Technical Report

Revista de Informática Teórica e Aplicada

# Computational Arithmetic: An Updated View

Tiarajú A. Diverio
Dalcidio M. Claudio
Instituto de Informática e CPGCC da UFRGS
P.O.BOX: 15.064 – 91.501-970 – Porto Alegre – Brazil
Phone: +55 (051) 316.68.46 – Fax: +55 (051) 319.15.76
E-mails: diverio@inf.ufgrs.br — dalcidio@music.pucrs.br

**Abstract**

This paper begins with a review of the floating-point system, where floating-point numbers are characterized. It reveals the need of a standard in order to solve the problem of production of different results in different computers, as well as the problems of library's portability and software package. It also characterizes the IEEE-754 standard, which specifies the binary floating-point system for computers since 1980. This standard also specifies: representation of numbers, arithmetic operations, conversion between formats and treatment of exception with underflow and overflow. Special types of the NaN (not a number), + and − infinite, non normalized numbers and implicit bit are also characterized.

## Key Words

Computational Arithmetic, Floating Point Arithmetic, High Performance Arithmetic.

# 1. Introduction

Some decades ago, everything considered as scientific computation was done by means of a slide rule or tables of logarithms of numbers and standard functions. The use of logarithms would avoid the necessity of using exponents, making easier the use of slide rules, but the precision was limited. Additions and subtractions were done so that terms of relatively small magnitude were excluded and subtraction of almost equal terms could let the results without representation. Unfortunately, they were ignored. When done by hand, the number of possible computations was so small that the effects of neglecting small terms and cancellation could actually be observed and appropriate measures could be taken (Kulisch, [KUL93]).

Computer arithmetic characterizes how real numbers are represented and how operations in a digital machine are performed. The representation of machine numbers are described by the representation system (fixed-point, floating-point or integer). The way the operations are carried out is described by the arithmetic itself.

The representation of real numbers in computers is an important question since the beginning of computer's development. Real numbers were initially represented in the fixed-point numerical system. The passage to the representation in floating-point started in the fifties and has characterized a significant evolution in Science of Computation and Scientific Computation, especially by the improvement of quality (accuracy) in the results of operations performed in floating-point.

This evolution continued, not in a well standardized form, by means of the representation of floating-point numbers with more and more digits in the mantissa. The mantissa length depended on the machine and, yet, it could vary, resulting in formats known as single precision, double precision and extended precision, respectively.

When one wants to talk about a floating-point system, or when we want to characterize it, it is necessary to specify the numerical base of the system, the mantissa's number of digits, the base's exponent range interval and how the representation of these numbers is done, how underflow and overflow are treated, how arithmetic operations are performed and which roundings are available and used in those operations, for they will all influence the analysis and the amount of errors that computations done in this system will have.

Undoubtedly, the representation of a floating-point number has many advantages, but it has also introduced some disadvantages, such as the error control problem in numerical computations, which many times turns out totally wrong results, but that seem to be correct. That is, the procedure is correct, but the result loses the meaning due to the inaccuracy of the numerical representation and of roundings applied in the evaluations of the floating-point operations and arithmetic expressions.

Another question resulting from this variety of machines and of its floating-point systems available in the market is that the computations done in each machine would provide different results. In some machines, the result of an expression or method was significant, whereas it was not in other machines.

In order to standardize and give new advantages, the standards Subcommittee of the Institute of Electrical and Electronics Engineers (IEEE) has developed standards, such as the standard for binary arithmetic, called IEEE-754 (recognized in 1985) or IEEE standard 854, which defines the norms for decimal floating-point arithmetic.

The IEEE 754 and 854 [IEEE 85] floating-point standards provide four basic arithmetical operations, $+$, $-$, $*$, / with maximum accuracy, including control of rounding towards zero, rounding to the nearest machine number, towards minus infinity and towards plus infinity. Maximum accuracy means least-bit accuracy. That is, there is no other machine number between the value and the rounded result. Nowadays, many of the floating-point exact processors available in the market are equipped with an arithmetic according to these standards, especially in microcomputers.

Another arithmetic has been proposed by Kulisch. In this arithmetic [KUL81, KUL83], new basic operations were added, called optimal scalar product of two vectors with rounding control evaluated with maximum accuracy. These basic operations are fundamental to arithmetical operations such as multiply and sum or multiply and accumulate [IGR89], and operations with vectors and matrices, all computed with maximum accuracy. The optimal scalar product is also a necessary operation for the verification and validation of numerical problem solutions, as commented later in this article.

