., x + a_k$, respectively. Then if all the values are computed correctly, the following equality holds:

$$P(\tilde{f}_0, \tilde{f}_1,..., \tilde{f}_k) = 0 \qquad (5)$$

This property provides a unified approach to the problem of error detection/correction in computation of numerical functions. Indeed we can consider inequality similar to $(2')$

$$\left|P(\tilde{f}_0, \tilde{f}_1,..., \tilde{f}_k)\right| \le \delta, \qquad (5')$$

where δ is a small positive number specified by the precision of the computation. In case of correct computation $(5')$ is satisfied. We have to note, however, that even if $(5')$ is satisfied it doesn't give us 100% warrantee that computation is correct. There are some faults that cannot be detected by $(5')$.

The first class of faults (we can call them software faults) are result of the fact that some other PC function $g(x) \ne \mathrm{f}(x)$ can have the same checking polynomial. For instance if $g(x) = \mathrm{f}(x + b)$, where b is a constant, then g(x) and f(x) have the same checking polynomials. Preliminary results show that a PC function with bounded spectrum is uniquely defined by its checking polynomial (the set of shifts is fixed) and its values at a finite set of points. This property can be used to fight the software faults.

The second class of faults which can not be detected by using $(5')$ are hardware faults. They are result of physical defects of a device which performs the calculation of function. Random faults are hardware faults. The fault coverage of random faults is calculated below for an important case. It is shown in (Abamowitz and Stegun 1965) that the class of PC functions is very broad even for a small *k*.

Denote by S the set of three functions: $x, e^x, \sin x$. Let $A \subseteq S$; denote by R(A) the field of all rational functions in $\boldsymbol{g}_j \in \boldsymbol{A}$ and by $\overline{R(A)}$ its algebraic closure.

Example 5. a) A = (x), R(A) = $\left\{\frac{P_i(x)}{Q_i(x)}\right\}$, where P_i, Q_j are polynomials of one variable with real coefficients. Its algebraic closure $\overline{R(A)}$ includes, as a special case any function *g(x)* which is a solution of an equation $P_n(x)g^n(x) + P_{n-1}(x)g^{n-1}(x) + ... + P_0(x) = 0$ where $P_i(x), i = 0,1,..., n$ are polynomials of one variable with real coefficients.

In particular, $\overline{R(A)}$ includes the set of all functions that can be obtained by application of finite number of additions, subtractions, multiplications, divisions, and raising to a rational power to the function $g(x) = x$.

b) $A = \{e^x, \sin x\}$; *R(A)* = $\frac{P_i(e^x, \sin x)}{Q_i(e^x, \sin x)}$, where P_i, Q_j are polynomials of two variables with real coefficients.

Theorem 2 Let $f: R \to R$ belong to the field $\overline{R(A)}, A \subseteq (x, e^x, \sin x)$. Then *f* is polynomially checkable with $k = |A|$.

Proof. We prove the theorem for the case $A = \{x, e^x, \sin x\}$. For the other cases the proof is analogous.

Let $f(x) \in \overline{R(x, e^x, \sin x)}$ and $a_1, a_2, a_3 \in R$; denote: $f_0(x) = f(x)$, $f_1(x) = f(x + a_1)$, $f_2(x) = f(x + a)$, $f_3(x) = f(x + a_3)$. We have to show that $f_0, ..., f_3$ are algebraically dependent. This follows from the statements:

1) Tr.deg of $R \subset \overline{R(x, e^x, \sin x)}$ equals to 3.
2) For every $a \in R, f(x + a) \in \overline{R(x, e^x, \sin x)}$.

Indeed $f(x) \in \overline{R(x, e^x, \sin x)} \Leftrightarrow$ there exists a polynomial $A \in R(x, e^x, \sin x)[T]$ such that $A(f) = A_n(x)f(x) + ... + A_1(x)f(x) + A_0(x) = 0$, where $A_i(x) \in R(x, e^x, \sin x)$ .Let us denote $\rho(x) = A_n(x)f(x) + ... + A_0(x)$ .Then

$\rho(x + a) = A_n(x + a)f(x + a) + ... + A_0(x + a) = 0, A_i(x + a) \in R(x, e^x \sin x, \cos x)$.

