A Revised Presentation of the CENA Method

Philippe Langlois

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Revised Presentation of the CENA Method

Philippe LANGLOIS

No 4025

Octobre 2000

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4029--FR

INRIA
RHÔNE-ALPES

A Revised Presentation of the CENA Method

Philippe LANGLOIS *

Thème 2 — Génie logiciel
et calcul symbolique
Projet Arénaire

Rapport de recherche n° 4025 — Octobre 2000 — 28 pages

Abstract: The CENA method is a new automatic method to correct the first-order effect of floating point rounding errors on the result of a numerical algorithm. A first presentation of this method is proposed in the previous report RR-3828 (december 1999). The current report completes and replaces the previous one. We hark back to the presentation of the CENA method exhibiting new important aspects. We illustrate the pros-and-cons of the method solving triangular linear systems. These systems are designed to generate a significantly inaccurate computed solution. We propose a new analysis of the analogy and discrepancy between the linear correction and computing with extented precision.

Key-words: Automatic error analysis, rounding error, floating point arithmetic, accuracy, stability.

(Résumé : tsvp)

* Part of this research was done while the author was a member of the IREMIA Laboratory at the University of La Réunion (France) and was visiting the Department of Mathematics at the University of Manchester (England) during his C.R.C.T. period.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International: +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

Une Nouvelle Présentation de la Méthode CENA

Résumé : La méthode CENA est une nouvelle méthode automatique de correction de l'effet d'ordre un des erreurs d'arrondi introduites par l'arithmétique flottante. Une première description de la méthode CENA fait l'objet du précédent rapport RR-3828 (décembre 1999). Le présent rapport complète et remplace le précedent. Nous revenons sur la présentation de la méthode CENA en exhibant certains aspects importants jusque-là non décrits. Nous illustrons les avantages et les limites de la méthode par la résolution de systèmes linéaires triangulaires. Ces systèmes sont spécialement définis pour que les solutions calculées soient significativement imprécises. Nous proposons une nouvelle analyse des analogies et des différences entre correction linéaire et calcul en précision étendue.

Mots-clé : Analyse d'erreur automatique, erreur d'arrondi, arithmétique flottante, précision, stabilité.

1 Introduction

The rounding errors introduced by floating point arithmetic contribute to the inaccuracy of computed results and to the instability of some numerical algorithms. In this paper, we focus on the propagation of these rounding errors. Other sources of error such as data errors and truncation errors are not considered here. A detailed analysis of rounding errors in the implementations of large algorithms is complex and tedious. So automatic approaches — that use the computer — have been developed to complement the classic manual analysis. Differential methods are well-known examples of such automatic methods.

Differential methods rely on the computation of the partial derivatives of the global forward error with respect to the elementary rounding errors. The *elementary rounding error* is the error introduced by each floating point arithmetic operation, and the *global forward (rounding) error* is the accumulated effect of these elementary rounding errors on the final result.

For example, let us consider the floating point evaluation $\widehat{f}$ of a real function f at the data $X = (x_1, \ldots, x_n)$ where intermediate variables $\widehat{x}_{n+1}, \ldots, \widehat{x}_{N-1}$, are computed to yield the result $\widehat{x}_N$. The global forward error satisfies

$$\widehat{x}_N - f(X) = \Delta_L - E_L,$$

where

$$\Delta_L = \sum_{k=n+1}^{N} \frac{\partial \widehat{f}}{\partial \delta_k}(X, \delta) \cdot \delta_k,$$

is the first-order approximate of the global rounding error with respect to the absolute elementary rounding errors $\delta = (\delta_{n+1}, \ldots, \delta_N)$ generated by the computation of corresponding intermediate variables. The last term E_L in the the global forward error relation is the *linearization error* associated with Δ_L.

The relative elementary rounding errors $\delta_k/|\widehat{x}_k|$ are bounded with the floating point precision. Hence, differential methods are generally used to compute a first-order bound for the global forward error [14], [13], [8]. However such computed bounds may be too large to be useful [3]. First-order statistical estimates of the global forward error are also derived from a statistical modeling of the elementary rounding errors [20], [12], [13], [8]. Both these bounding and statistical approaches neglect the terms of order higher than one with respect to the elementary rounding errors in the global forward error, *i.e.*, the linearization error E_L.

In practice, automatic or algorithmic differentiation techniques provide a general applicability and an efficient implementation of these differential methods [6].

We present here a new differential method where the elementary rounding errors δ_k are neither bounded nor described by a stochastic model, but *computed*. Hence, we compute the

first-order term of the global forward error with respect to the elementary rounding errors $\widehat{\Delta}_L = fl\,(\Delta_L)$. We note $fl\,(\Delta_L)$ the floating point computation of Δ_L. Using this term as a correcting factor to the computed result $\widehat{x}_N$, we derive the *corrected result*

$$\overline{x_N} = fl(\widehat{x}_N - \widehat{\Delta}_L).$$

Such a linear correction is particularly suitable when the global forward error is dominated by the first-order terms, for example in the *linear algorithms* we define in Section 3.3. Even if complex algorithms are not linear, the accuracy of computed solutions may rely on linear parts of the algorithm. Thus, we can apply this correction to final results or sensitive intermediate variables to improve the accuracy of the results and the stability of the algorithm.

This computed linear correction term also suffers from a truncation error and rounding errors. The remaining difference between the corrected result $\overline{x_N}$ and the exact result $f(X)$ is measured by a *residual error*. For linear algorithms, we compute an absolute bound B_{E_C} of this residual error that provides a confidence threshold associated to the corrected result. We have

$$|\overline{x_N} - f(X)| \leq B_{E_C}.$$

We name this approach the *CENA method*. CENA is the French acronym of *Correction des Erreurs Numériques d'Arrondi*, that means Correcting Numerical Rounding Errors. Linnainmaa has proven that the first-order approximation of the global forward error is an efficient tool for experimental analysis of the numerical stability of algorithms [13]. The CENA method extends Linnainmaa's error linearization method providing the computation of a correcting factor for the global forward error and a bound for the residual error of the corrected result. This correction is based on the numerous results on elementary error computation obtained between 1965–1975 by several authors, particularly Dekker [5], Kahan, Knuth [9] and Pichat [16].

Langlois and Nativel have given an introductory description of this approach in [10] (in French) and [11]. The aim of this paper is to present a complete and definitive description of the CENA method. Using Wilkinson's running error analysis, we also derive here a new dynamic bound for the residual error that improves the previous *a priori* result given in [10].

The paper is organized as follows. In the next section 2, we discuss introductory examples chosen to illustrate the pros and cons of the method (the examples are detailed in the Appendix). The main section of the paper is Section 3 where we present the principles of the CENA method, linear algorithms and the computation of the residual error bound. More technical aspects are presented in the next two sections 4 and 5 that may be skipped at the first reading. These sections respectively deal with the computation of the elementary rounding errors and the partial derivatives of the computed result with respect to these elementary rounding errors. We terminate summarizing the algorithm of the CENA method in Section 6.

2 Pros and Cons of the CENA Method from Introductory Examples of Application

We illustrate the general behavior of the CENA method solving two ill conditioned triangular linear systems. Using IEEE-754 floating point arithmetic and the forward or backward substitution algorithm, we compute the single precision solution and then its correction with the CENA method. We compare this single precision corrected value with the solution computed with the double precision.

The classic substitution algorithm is backward stable. It is well-known that the global forward error it generates is usually very small even for ill conditioned triangular linear systems [7]. The following two triangular systems are carefully defined to generate large forward errors in binary floating point arithmetic; we describe these systems in Appendix.

2.1 Recovering the Exact Solution

The first linear system is $U_\alpha x = b$, where U_α, an upper triangular matrix of order n, depends on the integer parameter α. All entries in U_α, b and x are integer values. Diagonal entries in U_α are ones and some others are $\pm 2^\alpha$. The matrix U_α is ill conditioned (see Appendix 7.1 for details). The Bauer-Skeel componentwise condition number of the matrix U_α satisfies $\mathrm{cond}(U_\alpha) = \| \, |U_\alpha^{-1}||U_\alpha| \, \|_\infty = O(n2^\alpha)$. The solution of this system is the unit vector

$$x = (1, 1, \ldots, 1)^T.$$

We solve $U_\alpha x = b$ with back substitution, *i.e.*, $x(i) = b(i) - \sum_{j=i+1}^{n} U_\alpha(i,j)x(j)$, $i = n, [-1], 1$. We choose values for α such that catastrophic cancellations appear in the previous summation. It is not surprising that the computed solutions in IEEE-754 single and double precision suffer from large forward errors. Computing with single or double precision yields

$$\widehat{x} = (0, 0, \ldots, 0, 1, 1)^T,$$

when respectively, $\alpha > 24$ and $\alpha > 53$. The first (n-2) entries of the computed solutions has no significant digit. It is worth noting that both the single and double precision solutions suffer from a similar forward error but for different threshold values. Increasing the precision in this computation is not sufficient to yield the exact result.