## 2. Computational Arithmetic Definition

In the computer's memory, each number is stored in a position, which consists in a signal ("+" or "−") plus a fixed number of digits. An important question in a computer architecture is how the digits that represent the numbers which compose the numerical systems will be used.

There are two principal numerical systems that are used in digital computers: the fixed-point and the floating-point systems. Each one has its own concept of computational arithmetic. The fixed-point system is still used in some commercial and financial applications. The results computed in those applications do not exceed the interval of representation of the integers, and consequently they are free of errors. For this reason, the computer's image before people in general is of a "perfect" computational tool.

In the representation model of fixed-point numbers two parts are considered, an integer and a fraction one. The characterization consists of the numerical base used ($b$), the number of digits ($n$) and the number of digits of the fraction part ($f$). It is represented by the 3-tuples $P(b, n, f)$. In the fifties, the floating-point representation in computers was introduced. Nowadays, most of computers use the floating-point number system, denoted by $F(b, n, e_1, e_2)$ where "$b$"is the base number , "$n$"is the precision and "$e_1$" and "$e_2$" are the minimum and the maximum exponent of the interval. Any non zero real number $x$, is represented in $F$ in the form:

$$x := \pm(\frac{d_1}{b^1} + \frac{d_2}{b^2} + \ldots + \frac{d_n}{b^n})b^e$$

But with this introduction a problem with error control arose. That is, the results obtained do not correspond to the reality. In some cases there were digits available or idle in the fraction part, whereas in the integer part there would occur overflow.

In other cases, the digits available were in the integer part, whereas the result would lose its meaning through lack of digits in the fraction part. The need of controlling these errors arose in order to make possible, at the end of the computation, a result with higher accuracy.

Problems involving various arithmetic calculations, where several digits are necessary, happens many times in natural and technological sciences.

The floating-point system allows floating-point arithmetic operations to be used with significant approximation in order to calculate the solution of these problems.

Computers today perform basic floating-point operations with high or even maximum accuracy. However, the results of computations composed of several arithmetic operations can be completely wrong. That is, the result of each operation differs at most by 1/2 unit in the last place, according to the choice of the rounding function. However, after two or more floating-point operations, the result can be totally wrong.

An example of this is the addition: $10^{50}+812-10^{50}+10^{35}+511-10^{35} = 1323$ . By adding these numbers from the left to the right, most computers give ZERO as result. This error comes up because the floating-point format of these computers cannot operate with the large interval of digits required for this computation.

Another example of this is the scalar product. Let $x$ and $y$ be two vectors with six components: $x = (10^{20}, 1223, 10^{18}, 10^{15}, 3, -10^{12})^t$ and $y = (10^{20}, 2, -10^{22}, 10^{13}, 2111, 10^{16})^t$. Let the scalar product be denoted by $x.y$. Here, the corresponding components are multiplied, and all the products are summed. In exact arithmetic, the result would be 8779, for the scalar product would have the following summands: $10^{40}+2446-10^{40}+10^{28}+6333-10^{28}$. However, every computer (including those with IEEE arithmetic) gives the value zero for this scalar product. The reason for this is that the summands have such different orders of magnitude that they cannot be processed correctly in floating-point. This catastrophic error occurs even though the data (the components of the vectors) consume less than 4% of the exponent range of computers of small and medium size.

As a third example, it will be considered a floating-point system with base 10 and a mantissa of five digits. The difference of the two numbers, $x = 0.10005 \cdot 10^5$ and $y = 0.99973 \cdot 10^4$, must be computed. Now, both operands are from the same magnitude order. The computer now gives the completely correct result: $x \cdot y = 0.7700 \cdot 10^1$. Now, suppose that the two numbers $x$ and $y$ are each the result of two previous multiplications. Since we are dealing with five-digit arithmetic, these products of course have ten digits. Suppose that the unrounded products are $x_1 \cdot x_2 = 0.1000548241\ .10^5$, $y_1 \cdot y_2 = 0.9997342213\ .10^4$. Subtracting these two numbers, after rounding to five places, one obtains the result $(x_1 \cdot x_2 - y_1 \cdot y_2) = 0.81402\ .10^1$, which differs in every digit from the result computed above.

In the second case, the value of the expression $x_1 \cdot x_2 - y_1 \cdot y_2$ was computed to the closest five-digit floating-point number. On the other hand, the floating-point arithmetic with rounding after each operation gave a completely wrong result.