Hence $f(x + a) \in \overline{R(x, e^x, \sin x, \cos,)}$. *But* $\overline{R(x, e^x, \sin x, \cos x)} = \overline{R(x, e^x, \sin x)}$,

hence $f(x+a) \in \overline{R(x,e^x,\sin x)}$ and, therefore, $f_0,f_1,f_2,f_3 \in \overline{R(x,e^x,\sin x)}$.

But the Tr.deg. of $R \subset \overline{R(x,e^x,\sin x)}$ equals to 3, hence f_0,f_1,f_2,f_3 are algebraically dependent.

Let f be the result of application of a finite number of additions, subtractions, multiplications, divisions and raising to a rational power to the following functions:
Const, $x, e^x, \sin(r_i x + b_i), \cos(r_j x + b_j)$, where r_i, r_j are rational numbers.
Then f is a PC function with $k \le 3$.

Example 6. The function $f(x) = \dfrac{(\sin(\frac{x}{11}+\frac{\pi}{7}) + e^x)^{\frac{3}{5}} + x^2\cos^4 x}{x^5 + (x^4 + x^2(\sin 2x + xe^x)^3)^{\frac{1}{3}}}$ is a PC function with k=3.

Example 7. Consider the function

$$f(x) = \frac{((\sin x)^{\frac{1}{3}} + \cos x)^{\frac{1}{2}}}{\sin(x+7) - (\cos 3x \sin(3x+5) - 4)^{\frac{1}{17}}}; f(x) \in \overline{R(\sin x)}$$

Tr.deg. of extension $R \subset \overline{R(\sin x)}$ equals to 1, therefore $f(x)$ is a PC function with k=1.
Note. The theorem 2 states that the class of PC functions is very big. We have to note, however, that a number of commonly used functions like ln(x), $\sin^{-1}(x), \cos^{-1}(x)$ are non PC functions.

3. Finding a Checking Polynomial by Least Square Estimation

To find a checking polynomial we consider the following optimization problem. Let

$$f : [A, B] \to R$$

Denote

$$\delta(\beta_0, \alpha_1, \ldots, \alpha_k) = \int_A^B 2(f(x) - \alpha_1 f(x + a_1) - \ldots - \alpha_k f(x + a_k) - \beta_0)^2 dx$$

Find such $\alpha_1, \ldots, \alpha_k, \beta_0$, that $\delta(\beta_0, \alpha_1, \ldots, \alpha_k)$ takes minimal value. To solve this problem consider the following equations:

$$\frac{\partial}{\partial \beta_0}\delta = \int_A^B 2\left(f(x) - \alpha_1 f_1 - \ldots - \alpha_k f_k - \beta_0\right)(-1)dx = 0$$

$$\frac{\partial}{\partial \alpha_1}\delta = \int_A^B 2\left(f(x) - \alpha_1 f_1 - \ldots - \alpha_k f_k - \beta_0\right)f_1 dx = 0$$

$$\vdots$$

$$\frac{\partial}{\partial \alpha_k}\delta = \int_A^B 2\left(f(x) - \alpha_1 f_1 - \ldots - \alpha_k f_k - \beta_0\right)f_k dx = 0$$

Let us denote by $\langle f, g\rangle = \int_A^B (f \cdot g)dx$. Using this notation, we can express the system of equations in the form

$$\langle f_0, 1\rangle = \alpha_1\langle 1, f_1\rangle + \ldots + \alpha_k\langle 1, f_k\rangle + \beta_0(B - A)$$

$$\langle f_0, f_1\rangle = \alpha_1\langle f_1, f_1\rangle + \ldots + \alpha_k\langle f_1, f_k\rangle + \beta_0\langle f_1, 1\rangle$$

$$\langle f_0, f_2\rangle = \alpha_1\langle f_2, f_1\rangle + \ldots + \alpha_k\langle f_2, f_k\rangle + \beta_0\langle f_2, 1\rangle$$

$$\vdots$$

$$\langle f_0, f_k\rangle = \alpha_1\langle f_k, f_1\rangle + \ldots + \alpha_k\langle f_k, f_k\rangle + \beta_0\langle f_k, 1\rangle$$