Applying the CENA method to the previous single precision computation, we provide the corrected result $\overline{x}$ together with an absolute bound B_{E_C} of the residual error, *i.e.*, $|\overline{x} - x| \leq B_{E_C}$.

The corrected result is

$$\overline{x} = (1, 1, \ldots, 1)^T,$$

for all acceptable values of α, *i.e.*, the values such that the correction process does not overflow. Overflow appears for values larger than both the previous single and double precision thresholds. Correcting the single precision result yields $\overline{x} = x$ while $\alpha \leq 115$ (2^α

overflows when $\alpha > 128$ in IEEE single precision). Therefore the expected solution x is recovered correcting the single precision computation for a large range of α values.

The absolute computed bound B_{EC} grows as $O(2^{\alpha n/2}\mathbf{u})$ where $\mathbf{u}$ denotes the precision of the computation. So this computation returns large values and may overflow. In this example, the computed bound B_{EC} is very pessimistic since the residual error is actually $|\overline{x} - x| = 0$.

2.2 Improving the Accuracy and *a posteriori* Control

Now we consider the triangular system $L_\alpha x = b_\alpha$, where L_α is a lower triangular matrix of order n. The coefficients of the matrix L_α and the right-hand side b_α depend on an integer parameter α (see Appendix 7.2 for details). We choose α such that no data error exists in L_α and b_α, *i.e.*, all components are single precision floating point numbers. The componentwise condition number of the matrix L_α, $\mathrm{cond}(L_\alpha) = \| \, |L_\alpha^{-1}||L_\alpha| \, \|_\infty$, grows exponentially with n. The solution x is such that one of its components is not a finitely representable binary floating point number ($x(1) = 0.01$). As the matrices L_α are ill conditioned even for small values of n and α, solving $L_\alpha x = b_\alpha$ with the substitution algorithm yields significantly large forward errors. The relative forward error of the computed solution $\widehat{x}$ satisfies

$$\mathrm{FE}(\widehat{x}) = \max_{1 \le i \le n} \frac{|\widehat{x}(i) - x(i)|}{|x(i)|} = \frac{|\widehat{x}(n) - x(n)|}{|x(n)|}. \tag{1}$$

We solve the triangular systems $L_\alpha x = b_\alpha$, for $\alpha = 224$ and dimensions from 1 to 10. We summarize with Figure 1 the results presented hereafter and the classic estimated accuracy bound $\mathbf{u} \times \mathrm{cond}(L_\alpha)$.

The substitution algorithm is first performed in IEEE-754 single precision ($\mathbf{u}_s \approx 5.96 \times 10^{-8}$). As expected, the forward error $\mathrm{FE}(\widehat{x})$ increases exponentially with the dimension n. No significant digit in $\widehat{x}$ is returned when $n \ge 5$. For example we have $\mathrm{FE}(\widehat{x}) \approx 1.32$ for $n = 5$.

Then the same resolution is performed with IEEE-754 double precision ($\mathbf{u}_d \approx 1.11 \times 10^{-16}$). The computed solution is denoted $\widehat{y}$. The forward error is twice better than the previous computation but of course exhibits the same behavior controlled by the exponential rate of the condition number. About half the accuracy in $\widehat{y}$ is lost for $n = 5$ ($\mathrm{FE}(\widehat{y}) \approx 2.95 \times 10^{-8}$), and the result has no significant digit for $n \ge 9$.

We apply the CENA method correcting the single precision $\widehat{x}$. The corrected result $\overline{x}$ has a better accuracy than the single precision result $\widehat{x}$, but this improvement is not uniform with respect to n. We distinguish two steps. For $n \le 4$, the corrected result $\overline{x}$ is about as accurate as its single precision computation allows (we recall that the exact solution x suffers from an unavoidable relative error of order $\mathbf{u}$). We have $\mathrm{FE}(\overline{x}) \approx 2.2 \times 10^{-8}$, for $n = 4$. For $n \ge 5$, the relative forward error $\mathrm{FE}(\overline{x})$ increases roughly as the forward error of the double precision computation (IEEE-754 double precision arithmetic is slightly more precise than twice the single precision).

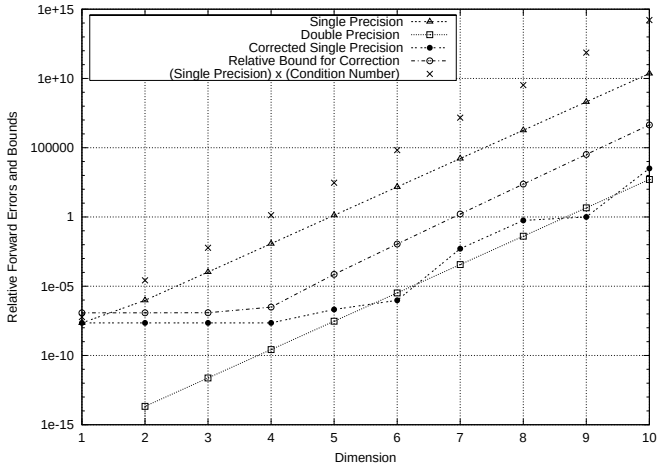

Figure 1: Solving $L_\alpha x = b_\alpha$, for $\alpha = 224$ and dimensions 1 to 10 .

The computed bound B_{E_C} validates the accuracy of the corrected results. We verify that the relative value $B_{E_C}/|x|$ bounds $\mathrm{FE}(\overline{x})$, the actual forward error of the corrected result. As for the forward error, the bound B_{E_C} is tight for $n \leq 4$, and then over-estimates the actual error by a factor of 5×10^3 in this example.

In Corollary 2, we prove that the scalar solution $x(i)$ satisfies $x(i) \in [\overline{x(i)} - B_{E_C}, \overline{x(i)} + B_{E_C}]$, that may provide an *a posteriori* control of the non-corrected result $\widehat{x}(i)$. For example when $n = 3$,

$$\begin{aligned} \widehat{x}(3) &= 2.000\,2127 \times 10^{-2}, \\ \overline{x}(3) - B_{E_C} &= 1.999\,9997 \times 10^{-2}, \text{and} \\ \overline{x}(3) + B_{E_C} &= 2.000\,0002 \times 10^{-2}, \end{aligned}$$

prove that the four last digits of $\widehat{x}(3)$ are not accurate. When $n = 7$,

$$\begin{aligned} \widehat{x}(7) &= 5.259\,5762 \times 10^{3}, \\ \overline{x}(7) - B_{E_C} &= -2.078\,7549 \times 10^{-1}, \text{and} \\ \overline{x}(7) + B_{E_C} &= 8.445\,9424 \times 10^{-1}, \end{aligned}$$

yield that $\widehat{x}(7)$ has no significant digits. This *a posteriori* control agrees with the actual relative forward error of the initially single precision computed solution for the considered values of n.

2.3 Discussion

These two examples exhibit the main properties of the CENA method when it is applied to correct the final result of a computation in a given precision (here, the IEEE single precision).

The first example illustrates the cases when correcting the single precision process yields the exact result whereas the double precision computation still returns inaccurate results. This kind of exact correction depends strongly on the computation order, that is a general fact for floating point computations. Such favorable cases excepted, the general efficiency of the CENA method is emphasized with the second example. The efficiency of the single precision corrected process is controlled by both the single and the double precisions.

While the relative accuracy of the double precision computed result is smaller than the single precision, the relative accuracy of the single precision result corrected result is of the order of this single precision, *i.e.*, the best possible accuracy for a computation in single precision. The range of such behavior of course depends on the condition of the problem. When the problem is so ill conditioned that the relative accuracy of the double precision computed result is worse than the single precision, the corrected single precision result is about as (in)accurate as this double precision computed result. In both cases, the corrected result is more accurate than the initially non-corrected result.

Therefore the correction process is globally similar to doubling the precision of intermediate computations. Nevertheless, the corrected algorithm is not exactly equivalent to the initial algorithm processed in double precision. Intermediate computed values in the corrected algorithm are not twice more accurate than the initially computed ones (they are still computed in the current precision). Only the initially computed final result is modified by the summation of the correcting term $\widehat{\Delta}_L$. So the corrected results may be different of the results computed with extended precision. The first example illustrates such a favorable case.

The computing error bound B_{E_C} validates the accuracy of the corrected result and may provide an *a posteriori* control of the initially computed result. The second example illustrates these two properties. The relative error bound $B_{E_C}/|x|$ agrees with the actual accuracy of the corrected result[1]. We also use the corrected result $\overline{x}$ and the bound B_{E_C} to prove that the computed result $\widehat{x}$ was inaccurate.