Computers, also, have become faster. In the middle of the fifties, they could perform about 100 floating-point operations per second. Today, fast computers can execute some billion floating-point operations per second, and this in situations where the whole process can takes hours. However, the formats of the data use has changed. The process is done with about 17 decimal digits. Sometimes, a format twice as long is used in cases of very important applications.

In the classical analysis of the error in numerical algorithms, the error in each floating-point operation is estimated. Without doubt, it is not possible to proceed in this way for an algorithm that performs $10^{14}$ operations per hour. Thus, an error analysis is no longer performed as a rule. Indeed, the possibility that the result may be wrong is not always taken into consideration. Once in a while, the user tries to justify the computed result by means of heuristic methods, or make it plausible. However, there are also users who are not aware of the fact that the computed results could be completely wrong, perhaps due to confusion with the completely different properties of computation with integers. From the mathematical point of view, the problem of correctness of computed results is of central importance because of the high computational speed attained today. The determination of the correctness of computed results is essential in many applications that are now classical, such as simulation or mathematical modeling. We have apparently used the floating-point arithmetic with those problems and it has been said it is a necessary evil.

## 2.1 Representation

A real number $x \in \mathbb{R}$ is called a normalized floating-point number if it is in the form $x = mb^e$, where $m = \pm 0.d_1 d_2 d_3 ... d_n$, and one fixed $n \in \mathbb{N}$ and where it holds that:

a) $1 \leqslant d_1 \leqslant b - 1$,

b) $0 \leqslant d_i \leqslant b - 1$, $i = 2(1)n$

c) $e_1 \leqslant e \leqslant e_2$, where $e_1 \leqslant 0$, $e_2 > 1$ and $e_1, e_2 \in \mathbb{Z}$

d) zero: $0 = 0.000...0.b^{e_1}$

The union of all floating-point numbers defined in this way is called a floating-point system. Let a floating-point system $F = F(b, n, e_1, e_2)$ be given. The number $0.1\ b^{e_1}$ represents the smalless positive non zero number of floating-points and $B = 0.[b-1][b-1][b-1]\cdots[b-1]\ .b^{e_2}$ represents the bigger floating-point number in this system.

A floating-point system $F$ consists of a finite number of elements. They are equally spaced between the successive powers of $b$ and its negative image. This space changes at each power of $b$. Figure shows a simple floating-point system $F = (2, 3, -1, 2)$ with 33 elements. The successive powers of 2 are $2^{1/4}, 2^{1/2}, 2^1, 2^2$. The floating-point system $F$ has lower and upper element. Each number in $F$ must represent a certain interval of real numbers. For example, in the figure 2.1, the floating-point number 3 may represent the black interval indicated. A floating-point system looks like a screen over the real numbers. The floating-point filter expression is also used by some authors.

| $e$ | $2^e$ | $m_1 = 0.100$ | $m_2 = 0.101$ | $m_3 = 0.110$ | $m_4 = 0.111$ |
|---|---|---|---|---|---|
| $-1$ | $1/2$ | 0.2500 | 0.3125 | 0.3750 | 0.4375 |
| 0 | 1 | 0.5000 | 0.6250 | 0.7500 | 0.8750 |
| 1 | 2 | 1.0000 | 1.2500 | 1.5000 | 1.7500 |
| 2 | 4 | 2.0000 | 2.5000 | 3.0000 | 3.5000 |

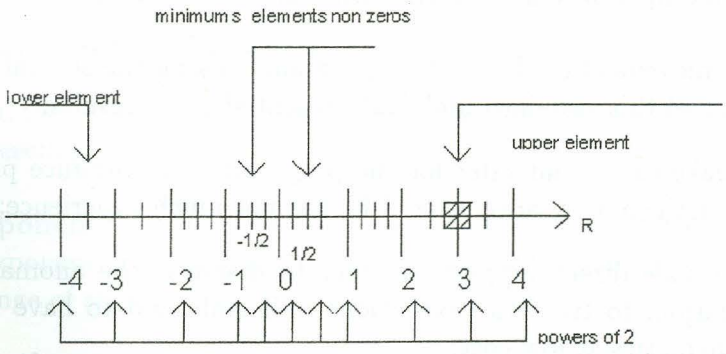


Figure 1: A basic floating-point system $F = F(2, 3, -1, 2)$

We have 33 elements including zero.