Solving this system we obtain $\beta_0, \alpha_1, \ldots, \alpha_k$. If $\delta(\beta_0, \alpha_1, \ldots, \alpha_k) = 0$ then f is an LC function with the checking polynomial

$$f_0 - \alpha_1 f_1 - \ldots - \alpha_k f_k - \beta_0 = 0$$

If $\delta(\beta_0, \alpha_1, \ldots, \alpha_k) = \delta \neq 0$ then f does not have a checking polynomial of degree 1. However, if δ is a small number the formula

$$\left|\tilde{f}_0 - \alpha_1 \tilde{f}_1 - \ldots - \alpha_k \tilde{f}_k - \beta\right| \leq \delta$$

can be used to verify the correctness of computations. A similar method can be used for obtaining a checking polynomial of degree > 1.

Other methods for finding a checking polynomial are described in (Vainstein 1998).

4. Numerical Results

A Matlab program was developed to implement the algorithm described using techniques from linear algebra. This program can determine the coefficients of the checking polynomial, $\beta_0, \alpha_1, \ldots, \alpha_k$, for various numerical functions.

From the system of equations defined above, denote

$$A = \begin{pmatrix} \langle 1, f_1 \rangle & \cdots & \langle 1, f_k \rangle & (B - A) \\ \langle f_1, f_1 \rangle & \cdots & \langle f_1, f_k \rangle & \langle f_1, 1 \rangle \\ \langle f_2, f_1 \rangle & \cdots & \langle f_2, f_k \rangle & \langle f_2, 1 \rangle \\ \vdots & \vdots & \vdots & \vdots \\ \langle f_k, f_1 \rangle & \cdots & \langle f_k, f_k \rangle & \langle f_k, 1 \rangle \end{pmatrix}.$$

Define vector X as

$$X = \begin{pmatrix} \beta_o \\ \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix}.$$

And, define vector B as

$$B = \begin{pmatrix} \langle f_0, 1 \rangle \\ \langle f_0, f_1 \rangle \\ \langle f_0, f_2 \rangle \\ \vdots \\ \langle f_0, f_k \rangle \end{pmatrix}.$$

First, the program computes the coefficients of matrix A and vector B using the trapezoidal integration method. Then, the equation $AX = B$ is solved by a reduced row echelon form method. The resulting values of vector X then give us the coefficients of the checking polynomial, $\beta_0, \alpha_1, \ldots, \alpha_k$.

The values of $\beta_0, \alpha_1, \ldots, \alpha_k$ are then used to evaluate $\delta(\beta_0, \alpha_1, \ldots, \alpha_k)$ as described above. This gives the value of δ.

In order to investigate the effect of increasing *k*, the program finds the values of δ for a given range of k values and determines which of these produces the minimum deviation. Shown below are the results from these computations for various numerical functions.

In Table 1 and Figure 1 we see the algorithm's results when applied to $f(x)=\ln(x)$.

Function	$\ln(x)$
Best k	9
Accuracy, δ	$1.063*10^{-6}$
α_1	13.636083507129
α_2	-48.3372588311351
α_3	33.8356474758064
α_4	104.402870905685
α_5	-173.871357832086
α_6	19.6783752309016
α_7	95.3406000928846
α_8	-40.7070503540183
α_9	-2.97970571433605
β_o	0.0104815654867488
Stepsize, h	0.0001
Lower limit, A	1
Upper limit, B	100

Table 1. Best results for $f(x)=\ln(x)$.

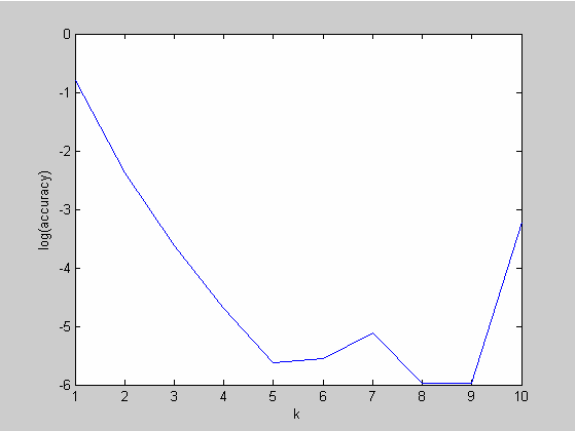


Figure 1. Graph of the accuracy δ versus k for $f(x)=\ln(x)$.