Nevertheless, this bounds suffers from the well-known limitations of automatic forward error bounds. We describe in Section 3.4 that the computation of B_{E_C} mainly relies on the classic Wilkinson's running error analysis [21]. So pessimistic overestimates of the actual error may be returned, for instance when the problem is ill conditioned or when the algorithm computes intermediate quantities with absolute values larger than the final results. The first linear system illustrates this latter scaling effect. The second example and associated Figure 1 exhibit the effect of the condition of the system.

[1] The bound B_{E_C} is applied as an absolute bound to the corrected result since the exact solution is usually unknown!

3 The CENA Method

3.1 Floating Point Computation

Let $\mathbb{F}$ be a set of floating point numbers with precision $\mathbf{u}$. Elementary floating point operations that correspond to arithmetic operations $(+, -, \times, /, \sqrt{})$ are defined on $\mathbb{F}$. We assume the standard model of floating point arithmetic is satisfied [7]. For $x, y \in \mathbb{F}$, and $\circ \in \{+, -, \times, /, \sqrt{}\}$, we define $z = x \circ y$ and $\widehat{z} = fl(x \circ y)$. Evaluated expressions are denoted by $fl(\cdot)$, and computed quantities classically wear a hat. With the standard model of floating point arithmetic,

$$\widehat{z} = z(1+\zeta), \quad |\zeta| \leq \mathbf{u}, \tag{2}$$

and similarly,

$$\widehat{z} = z/(1+\zeta), \quad |\zeta| \leq \mathbf{u}. \tag{3}$$

Let f be a function of n real variables and $fl(f) = \widehat{f}$, its numerical evaluation on $\mathbb{F}$. For simplicity, we assume that f is a real function — a vector function will be considered componentwise. The computation of $x_N = f(X)$ returns $\widehat{x}_N$ for $X = (x_1, \ldots, x_n)$ belonging to $\mathbb{F}$. The global forward error is $\widehat{x}_N - f(X)$. The computation of each intermediate variable $\widehat{x}_k$ introduces an (absolute) elementary rounding error δ_k for $k = n+1, \ldots, N$. For $\widehat{x}_i$, $\widehat{x}_j$ and $\widehat{x}_k$ in $\mathbb{F}$ such that $1 \leq i \leq j < k \leq N$, and $\circ \in \{+, -, \times, /, \sqrt{}\}$,

$$\begin{aligned} \widehat{x}_k &= fl(\widehat{x}_i \circ \widehat{x}_j) \\ &= \widehat{x}_i \circ \widehat{x}_j + \delta_k \qquad (4) \\ &= x_k + \delta_k, \qquad (5) \end{aligned}$$

where $\widehat{x}_j \neq 0$ when $\circ = /$, and $\widehat{x}_i = 0$, $\widehat{x}_j \geq 0$ when $\circ = \sqrt{}$.

So, the numerical evaluation of $f(X)$ on $\mathbb{F}$ is $\widehat{f}(X, \delta)$, that is, a function of the data X and the elementary rounding errors $\delta = (\delta_{n+1}, \ldots, \delta_N)$. A first-order approximate of the global rounding error with respect to the elementary rounding errors is

$$\Delta_L = \sum_{k=n+1}^{N} \frac{\partial \widehat{f}}{\partial \delta_k}(X, \delta) \cdot \delta_k \,. \tag{6}$$

As $f(X) = \widehat{f}(X, 0)$, the global forward rounding error is

$$\widehat{f}(X, \delta) - f(X) = \Delta_L - E_L, \tag{7}$$

where E_L is the *linearization error* associated with Δ_L.

3.2 Principles of the CENA Method

Traditional numerical analysis books use the floating point arithmetic model (2) and (3) to bound the absolute elementary errors δ_k in relation (6) — and so bound $|\Delta_L|$, and then neglect E_L to derive a first-order bound of the global rounding error [4], [19]. We propose here to compute $\widehat{\Delta}_L = fl(\Delta_L)$ to derive a *corrected result* $\overline{f(X)}$ defined as

$$\overline{f(X)} = fl\left(\widehat{f}(X,\delta) - \widehat{\Delta}_L\right). \tag{8}$$

We compute $\widehat{\Delta}_L$ evaluating relation (6) in floating point arithmetic. The partial derivatives are computed with algorithmic differentiation. Elementary errors δ_k are computed using the relations presented in next Table 1. These two kinds of computations are well-known but merging them provides a better handling on the rounding errors than the classic first-order bound.

The corrected result $\overline{f(X)}$ suffers from the truncation error E_L and a computing error E_C. This computing error comes from the finite precision evaluation of the partial derivatives and the elementary rounding errors in relation (6), and the final subtraction in the correction (8). So the *residual error* between the corrected result and the exact result is

$$\overline{f(X)} - f(X) = -(E_L + E_C). \tag{9}$$

The smaller the residual error is, the more efficient the correcting method is.

Together with the corrected result $\overline{f(X)}$, the CENA methods provides a bound of the computing error E_C. This bound B_{E_C} bounds the residual error when $E_L = 0$, that is for *linear algorithms*, a particular class of algorithms we define in the next section.

3.3 The Linearization Error E_L and Linear Algorithms

Higher than first-order effects of the elementary rounding errors may contribute to the inaccuracy of a computed result.

EXAMPLE. Let $\widehat{f}_1$ be the evaluation of $f(x,y,z) = x^2 - y^2 - z^2$ with Algorithm 3.3.

$x_1 = x, \quad x_2 = y, \quad x_3 = z$
$x_4 = x_1 + x_2$
$x_5 = x_1 - x_2$
$x_6 = x_4 \times x_5$
$x_7 = x_3 \times x_3$
$x_8 = x_6 - x_7$
$\widehat{f}_1(x,y,z) = x_8$

Relation (5) yields

$$\widehat{f}_1(x,y,z) - f(x,y,z) = \delta_4(x-y) + \delta_5(x+y) + \delta_6 - \delta_7 + \delta_8 + \delta_4\delta_5. \tag{10}$$

Then, computing $f(2^{25},1,2^{25}) = -1$ in IEEE-754 single precision ($\mathbf{u} = 2^{-24}$) gives

$$\widehat{f}_1(2^{25},1,2^{25}) = 0,$$

as

$$\delta_4 = -1, \quad \delta_5 = 1 \quad \text{and} \quad \delta_6 = \delta_7 = \delta_8 = 0.$$

In this case, both the first and second-order terms in relation (10) significantly contribute to the global forward error.

The order and terms that are significant in the development of the global forward error with respect to the elementary rounding errors depend on the algorithm and the data. A general criterion for when to stop computing higher order terms is difficult to define. The *a priori* choice of a maximal order such that higher order terms are not considered may lead to neglect significant terms for the global rounding error. Such terms are here taken into account in the linearization error E_L defined by relation (7).

An accurate bound of E_L is also difficult to compute. Nevertheless, the following linear algorithms satisfies the interesting property $E_L = 0$.

Definition 1 *A* linear algorithm on $\mathbb{F}$ *is an algorithm that only contains the operations* $\{+,-,\times,/,\sqrt{}\}$ *and such that*

- *every multiplication* $fl(\widehat{x}_i \times \widehat{x}_j)$ *satisfies* $\widehat{x}_i = x_i$ *or* $\widehat{x}_j = x_j$, *and*
- *every division* $fl(\widehat{x}_i/\widehat{x}_j)$ *satisfies* $\widehat{x}_j = x_j$, *and*
- *every square root* $fl\left(\sqrt{\widehat{x}_j}\right)$ *satisfies* $\widehat{x}_j = x_j$.

Linear algorithms are such that the identified operands $\widehat{x}_k$ suffer from no previous rounding error. Constants and data of any implemented algorithm are such $\widehat{x}_k$. According to their numerical value, some computed quantities may also satisfy $\widehat{x}_k = x_k$. Even if the final result $\widehat{x}_N$ is not computed with a linear algorithm, some intermediate variables may be returned by linear parts of the algorithm.

Some basic algorithms for scientific computing are linear algorithms. For example, we mention the natural order summation algorithm of n real numbers, the inner product of two vectors and therefore most of the BLAS routines, the polynomial evaluation using Horner's method and the substitution algorithm to solve triangular linear systems.