We see some properties of floating-point numbers. Let $F$ be a floating-point system described by $F = (b, n, e_1, e_2)$. Then, the system's number of elements, which is denoted by $\#F$ (cardinal of $F$) is given by

$$\#F = 2.(b-1).(b^{n-1}).(e_2 - e_1 + 1) + 1$$

For any mantissa, $m$ is worth $b^{-1} \leqslant \mid m \mid \leqslant 1$. $\mid m \mid < 1$, for every mantissa has as first digit (before the point) the zero. $\mid m \mid < b^{-1}$, for if $\mid m \mid > b^{-1}$ we would not have a normalized number, for the first digit after the point is not null $(\forall x \in F)[-x \in F]$.

If $x$ allows a representation in $F$, then it is necessary just to change the sign of $x$ and then $-x \in F$. Here we do not take into consideration the several codes for the representation of integer negative numbers, for in the case of complement of two, we can have $x \in F$ and $-x \in F$, except if $x$ is the biggest positive integer.

# 3. IEEE Arithmetic Standard

In the beginning of the eighties, an IEEE standard was started, defined as a set of rules that was commercially available for new systems to execute binary floating-point arithmetic.

The relevant topics in the formulation of this standard were:

- easy movement of the existing programs among the several computers that had this standard and that presented equal results;
- to make easier and safer for the programmer to produce programs in the mathematical area, even if he has not much experience;
- to provide direct support in order to diagnose the anomalies during execution, to treat the executions uniformly and to have an interval mathematics at low cost;
- to provide standard elementary functions development, as sine, cosine and exponential, high precision arithmetic (several words) and functions that link symbolic algebraic computation with the numerical one.
- to enable refinements and extensions instead of just prevent them.

IEEE standard specifies the format of:

a) an extended and basic floating-point number;

b) addition, subtraction, multiplication, division, square root, remainder and comparison operations;

c) the conversion between integer and floating-point formats;

d) the conversion between different floating-point formats;

e) the conversion between basic floating-point numbers and decimal strings;

f) the floating-point exceptions and their handling, including non numbers (NaNs).

However, IEEE standard does not specify the formats of:

a) decimal strings (numbers represented in the 10 base) and integer;

b) an interpretation of signal and mantissa fields of nonnumbers (NaNs);

c) a conversion between binary and decimal extended number formats.

## 3.1 Basic Definitions

In this item, the principal definitions used in the specification of IEEE standard are presented.

**Biased exponent:** the sum of a constant (bias) and the exponent, so that the exponent is never negative, making easier the representation. It is a change of scale.

**Binary floating-point number:** a bit-string is characterized by three components: a sign, a signed exponent and a mantissa. The numerical value, if there are any, is the signed product of the mantissa and of the binary base raised to the power of its exponent. In this standard it is not always possible to distinguish a bit-string from the number it may represent.

**Denormalized number:** a non zero floating-point number whose exponent has a reserved value, usually the format's minimum, and whose mantissa's first bit is zero. It is useful to represent lower numbers than the lowest machine number of the normalized standard.

**Destination:** the location for the result of a binary or unary operation. The destination may be either explicitly specified by the user or implicitly supplied by the system (for example, intermediate results in subexpressions or arguments for procedures). Some languages place the results of intermediate calculations in destinations out of the user's control. This standard defines the result of an operation in terms of the destination's format and the operands' values.

**Exponent:** the number two is raised to this component of the binary floating-point number. Occasionally, the exponent is called the signed or unbiased exponent.

**Mode:** a variable that a user may set, save and restore in order to control the execution of a subsequent arithmetic operation. The default mode is the mode that is active in the program, unless an explicit contrary statement is included in either the program or its specifications.

The following mode must be implemented: rounding, to control the direction of rounding errors. In certain implementations, rounding precision may be necessary in order to shorten the precision of the result. The developer may, at his option, implement the following modes: disabled/enabled traps, for handling of exceptions.

**NaNs:** it is a code of a symbolic entity in floating-point format. Its aim is not to interrupt the calculation in situations such as 0/0 and root of a negative number, allowing the computation to go on, but indicating that there has been some invalid calculation. There are two types of NaNs:

a) Signaling NaNs: they signal invalid operation exception, whenever the operations appear as operands.

b) Quiet NaNs: they propagate through almost every arithmetic operation without signaling exceptions.

**Mantissa:** a component of a binary floating-point number that consists of an explicit leading bit to the left of the point and the fraction field to the right.