These results show that, in the interval $[1,100]$, and with a step size of $h = 0.0001$, the checking polynomial of $f(x) = \ln(x)$ that returns the best accuracy, or minimum deviation δ, is the polynomial with values for $\beta_0, \alpha_1, \ldots, \alpha_k$ as given in Table 1.

As a result of the computational experiments we observed that, as a rule, the deviation, δ, is decreasing with increasing *k*, for small values of *k*. However, as *k* continues to increase δ eventually begins to increase. The reason for this increase is the limited accuracy of computer arithmetic. In general, we are interested in the smallest value of *k* (since the overhead increases with *k*) that provides us with a satisfactory deviation.

As another example, let's consider a more complicated looking numerical function

$$f(x) = \cos\left(\sin(x) - \sqrt{x}\right) - \cos(x) + \sqrt{x} \tag{6}$$

The graph δ versus *k* for this function is shown below in Figure 3.

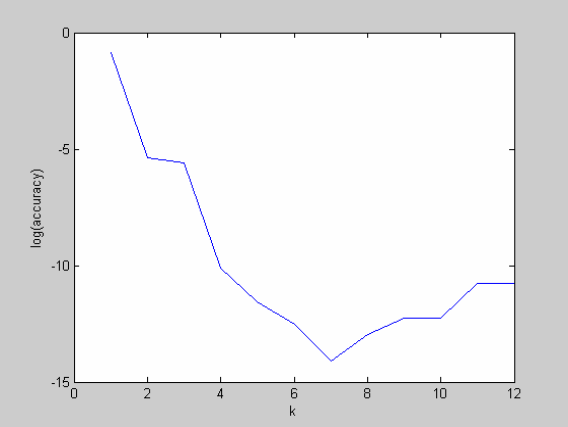


Figure 3. Graph of the accuracy δ versus k for $f(x) = \cos\left(\sin(x) - \sqrt{x}\right) - \cos(x) + \sqrt{x}$.

We note in the results of Figure 3 the two important features mentioned above: the deviation initially decreases but then eventually increases with increasing *k*.

Given in the table below are some other sample results.

Function	*k*	δ	$\alpha_1, \alpha_2, \ldots, \alpha_k$	β
$\cos(x)$	2	$1.5655594 * 10^{-26}$	1.08060461 -0.999999999	- $4.6509012 * 10^{-15}$
x^2	2	$5.3305333 * 10^{-20}$	2.00000000 -1.00000000	2.0000000
$\sqrt{x}$	6	$3.5165389 * 10^{-9}$	17.6933009	-0.73834646

			-82.75929 138.617962 -51.0106681 -69.7915475 48.319250	
e^x	2	$5.4703917*10^{-22}$	0.00810684622 0.132352941	$-5.80812*10^{-12}$
$\log\left[x+\left(x^2+1\right)^{\frac{1}{2}}\right]$	3	$1.8593950*10^{-10}$	1.67126425 3.81365972 -5.13358327	2.2870721
$\cos\left(\sin(x)-\sqrt{x}\right)-\cos(x)+\sqrt{x}$	7	$7.6541494*10^{-15}$	-1.16902598 -0.58592973 -0.74339752 -1.48341738 -0.454274819 -0.230924645 0.684256990	7.6417875

Table 3. Sample results of the least square estimation method. All results for step size, $h = 0.0001$

These results in Table 3 give the values of $\beta_0, \alpha_1, \ldots, \alpha_k$ that define the checking polynomial of the form

$$f_0 - \alpha_1 f_1 - \ldots - \alpha_k f_k - \beta_0 = 0$$

for each numerical function.

A number of other common and specialized numerical functions were also tested. The results are shown in Table 4 below. Each of these functions were tested over a domain interval for which they are well behaved.