EXAMPLE (continued). We compute the previous function $f(x,y,z)$ with the following linear algorithm that returns $\widehat{f}_2$.
We have

$$\widehat{f}_2(x,y,z) - f(x,y,z) = \delta_4 - \delta_5 + \delta_6 - \delta_7 + \delta_8.$$

$x_1 = x, \quad x_2 = y, \quad x_3 = z$
$x_4 = x_1 \times x_1$
$x_5 = x_2 \times x_2$
$x_6 = x_4 - x_5$
$x_7 = x_3 \times x_3$
$x_8 = x_6 - x_7$
$\widehat{f}_2(x, y, z) = x_8$

The global forward error of the linear algorithm $\widehat{f}_2$ is a linear function of the elementary rounding errors. This property holds for all linear algorithms.

Theorem 1 *Let $\widehat{x}_N$ be the computed result of $f(X)$ by a linear algorithm on $\mathbb{F}$ for $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$. The global forward rounding error $\widehat{x}_N - f(X)$ is a linear function of the elementary rounding errors $\delta = (\delta_{n+1}, \ldots, \delta_N)$.*

proof 1 *(Main steps) For each elementary operation and $k = n+1, \ldots, N$, we prove by induction that the global rounding error in $\widehat{x}_k$ is a linear function of the elementary rounding errors δ_r $(n+1 \leq r \leq k)$.*

A non-linear propagation of the elementary rounding error δ_r may appear when $\widehat{x}_r$ is one of the two operands of a multiplication, the divisor of a division or the operand of a square root. For the multiplication and $n+1 \leq s \leq r < k$,

$$\begin{aligned} \widehat{x}_k &= \widehat{x}_r \times \widehat{x}_s + \delta_k \\ &= (x_r + \delta_r) \times (x_s + \delta_s) + \delta_k \\ &= x_k + \delta_k + \delta_r x_s + \delta_s x_r + \delta_r \delta_s. \end{aligned}$$

From the linear algorithm definition, we assume for example that $\delta_r = 0$. The non-linear term $\delta_r \delta_s$ vanishes, and the global rounding error $\widehat{x}_k - x_k$ is linear with respect to δ_s and δ_k. The induction is initialized since the data $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$. The division and square root cases are similar and presented in [11].

For a given computation, a linear algorithm provides a linear global forward error. Of course, this global forward error is not necessary smaller than the error produced by another non-linear algorithm.

EXAMPLE (continued). The two previous computations of $\widehat{f}_1$ and $\widehat{f}_2$ suffer from the same forward error for $(x, y, z) = (2^{25}, 1, 2^{25})$.

Now we discuss the correction of linear algorithms.

3.4 The Computing Error E_C and its Bound B_{E_C} when Correcting Linear Algorithms

The correction $fl(\widehat{f} - \widehat{\Delta}_L)$ is performed in floating point arithmetic and so introduces new rounding errors. This computing error E_C is defined by relation (9). The two following corollaries consider this computing error when the results of a linear algorithm are corrected. As linear algorithms satisfy $E_L = 0$, these results derive from relation (9).

Corollary 1 *When $f(X)$ is computed with a linear algorithm, the corrected result $\overline{f(X)}$ only suffers from the computing error E_C,* i.e.*,*

$$\overline{f(X)} = f(X) - E_C.$$

Corollary 2 *Let B_{E_C} be a bound for E_C, that is, $|E_C| \leq B_{E_C}$. The exact result $f(X)$ and the corrected result $\overline{f(X)}$ of a linear algorithm are such that*

$$f(X) \in \mathcal{I}_{E_C} = [\overline{f(X)} - B_{E_C}, \overline{f(X)} + B_{E_C}].$$

The B_{E_C} bound of the rounding errors introduced by the correction process provides a confidence threshold for the corrected result $\overline{f(X)}$. Let us assume until next section that B_{E_C}, and hence the interval $\mathcal{I}_{E_C}$, are computed. From Corollary 2, $f(X)$ belongs to $\mathcal{I}_{E_C}$ (but is unknown). Thus, the corrected value $\overline{f(X)}$ and the confidence threshold B_{E_C} could be used to control the accuracy of the original computed result $\widehat{x}_N$. Comparing $\widehat{x}_N = \widehat{f}(X, \delta)$ with the $\mathcal{I}_{E_C}$ range may provide an estimate of the number of correct digits in the computed result.

EXAMPLE (continued). We apply the correcting process to the previous linear algorithm $\widehat{f}_2$. The corrected result is

$$\overline{f_2(2^{25}, 1, 2^{25})} = -1 = f(2^{25}, 1, 2^{25}).$$

We compute the bound B_{E_C} with Algorithm 3.5 described in the next section. With IEEE-754 single precision arithmetic ($\mathbf{u} = 2^{-24} \approx 5.96 \times 10^{-8}$), we get

$$B_{E_C} = 7.152 \times 10^{-7}.$$

From Corollary 2 and $\overline{f_2(2^{25}, 1, 2^{25})} = -1.0$, the exact result satisfies

$$f(2^{25}, 1, 2^{25}) = -1.0 \pm 7.152 \times 10^{-7}.$$

So we prove that the initial computed value $\widehat{f}_2(2^{25}, 1, 2^{25}) = 0.0$, has no significant digit.

In the following section, we derive an algorithm to compute the bound B_{E_C} for the computing error E_C.

3.5 Computing the Bound B_{E_C}

We write the computing error $E_C = E'_C + e$, where E'_C denotes the rounding errors in the computation of the correcting factor $\widehat{\Delta}_L$ and e is the rounding error in the final subtraction $fl(\widehat{f}(X,\delta) - \widehat{\Delta_L})$.

This last rounding error e satisfies relation (2) of the standard model of floating point arithmetic, *i.e.*,

$$|e| \leq \mathbf{u}\,|\overline{f(X)}|. \tag{11}$$

The other component $E'_C = \widehat{\Delta}_L - \Delta_L$, comes from the floating point evaluation of relation (6), that is, the computation of the partial derivatives $\partial\widehat{f}/\partial\delta_k$, the elementary errors δ_k and the inner product Δ_L $(n+1 \leq k \leq N)$. Langlois and Nativel provide a theoretical approximate bound for E'_C that derives from *a priori* inequalities [10]. However, this bound suffers from the characteristic weakness of *a priori* results: the actual elementary errors are not taken into account as all these errors are bounded by the worst one. The *a priori* bound also increases with the number of intermediate variables which can be very large in applications. So, the computation of this *a priori* bound for E'_C may return large bounds and suffer from overflow.

We propose here to compute a sharper bound for the computing error E_C using Wilkinson's running error analysis [21]. Algorithm 3.5 describes the computation of $\widehat{\Delta}_L$. We derive Algorithm 3.5 that performs the running error analysis of Algorithm 3.5. This running error analysis automatically yields the partial bound $B_{E'_C}$ from which it is immediate to compute the expected bound B_{E_C}.

The computation of the correcting factor $\widehat{\Delta}_L = fl(\sum_{k=n+1}^{N} \frac{\partial\widehat{f}}{\partial\,\delta_k}(X,\delta)\cdot\delta_k)$ is described by Algorithm 3.5 where $u_k = (\partial\widehat{f}/\partial\,\delta_k)(X,\delta)$ and $v_k = \delta_k$ $(k = n+1,\ldots,N)$. The functions A and B are introduced to take into account the rounding errors due to the previous computations of the partial derivatives $\partial\widehat{f}/\partial\,\delta_k$ and the elementary errors δ_k.

$S_n = 0$
for $k = n+1$ **to** N **do**
  $U_k = A(u_k)$
  $V_k = B(v_k)$
  $P_k = U_k \times V_k$
  $S_k = S_{k-1} + P_k$
end for
$\widehat{\Delta}_L = S_N$

Let us assume that global rounding error bounds on $\widehat{U}_k = fl\,(u_k)$, and $\widehat{V}_k = fl\,(v_k)$, are computable and are such that

$$\widehat{U}_k - u_k = \mu_k \quad \text{with} \quad |\mu_k| \leq \mathbf{u}\,\alpha_k \quad (\mu_k, \alpha_k \in \mathbb{R}), \tag{12}$$

and

$$\widehat{V}_k - v_k = \nu_k \quad \text{with} \quad |\nu_k| \leq \mathbf{u}\,\beta_k \quad (\nu_k, \beta_k \in \mathbb{R}). \tag{13}$$

Such bounds α_k and β_k are proposed in Sections 5 and 4.