Function	k	δ
Airy Function: $A_i(z)=\frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\left(zt+\frac{t^3}{3}\right)}dt$; for $i=1$	6	$8.01855604514853*10^{-13}$
Bessel Function of 1st Kind: $J_1(z)=\sum\frac{(-1)^k}{\Gamma(k+\nu+1)k!}\left(\frac{z}{2}\right)^{2k+\nu}$	7	$6.77944398369068*10^{-12}$
Beta Function: $B(a,a)=\int_0^1 t^{a-1}(1-t)^{a-1}dt=\frac{\Gamma(a)\Gamma(a)}{\Gamma(a+a)}$	5	$1.94247036365353*10^{-9}$

Scaled Complementary Error Function: $f(x)=e^{x^2}\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt$	6	$5.53736923276668*10^{-11}$
Exponential Integral: $f(x)=\int_x^\infty \frac{e^{-t}}{t}dt$	5	$1.09725876920181*10^{-10}$
Logarithm of the Gamma Function: $\log\Gamma(z)=\sum_{k=1}^{\infty}\left(\frac{z}{k}-\log\left(1-\frac{z}{k}\right)\right)-\gamma z-\log(z)$	5	$2.24588944104907*10^{-7}$
Inverse Cosine: $\cos^{-1}(z)=\frac{\pi}{2}+i\log\left(iz+\sqrt{1-z^2}\right)$	6	$9.57918288982022*10^{-6}$
Inverse Hyperbolic Cosine: $\cosh^{-1}(z)=\log\left(z+\sqrt{z-1}\sqrt{z+1}\right)$	4	$5.68478815529586*10^{-9}$
Hyperbolic Cosine: $\cosh(z)=\frac{e^z+e^{-z}}{2}$	2	$1.63905802795885*10^{-21}$
Inverse Tangent: $\tan^{-1}(z)=\frac{i}{2}\log\left(\frac{i+z}{i-z}\right)$	5	$2.2975351543044*10^{-6}$
Complete Elliptic Integral of the First Kind: $K(z)==\int_0^1 \frac{1}{\sqrt{1-t^2}\sqrt{1-zt^2}}dt$	4	$1.29605146674611*10^{-11}$
Riemann Zeta Function: $\zeta(s)==\sum_{k=1}^{\infty}\frac{1}{k^s}$	3	$5.31429308150972*10^{-9}$
Dawson's Integral: $F(x)=e^{-x^2}\int_0^x e^{-t^2}dt$	7	$1.48383522053584*10^{-6}$
Fresnel Sine Integral: $S(x)=\int_0^x \sin\left(\frac{\pi}{2}\cdot t^2\right)dt$	6	$4.04417425710405*10^{-5}$

Table 4. Results for various functions (Abamowitz and Stegun 1965; Wolfram 1999).

5. Conclusions and Future Work

We have demonstrated that checking polynomials can be effectively used for fault tolerant computations. In particular, checking polynomials for some common numerical functions and some specialized functions have been found.

A program was developed in Matlab that allow us to obtain an "approximate" checking polynomial for a wide range of numerical functions.

The examples considered showed that even for functions that do not appear simple an approximate checking polynomial provides a small value of deviation, δ .

A future paper will describe a hardware implementation of this fault tolerance technique. Issues related to computational overhead and comparisons to overhead incurred by other methods will be discussed.

We will also consider the problem of obtaining checking polynomials of degree greater than one. Once this theoretical foundation is established an approach such as that outlined in this paper will be developed for finding coefficients of higher degree checking polynomials.

References

Abramowitz, M. and I.A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, Applied Math. Series #55, Dover Publications, 1965.

Ercegovac, Milos D., Tomás Lang, Jean-Michel Muller, and Arnaud Tisserand. "Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers." *IEEE Transactions on Computers*, 49(7):628-637, July 2000.

Koren, Israel. *Computer Arithmetic Algorithms*. A K Peters, 2001.

Lala, Parag K. *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2001.

Lang, S., *Algebra*, Addison-Wesley Publishing Co., 1992.

Muller, Jean-Michel. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, 1997.

Vainstein, Feodor. "Algebraic Methods in Hardware/Software Testing." Ph. D. Thesis, Boston University, 1993.

Vainstein, F. S. "Self Checking Design Technique for Numerical Computations." *VLSI Design*, 1998, Vol. 5, No. 4, pp. 385-392.

Wolfram, Stephen, *The Mathematica Book, 4th Ed.*, Cambridge University Press, 1999.

Wong, W.F. and Goto, E. "Fast hardware-based algorithms for elementary function computations using rectangular multipliers." *IEEE Transactions on Computers*, 43(3):278-294, March 1994.