Applying the floating point arithmetic model (2) and (3) to $\widehat{P}_k$ and $\widehat{S}_k$ computed by Algorithm 3.5 yield

$$\widehat{P}_k = \frac{\widehat{U}_k \widehat{V}_k}{1+\pi_k}, \quad |\pi_k| \leq \mathbf{u},$$

and

$$(1+\sigma_k)\widehat{S}_k = \widehat{S}_{k-1} + \widehat{U}_k \widehat{V}_k - \pi_k \widehat{P}_k, \quad |\sigma_k| \leq \mathbf{u}.$$

Relations (12) and (13) give

$$\widehat{U}_k \widehat{V}_k = u_k v_k + \mu_k \widehat{V}_k + \nu_k \widehat{U}_k - \mu_k \nu_k.$$

With the error $e_k = \widehat{S}_k - S_k$ in relation (3.5) and as $S_k = S_{k-1} + u_k v_k$, we have

$$e_k = e_{k-1} - \pi_k \widehat{P}_k - \sigma_k \widehat{S}_k + \mu_k \widehat{V}_k + \nu_k \widehat{U}_k - \mu_k \nu_k.$$

So, a bound that only depends on computable quantities is

$$|e_k| \leq |e_{k-1}| + \mathbf{u}\,(|\widehat{P}_k| + |\widehat{S}_k| + \alpha_k |\widehat{V}_k| + \beta_k |\widehat{U}_k| + \mathbf{u}\,\alpha_k \beta_k).$$

We write the bound $|e_k| = e_k^{(1)} + e_k^{(2)}$, where $e_k^{(1)}$ and $e_k^{(2)}$ respectively represent the first and second-order terms in $\mathbf{u}$ of $|e_k|$. Since $e_n = 0$, following Algorithm 3.5 is the transformation of Algorithm 3.5 to compute both the correcting factor $\widehat{\Delta}_L$ and a running error bound $B_{E'_C}$ for E'_C.

Knowing the corrected result $\overline{f(X)}$ in relation (11), a bound B_{E_C} for the computing error $E_C = E'_C + e$ introduced by the correcting process is

$$B_{E_C} = \mathbf{u}[(e_N^{(1)} + \overline{f(X)}) + \mathbf{u} e_N^{(2)}]. \tag{14}$$

Hence, the immediate modification of Algorithm 3.5, replacing its last instruction by relation (14), provides the computation of the correcting term $\widehat{\Delta}_L$ and the associated running error bound B_{E_C}.

In practice, a first order running error bound is computed neglecting second-order terms $e_k^{(2)}$ $(k = n+1, \ldots, N)$. Numerical experiments confirm that running error bounds are more accurate than *a priori* bounds. Algorithm 3.5 provides about 100 times sharper bounds than the previous *a priori* result from [10]. This bound improves the previously pointed out assessment of the computed result accuracy that Corollary 2 provides. However, pessimistic bounds and overflow risks are not definitely avoided as a running error bound is also a summation of absolute values which necessarily increases as the computation proceeds.

$S_n = 0, \quad e_n^{(1)} = 0, \quad e_n^{(2)} = 0$
for $k = n+1$ **to** N **do**
  $U_k = A(u_k)$
  $V_k = B(v_k)$
  $P_k = U_k \times V_k$
  $S_k = S_{k-1} + P_k$
  $e_k^{(1)} = e_{k-1}^{(1)} + |P_k| + |S_k| + \alpha_k \times |V_k| + \beta_k \times |U_k|$
  $e_k^{(2)} = e_{k-1}^{(2)} + \alpha_k \times \beta_k$
end for
$\widehat{\Delta}_L = S_N$
$B_{E'_C} = \mathbf{u} \times (e_N^{(1)} + \mathbf{u} \times e_N^{(2)})$

3.6 Correcting Intermediate Results

Correcting the computed final result $\widehat{f}(X)$ is neither always possible nor always efficient. Computing the (absolute) correction $\overline{f(X)} = fl(\widehat{f}(X) - \widehat{\Delta}_L)$ does not guarantee the (relative) accuracy of the corrected $\overline{f(X)}$. This accuracy is limited by the initial accuracy of the computed result $\widehat{f}(X)$ and the precision $\mathbf{u}$. The final subtraction $fl(\widehat{f}(X) - \widehat{\Delta}_L)$ may suffer from severe cancellation and magnifies previous errors in $\widehat{f}(X)$ and $\widehat{\Delta}_L$.

We illustrate this limitation even in the favorable case of a linear algorithm and neglecting the E'_C part of the computing error. In this case the exact correction $\Delta_L = \widehat{f} - f$, generally yields the optimal correcting factor $\widehat{\Delta}_L = fl(\Delta_L)$ in $\mathbb{F}$, *i.e.*, $\widehat{\Delta}_L$ minimizes $|\overline{f(X)} - f(X)|$. A catastrophic cancellation appears when $fl(\widehat{f}(X) - \widehat{\Delta}_L)$ is computed with precision $\mathbf{u}$ and $\mathbf{u}|\widehat{f}(X)| > |f(X)| > 0$. Then $\widehat{\Delta}_L = \widehat{f}(X)$, and $\overline{f(X)} = 0$. When $f(X) \neq 0$, we have $|\widehat{f}(X)| \gg |f(X)| > |\overline{f(X)}| = 0$. Thus the corrected result is more accurate than the initially computed result, however its accuracy may be arbitrarily bad.

EXAMPLE (continued). Let us consider the IEEE-754 single precision evaluation of $g(2^{50}, 2^{25}, 1)$, with $g(x, y, z) = z^2$, but computed as

$$\widehat{g}(x, y, z) = x^2 - y^2 - x^2 + y^2 + z^2.$$

Evaluating $\widehat{g}$ from the left to the right insures $E_L = 0$. We have $g(2^{50}, 2^{25}, 1) = 1$ and $\widehat{g}(2^{50}, 2^{25}, 1) = 2^{50}$. In finite precision $\mathbf{u} = 2^{-24}$, the global error $\Delta_L = 2^{50} - 1$ yields the inaccurate correcting factor $\widehat{\Delta}_L = 2^{50}$. With such a Δ_L, the corrected result would be $\overline{g(2^{50}, 2^{25}, 1)} = 0$, that is not an accurate correction of the exact value 1.

A natural way to avoid this limitation is to reduce the global error in $\widehat{f}(X)$ before it becomes too important to be corrected with the final subtraction $fl(\widehat{f}(X) - \widehat{\Delta}_L)$. This may be achieved by *correcting* some well-chosen *intermediate results* in the computation of $\widehat{f}(X)$.

Intermediate results of a linear algorithm are also computed with a linear algorithm. Hence, it is legitimate to apply the correcting process to such intermediate results.

EXAMPLE (continued). The previous function g satisfies $g(x,y,z) = f_2(x,y,x) + y^2 + z^2$, with function f_2 defined in Section 3.3. As $f_2(2^{50}, 2^{25}, 2^{50}) = -2^{50}$, correcting the intermediate computation of f_2 in function g_2,

$$\overline{g_2(x,y,z)} = \overline{f_2(x,y,x)} + y^2 + z^2,$$

yields the corrected result

$$\overline{g_2(2^{50}, 2^{25}, 1)} = 1 = g(2^{50}, 2^{25}, 1).$$

In this example, no other correction than this intermediate correction is necessary to avoid the previous inaccuracy of the computed result. The final accuracy of this computation relies on the accuracy of this intermediate computation. We name such an intermediate variable a *sensitive intermediate variable*. Of course, to identify sensitive intermediate variables is generally a difficult task.

Correcting sensitive intermediate results extends the applicability and the efficiency of the CENA method. An interesting application of such intermediate correction arises when the stability of the algorithm depends on the accuracy of intermediate results computed with linear algorithms. For example, we improve the stability of the Newton's iteration when computing a polynomial multiple root. The Newton's iteration is not a linear algorithm but involves two polynomial evaluations that may be computed with a linear algorithm, the Horner's scheme. Hence the intermediate evaluations of the polynomial and its derivative are corrected with the CENA method that improves the accuracy of the approximate root and the stability of the iteration. This example of intermediate correction is presented in [3].

In next Sections 4 and 5, we complete the presentation of the CENA method with more technical aspects of the computation of the correcting term $\widehat{\Delta}_L$.

4 Computing the Elementary Rounding Errors

We extend the notation δ_k introduced in relation (4) to include the operation and the operands concerned by the elementary rounding error. For $\circ \in \{+, -, \times, /, \sqrt{}\}$, let $\widehat{x}_i$, $\widehat{x}_j$ and $\widehat{x}_k$ be in $\mathbb{F}$ and such that $\widehat{x}_k = fl(\widehat{x}_i \circ \widehat{x}_j)$. The *absolute elementary rounding error* in the computation of $\widehat{x}_k$ is

$$\delta_k[\circ, x_i, x_j] = fl(\widehat{x}_i \circ \widehat{x}_j) - (\widehat{x}_i \circ \widehat{x}_j) = \delta_k,$$

where $\widehat{x}_j \neq 0$ when $\circ = /$, and $\widehat{x}_i = 0$, $\widehat{x}_j \geq 0$ when $\circ = \sqrt{}$. The elementary rounding error $\delta_k[\circ, x_i, x_j]$ satisfies the two following properties.

Proposition 1 *For* $\circ \in \{+, -, \times\}$*, the elementary rounding error* $\delta_k[\circ, x_i, x_j]$ *belongs to* $\mathbb{F}$*, and is computable using the operations defined on* $\mathbb{F}$*.*

As the elementary rounding error $\delta_k[\circ, x_i, x_j] \notin \mathbb{F}$ when $\circ \in \{/, \sqrt{}\}$, we introduce the approximate elementary error $\widehat{\delta_k}[\circ, x_i, x_j]$ and an approximation bound β_k, to derive a similar property.

Proposition 2 *For* $\circ \in \{/, \sqrt{}\}$*, an approximate elementary error* $\widehat{\delta_k}[\circ, x_i, x_j] \in \mathbb{F}$*, and an approximation bound* $\beta_k \in \mathbb{F}$ *such that*

$$\left|\widehat{\delta_k}[\circ, x_i, x_j] - \delta_k[\circ, x_i, x_j]\right| \leq \mathbf{u}\,\beta_k,$$

are computable using the operations defined on $\mathbb{F}$*.*

The operations $\circ \in \{+, -, \times\}$ satisfy Property 2 with $\beta_k = 0$. So, we can compute the elementary rounding errors δ_k, and the bound β_k for the rounding errors in the computation of δ_k. We need this bound β_k in relation (13) to implement the running error analysis of Section 3.5.

Remark 1 (About the proof of Properties 1 and 2). Table 1 summarizes the computing relations for δ_k, the bounds β_k and the overhead introduced by the computation of δ_k, assuming a binary arithmetic with the "round to the nearest" rounding mode. No hat notation is used in this table as operands and operations are obviously computed quantities. Elementary rounding errors were largely studied in the 1970s. Except for the square root operator, the relations for computing δ_k are quoted from references [5], [9], [15]. The integer p_1 is such that $p_1 = p/2$ for an even number of digits p of the floating point mantissa, and $p_1 = (p-1)/2$ otherwise.

The numerator of $\delta_k[/, x_i, x_j]$ belongs to $\mathbb{F}$ [16]. Hence, the error bound β_k for this operation only takes into account the rounding error of the last division by x_j in the computation of $\delta_k[/, x_i, x_j]$. For the square root, we neglect the second-order terms in $\mathbf{u}$ and derive the bound β_k for $\delta_k[\sqrt{}, x_i, x_j]$ after some straightforward manipulations (see [10] for details).

In practice, the computation of the elementary errors partly contributes to the time overhead introduced by the CENA method. For each elementary operation $\circ$, let $t_{\delta_k[\circ]}$ be the time to compute the elementary rounding error $\delta_k[\circ]$, and $t_\circ$ be the time to compute the operation $\circ$. As the latter depends on the computer, we assume that the floating point operations satisfy the following (realistic) relative performances : $t_\pm = t_\times = t, t_/ = 10t$ and $t_\surd = 20t$ [18]. Hence, the ratios of time $t_{\delta_k[\circ]}/t_\circ$ presented in Table 1 measure the time overheads due to the computation of each elementary rounding error.

Table 1: Elementary rounding errors relations, bounds and overheads.

$\circ$	$\delta_k[\circ, y, z]$ for $x_k = y \circ z$	β_k	$\frac{t_{\delta_k[\circ]}}{t_\circ}$
$\pm$	$(x_k - y) \mp z$ where $\lvert y\rvert \geq \lvert z\rvert$	0	2
$\times$	$x_k - (y^U \times z^U) - [(y^U \times z^L) + (y^L \times z^U)] - (y^L \times z^L)$, with $x^U = [x - (x \times M)] + (x \times M)$, $x^L = x - x^U$, $(x = y, z)$, and $M = 2^{p_1} + 1$.	0	16
$/$	$\{(x_k \times z) - y - \delta[\times, x_k, z]\} / z$	$\delta_k[/, y, z]$	2.9
$\sqrt{}$	$\{(x_k \times x_k) - y + \delta[\times, x_k, x_k]\} / (2 \times x_k)$	$\frac{5}{2}\,\delta_k[\sqrt{}, y]$	3

5 Computing the Partial Derivatives

The other part of the correcting factor computation relies on algorithmic differentiation. We limit this section to the main aspects of algorithmic differentiation in reverse mode (Section 5.1) and derive an algorithm to compute the associated running error bound (Section 5.2). Practical aspects are briefly described in last Section 5.3. Complementary material concerning algorithmic differentiation is detailed in the recent reference [6].

5.1 Algorithmic Differentiation in Reverse Mode

The partial derivatives $\partial \widehat{f} / \partial \delta_k$ in the correcting factor $\widehat{\Delta_L}$ are computed with algorithmic differentiation. The algorithmic differentiation of $\widehat{f}$ is a direct application of the differentiation chain rule

$$\frac{\partial \widehat{f}}{\partial \widehat{x}_k} = \sum_{C \in \Gamma_k} \prod_{\text{arc}(\widehat{x}_i, \widehat{x}_j) \in C} \frac{\partial \widehat{x}_j}{\partial \widehat{x}_i} \quad (1 \leq k \leq N).$$

This computation relies on a graph representation of algorithm $\widehat{f}$. The graph vertices are the initial and intermediate variables $\widehat{x}_k$ of $\widehat{f}$ $(1 \leq k \leq N)$. The graph arc between $\widehat{x}_i$ and $\widehat{x}_j$, $\text{arc}(\widehat{x}_i, \widehat{x}_j)$, is weighted by $\partial \widehat{x}_j / \partial \widehat{x}_i$ for $i < j$. This elementary partial derivative $\partial \widehat{x}_j / \partial \widehat{x}_i$

is computed given that $\widehat{x}_j = fl(\widehat{x}_i \circ \widehat{x}_l)$ $(i,l<j)$. For a given intermediate variable $\widehat{x}_k$, Γ_k is the set of paths C from $\widehat{x}_k$ to the result $\widehat{x}_N = \widehat{f}(X)$ $(n+1 \leq k \leq N)$. Let us introduce a Vertices_From function that returns the (at most two) vertices connected by an arc to a given vertex. Vertices_From$(\widehat{x}_k)$ returns $\widehat{x}_i$ and $\widehat{x}_j$ when $\widehat{x}_k = fl(\widehat{x}_i \circ \widehat{x}_j)$.

As $\widehat{x}_k = x_k + \delta_k$,

$$\frac{\partial \widehat{x}_k}{\partial \delta_k} = 1 \quad (n+1 \leq k \leq N),$$

and as $x_k \in \mathbb{F}$ for $1 \leq k \leq n$,

$$\frac{\partial \widehat{x}_k}{\partial \delta_k} = 0 \quad (1 \leq k \leq n).$$

So the expected partial derivatives with respect to the elementary rounding errors are computable using

$$\frac{\partial \widehat{f}}{\partial \delta_k} = \sum_{C \in \Gamma_k} \prod_{\text{arc}(\widehat{x}_i, \widehat{x}_j) \in C} \frac{\partial \widehat{x}_j}{\partial \widehat{x}_i} \quad (n+1 \leq k \leq N). \tag{15}$$

The *reverse mode* of algorithmic differentiation enables us to compute relation (15). The reverse mode includes two stages. The graph construction is first performed, for example as algorithm $\widehat{f}$ is processed, from the bottom to the top, that is, from $\widehat{x}_1, \ldots, \widehat{x}_n$ to $\widehat{x}_N$. Then, the summation in relation (15) is taken from the top to the bottom of the graph, that is, from $\widehat{x}_N$ to $\widehat{x}_k$ with $n+1 \leq k < N$.

Assuming the graph has been constructed, the computation of the derivatives using algorithmic differentiation in reverse mode is described by Algorithm 5.1.

Require: $\partial x_k / \partial x_i$, $i \in$ Vertices_From(x_k) *for* $k = n+1, N$.
  for $k = 1$ **to** $N-1$ **do**
    $Dx_k = 0$
  end for
  $Dx_N = 1$
  for $k = N$ **down to** $n+1$ **do**
    for $i \in$ Vertices_From(x_k) **do**
      $Dx_i = Dx_i + Dx_k \times (\partial x_k / \partial x_i)$
    end for
  end for
  $\partial \widehat{f} / \partial \delta_k = Dx_k$

5.2 Bounding the Rounding Errors in Algorithmic Differentiation

Using running error analysis, we bound the rounding errors introduced by the computation of the derivatives $\partial\widehat{f}/\partial\delta_k$, and hence derive the bound α_k defined by relation (12) for $n+1 \leq k \leq N$.

The bound analysis is similar to the analysis conducted for E_C in Section 3.5. The rounding errors in the summation and the product of Algorithm 5.1 are taken into account using the relations (2) and (3) of the standard model of floating point arithmetic. The computation of $\partial\widehat{x}_k/\partial\widehat{x}_i$ may suffer from rounding error when $\widehat{x}_k = \widehat{x}_j/\widehat{x}_i$ or $\widehat{x}_k = \sqrt{\widehat{x}_i}$, as $(-1/\widehat{x}_i^2)$ and $1/(2\widehat{x}_k)$ are computed. These rounding errors are taken into account within the global error of the computation of (15), introducing a scalar C_{ki} and a bound γ_{ki} such that

$$C_{ki} = \frac{\partial\widehat{x}_k}{\partial\widehat{x}_i},$$

and

$$\left|\widehat{C}_{ki} - \frac{\partial\widehat{x}_k}{\partial\widehat{x}_i}\right| \leq \mathbf{u}\,\gamma_{ki}.$$

In binary arithmetic with "round to the nearest" rounding mode, we have

$$\gamma_{ki} = \begin{cases} 2.02\,|\widehat{C}_{ki}| & \text{when } \widehat{x}_k = \widehat{x}_j/\widehat{x}_i, \\ |\widehat{C}_{ki}| & \text{when } \widehat{x}_k = \sqrt{\widehat{x}_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

The first result derives from a classic *a priori* rounding error analysis of the computation of $(-1/\widehat{x}_i^2)$ (for example, applying Lemma 3.4 of [7] as $2\mathbf{u} < 1$). The second result is similar for $1/(2\widehat{x}_k)$ and uses the fact that the basis $\mathsf{b} = 2$. The same kind of bounds can be derived according to other specific floating point arithmetic properties. Running error analysis yields here equivalent bounds with more operations to compute.

To simplify the algorithm that yields the running error bound associated with the computation of the derivatives, the second-order terms in $\mathbf{u}$ are hereafter neglected. Taking them into account leads to an algorithm similar to Algorithm 3.5. Thus, Algorithm 5.2 computes both the derivatives $\partial\widehat{f}/\partial\delta_k$ and the running error bound α_k associated to the rounding errors introduced by this computation.

5.3 Discussion

The computation of the derivatives is the main contribution to the time overhead introduced by the CENA method we began to discuss at the end of Section 4.

The main theoretical results about time and space complexity of algorithmic differentiation are from [2]. For the reverse mode, the relative time overhead is bounded by m times a constant when $\widehat{f}(X) \in \mathbb{R}^m$. For straightforward implementation, Iri derives the constant

Require: $\partial x_k/\partial x_i$, $i \in$ Vertices_From(x_k) *for* $k = n+1, N$.
  for $k = 1$ **to** $N-1$ **do**
    $Dx_k = 0$
    $e_k = 0$
  end for
  $Dx_N = 1$
  for $k = 1$ **to** N **do**
    $\alpha_k = 0$
  end for
  for $k = N$ **down to** $n+1$ **do**
    for $i \in$ Vertices_From(x_k) **do**
      $C_{ki} = \partial x_k/\partial x_i$
      $Dx_i = Dx_i + Dx_k \times C_{ki}$
      $e_i = e_i + |Dx_k \times C_{ki}| + |Dx_i| + e_k \times |C_{ki}| + \gamma_{ik} \times |Dx_k|$
    end for
  end for
  $\partial \widehat{f}/\partial \delta_k = Dx_k$
  $\alpha_k = \mathbf{u} \times e_k$

4 in [8]. The relative space overhead in reverse mode is bounded by a constant times the sum of $\widehat{f}$ computation space and time units.

The practical overheads of existing algorithmic differentiation tools are not necessarily of the same order. The graph construction mainly consists of memory management that is very sensitive to the size of $\widehat{f}$, implementation choices and machine memory characteristics. Overloading operators provide a direct implementation of algorithmic differentiation and elementary error computations. We use such facilities in Fortran 90 and Ada. In our experience, a constant of an order of 20 is a more reasonable practical bound for the relative time overhead of the algorithmic differentiation part of the correcting process. It is interesting to note than Linnainmaa's pioneering work reports that the linearization error increased the execution time by a factor of a few tens [13].

6 The Algorithm of the CENA Method

In this section, we summarize the previous results and give a description of the three main stages of the CENA method.

1. Before computing $\widehat{f}$: Choose the data $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and the result variable x_N (in the scalar case) that defines $\widehat{x}_N = \widehat{f}(X)$ for $\widehat{f}$ satisfying Definition 1.

2. During the computation of $\widehat{f}$: For each elementary computation $\widehat{x}_k = fl\,(\widehat{x}_i \circ \widehat{x}_j)$,

   - Compute $\widehat{x}_k$;

- Compute the elementary error $\delta_k[\circ, x_i, x_j]$ according to the relations shown in Table 1;
- Compute the associated rounding error bound β_k according to the relations shown in Table 1;
- Compute the elementary partial derivatives $\partial\widehat{x}_k/\partial\widehat{x}_i$ and $\partial\widehat{x}_k/\partial\widehat{x}_j$, given that $\widehat{x}_k = fl\,(\widehat{x}_i \circ \widehat{x}_j)$;
- Compute the associated rounding error bound γ_{ki} and γ_{kj} according to relation (16);
- Update the computational graph for $\widehat{f}$ such that function Vertices_From($\widehat{x}_k$) returns $\widehat{x}_i$ and $\widehat{x}_j$;
- Update the computational graph for $\widehat{f}$ storing the values of $\delta_k[\circ, x_i, x_j]$, β_k, $\partial\widehat{x}_k/\partial\widehat{x}_i$, $\partial\widehat{x}_k/\partial\widehat{x}_j$, γ_{ki} and γ_{kj}.

3. After $\widehat{x}_N = \widehat{f}(X)$ is computed :
   - Compute the correcting factor $\widehat{\Delta}_L$ according to relation (6);
   - Compute the corrected result $\overline{f(X)}$ according to relation (8);
   - Compute the error bound α_k for the computation of the derivatives according to Algorithm 5.2;
   - Compute the confidence threshold B_{E_C} according to relation (14) and Algorithm 3.5;
   - (optional) Assess the accuracy of the original computed result $\widehat{x}_N$ according to Corollary 2.

7 Conclusion

The CENA method is a forward and deterministic method, useful to highlight the effect of rounding errors in numerical algorithms. Finite precision computation provides results that suffer from every elementary rounding error generated by the floating point arithmetic operations. For linear algorithms, the global forward error is a linear function of these elementary rounding errors. The CENA method computes the global forward error and provides a corrected result, together with a bound of the residual error of the corrected value.

We apply the CENA method to correct final results or sensitive intermediate variables computed with linear algorithms. Correcting sensitive intermediate variables extends the applicability of the method and enhances the stability of some numerical algorithms [3].

In most cases, the correcting process improves the accuracy of the computed result. The general efficiency of the CENA method is close to double the precision of the computation. The correction relies on floating point computations that also suffer from rounding errors.

Hence cases for which the correcting process does not improve the initial accuracy may be designed. Conversely, we have presented examples where the corrected result is significantly more accurate than the result computed with extended precision.

The confidence threshold associated to the corrected result sometimes enables us to check the initial accuracy of the computed result. The reliability of the method depends on the linearity property of the algorithm. Applying the CENA method to a non linear algorithm may improve the accuracy of the result but forbid to analyze the confidence threshold B_{E_C} associated to the corrected result.

Numerical applications frequently use elementary functions such as trigonometric functions, exponential functions, ... Further development of the CENA method should efficiently manage the computation of accurate approximations of the rounding error of elementary functions. In practice, the performance of the implementation will be improved taking into account the newest improvements in algorithmic differentiation software. Future work will also compare corrected algorithms and algorithms computed with extended precision library such as MPFUN [1].

Finally, it is interesting to point out recent results similar in concept to the linear correction principle of the CENA method but applied to another kind of error, the discretization errors in approximating partial differential equations [17]. Pierce and Giles are concerned with the effect of these discretization errors to outputs that are integral functionals expected with a very accurate prediction, *e.g.*, lift and drag in aeronautical applications of computational fluid dynamics. They compute a correcting factor that is an inner product of local residual errors with an approximate solution of the adjoint equations, and an *a posteriori* error bound of the remaining error term. They present numerical experiments showing that the corrected solution has twice the order of accuracy of the initially computed approximate solution.

Acknowledgments

The author is grateful to Thierry Braconnier, Fabrice Nativel and Jean-Chris-tophe Rioual (Université de La Réunion) for their collaboration to parts of this work. He also thanks Nicholas J. Higham for his several useful suggestions, Sheung H. Cheng and Philip I. Davies (University of Manchester) for their numerous language correction and the anonymous referees for remarks leading to a significant improvement of the manuscript.

Appendix

We describe the two triangular systems $U_\alpha x = b$ and $L_\alpha x = b_\alpha$ introduced in Section 2. These systems are designed to generate a large forward error when the solution is computed by the substitution algorithm. These systems satisfy the following characteristics.

- Relations depending on the order n and the integer parameter α define the matrices, the right-hand side members and the corresponding solutions.
- The solution x of the triangular linear system for dimension n also yields the solutions of corresponding systems of dimension $m < n$, extracting m *ad hoc* entries from x.
- Matrices are very ill conditioned even for small dimension n.
- Matrices and right-hand side members entries are exact IEEE-754 single precision floating point numbers.

7.1 The Triangular Linear System $U_\alpha x = b$

All entries of this upper triangular linear system $U_\alpha x = b$ are integer values defined as follows.

Matrix U_α:

$$\left|\begin{array}{lll} U_\alpha(i,i) = 1, & i = 1,\ldots,n; & \\ U_\alpha(i,n) = 1, & i = n-1,[-2],1; & \\ U_\alpha(i,j) = (-1)^{i+j+1}.2^\alpha, & i = n-2,[-2],1; & j = i+1,\ldots,n; \\ U_\alpha(i,j) = (-1)^{i+j}, & i = n-3,[-2],1; & j = i+1,\ldots,n-1; \\ U_\alpha(i,j) = 0, & i > j. & \end{array}\right.$$

Right-hand side b:

$$\left|\begin{array}{ll} b(i) = 1, & i = n,[-2],1; \\ b(i) = 2, & i = n-1,[-2],1. \end{array}\right.$$

Solution x:

$$\left|\; x(i) = 1, \quad i = 1,\ldots,n.\right.$$

The componentwise condition number for U_α of order n verifies

$$\mathrm{cond}(U_\alpha) = \| \, |U_\alpha^{-1}||U_\alpha| \, \|_\infty = O(n2^\alpha).$$

We choose α such that the entries $\pm 2^\alpha$ are exact in binary floating point arithmetic, *i.e.*, 2^α suffers from no overflow nor underflow. Catastrophic cancellations in the back substitution appear respectively for IEEE-754 single and double precision when $\alpha > 24$ and $\alpha > 53$. In these cases, the computed solution $\widehat{x}$ satisfies $\widehat{x}(i) = 0$ for $i = 1,\ldots,n-2$.

Example of Section 2.1 is the system $U_\alpha x = b$, for $\alpha = 55$ and $n = 6$, *i.e.*,

$$\begin{bmatrix} 1 & -1 & 1 & -1 & 1 & 1 \\ 0 & 1 & 2^{55} & -2^{55} & 2^{55} & -2^{55} \\ 0 & 0 & 1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2^{55} & -2^{55} \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \\ 1 \\ 2 \\ 1 \end{bmatrix}.$$

Matlab returns the corresponding componentwise condition number $\mathrm{cond}(U_\alpha) \approx 3.60 \times 10^{17} \gtrsim (\mathbf{u}_{\mathrm{double}})^{-1}$, where $\mathbf{u}_{\mathrm{double}} \approx 1.11 \times 10^{-16}$ is the IEEE-754 double precision.

7.2 The Triangular Linear System $L_\alpha x = b_\alpha$

The lower triangular linear system $L_\alpha x = b_\alpha$ satisfies the following relations.

Matrix L_α:

$$\left|\begin{array}{lll} L_\alpha(1,1) = 100; & & \\ L_\alpha(i,i) = 1, & i = 2,\ldots,n; & \\ L_\alpha(i,j) = (-1)^{i+j}\alpha, & i = 2,\ldots,n; & j = 1,\ldots,i-1; \\ L_\alpha(i,j) = 0, & i < j. & \end{array}\right.$$

Right-hand side b_α:

$$\left|\begin{array}{ll} b_\alpha(1) = 1; & \\ b_\alpha = -\frac{\alpha+1}{100}(-2)^{i-2}, & i = 2,\ldots,n. \end{array}\right.$$

Solution x:

$$\left|\begin{array}{ll} x(1) = 0.01; & \\ x(i) = -\frac{1}{100}(-2)^{i-2}, & i = 2,\ldots,n. \end{array}\right.$$

We choose α such that the entries of L_α and b_α are exact floating point numbers. In binary arithmetic, the values 24, 224, 1024, ... are acceptable choices for α with a simple decimal notation.

On the contrary, the first entries in the solution are not finite binary floating point numbers. So, any binary computed solution suffers from this unavoidable rounding error in the first entries. As the componentwise condition number of matrices L_α grows at exponential rate, these initial rounding errors propagate exponentially.

Examples in Section 2.2 rely on the following system $L_\alpha x = b$, with $\alpha = 224$ and $n = 10$,

$$\begin{bmatrix} 100 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ -224 & 1 & \ddots & & & & \vdots \\ 224 & -224 & 1 & \ddots & & & \vdots \\ -224 & 224 & \ddots & 1 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 224 & \ddots & \ddots & \ddots & \ddots & 1 & 0 \\ -224 & 224 & & & 224 & -224 & 1 \end{bmatrix} \begin{bmatrix} 0.01 \\ -0.01 \\ 0.02 \\ \vdots \\ \vdots \\ \vdots \\ 1.28 \\ -2.56 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.25 \\ 4.5 \\ \vdots \\ \vdots \\ \vdots \\ 288 \\ -576 \end{bmatrix}.$$

With Matlab, we verify that $\mathrm{cond}(L_\alpha) \approx 2.76 \times 10^{21} \gg (\mathbf{u}_{\mathrm{double}})^{-1}$.

References

[1] D. H. Bailey, *Algorithm 719: Multiprecision translation and execution of FORTRAN programs*, ACM Trans. Math. Software, 19 (1993), pp. 288–319.

[2] W. Baur and V. Strassen, *The complexity of partial derivatives*, Theoretical Computer Science, 22 (1983), pp. 317–330.

[3] T. Braconnier and P. Langlois, *From rounding error estimation to automatic correction with automatic differentiation*, in Proceedings of the Third International Conference on Automatic Differentiation: from Simulation to Optimization, Nice, June 2000, Springer-Verlag. (To appear, available at URL = `http://www.ens-lyon.fr/~plangloi`).

[4] G. Dahlquist and Å. Björck, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1974. Translated by Ned Anderson.

[5] T. J. Dekker, *A floating-point technique for extending the available precision*, Numer. Math., 18 (1971), pp. 224–242.

[6] A. Griewank, *Evaluating derivatives. Principles and techniques of algorithmic differentiation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[7] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.

[8] M. Iri, *History of automatic differentiation and rounding error estimation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1991, pp. 3–16.

[9] D. E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Addison-Wesley, Reading, MA, USA, third ed., 1998.

[10] P. Langlois and F. Nativel, *Reduction and bounding of the rounding error in floating point arithmetic*, C.R. Acad. Sci. Paris, Série 1, 327 (1998), pp. 781–786.

[11] ——, *When automatic linear correction of rounding errors is exact*, C.R. Acad. Sci. Paris, Série 1, 328 (1999), pp. 543–548. Erratum in 328:829, 1999.

[12] S. Linnainmaa, *Towards accurate statistical estimation of rounding errors in floating-point computations*, BIT, 15 (1975), pp. 165–173.

[13] ——, *Error linearization as an effective tool for experimental analysis of the numerical stability of algorithms*, BIT, 23 (1983), pp. 346–359.

[14] W. MILLER, *Software for roundoff analysis*, ACM Trans. Math. Software, 1 (1975), pp. 108–128.

[15] M. PICHAT, *Correction d'une somme en arithmétique à virgule flottante*, Numer. Math., 19 (1972), pp. 400–406.

[16] ——, *Contribution à l'étude des erreurs d'arrondi en arithmétique à virgule flottante*, Doctorat d'Etat Thesis, Université de Grenoble 1, Grenoble, France, 1976. (In French).

[17] N. A. PIERCE AND M. B. GILES, *Adjoint recovery of superconvergent functionals from PDE approximations*, SIAM Review, 42 (2000), pp. 247–264.

[18] P. SODERQUIST AND M. LEESER, *Division and square root: Choosing the right implementation*, IEEE Micro, 17 (1997), pp. 56–66.

[19] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, second ed., 1993.

[20] M. TIENARI, *A statistical model of roundoff error for varying length floating-point arithmetic*, BIT, 10 (1970), pp. 355–365.

[21] J. H. WILKINSON, *Error analysis revisited*, IMA Bulletin, 22 (1986), pp. 192–200.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

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