**Status flag:** a variable that may take two states, set/clear. The user may clear a flag, copy it, restore it to a previous state. When set, the status flag may contain additional system information, possibly inaccessible to some users. The operations of this standard may as side effect set some of the following flags: inexact result, underflow, overflow, division by zero and invalid operation.

**Formats:** this standard defines four floating-point formats divided into two groups, basic and extended, of which precision is single and double. The level of implementation of this standard is characterized by the combinations of the formats supported.

# 4. High Accuracy Arithmetic

The computer was invented to do complicated work for people. The evident discrepancy between computational power and control of computational errors suggest also that we place the error estimation process into the computer. In order to do that, the computer has to be made arithmetically more powerful than the ordinary one. In ordinary floating-point arithmetic, most errors occur in accumulations, that is, by execution of a sequence of additions and subtractions. In fixed-point arithmetic, however, the accumulation operation is performed without errors. Thus, most errors encountered in floating-point can be avoided if the accumulation is performed in fixed-point.

With the current technology we can easily realize the fixed-point accumulation. We only need to provide a fixed-point register in the arithmetic unit which covers the whole floating-point range. If a register with this characteristic is not available, then it can be simulated in the principal memory by software. This naturally results in loss of speed, which in many cases is considered to be more important than the gain in confiability.

If this register has double precision, which is easily possible, then the scalar products of vectors of any finite dimension can always be calculated exactly (the products, that is, the summands in scalar product, have double precision mantissa, and the exponent range is likewise doubled). The possibility of calculating the scalar products of vectors in floating-point of any dimension with total exactness opens a new dimension in numerical analysis. In particular, the optimal scalar product proves itself to be an essential instrument to achieve higher computational accuracy.

In order to adapt the computer also for automatic error control, its arithmetic must be extended with still another element. All the operations with floating-point numbers (addition, subtraction, division and multiplication, and the optimal scalar product of floating-point vectors) must be supplied with directed roundings, that is, roundings to the previous machine number (downward), to the posterior (upward) and to the nearest (symmetric).

An interval arithmetic for real and complex floating-point numbers, as well as for vectors and matrices with real and complex floating-point components can be built with these operations. Intervals bring the continuum into the computer. An interval is represented in the computer by a pair of floating-point numbers. This describes the continuum of real numbers, which is limited by these two floating-point numbers.

The operations on two intervals in the computer result from operations on two appropriately chosen endpoints of the interval operands. In this, the computation of the lower endpoint of the interval result is rounded upward, and the computation of the upper endpoint is rounded upward. The result certainly contains all the results of the operation, individually applying the elements of the first and second interval operands.

# 5. High Performance Arithmetic

The high performance arithmetic is projected to make possible the use of supercomputers in the solution of physical and chemical problems of engineering, among other problems that may need high accuracy. For this, besides vector interval arithmetic (operations, functions and evaluation of expressions), we have to provide libraries that may render available for these users the interval methods for the solution of their problems.

Therefore, in order that we may obtain a high performance arithmetic, this arithmetic must be extended with other elements. All the operations with floating-point numbers must be supplied with directed roundings.

An interval arithmetic for real and complex floating-point numbers can be built with these operations.

In order to guarantee maximum accuracy in interval operations, it is necessary to define the interval operations on the set of floating-point numbers ($\nabla^+, \nabla^-, \nabla^*, \nabla^/, \Delta^+, \Delta^-, \Delta^*$ and $\Delta^/$). These operations would be defined with the help of monotone roundings directed downwards $\nabla$ and upwards $\Delta$. The calculation of the lower endpoint is rounded downwards and the calculation of the upper endpoint is rounded upwards.

Thus, the high performance arithmetic gathers the characteristics of the high accuracy arithmetic, of the interval mathematics and of the vector processing.

Vector processing or vectorization means the introduction of vector instructions of hardware in its programs, in order that the high speed of these instructions may be used so that it can improve the performance of its programs.

The primary objective of vectorization is to find out sequential operations that may be converted into vector operations semantically equivalent, making use of the advantages of the vector hardware. We may obtain the vectorization by re-compiling the scalar codes in a compiler that vectorize them automatically; through re-structuring the source code, helping the compiler in order to obtain a better scale of vectorization and through the development of a new algorithm to explorer the benefits of the vector characteristics of the machine.

The vector processing removes most of tests and control of operations. It minimizes the time spent in the wait for the load of operands and store of the result, when referring groups of memory and through the use of registers of multiple elements. Another advantage of vector processing is that it reduces the number of instructions, in machine language, which needs to be loaded, decoded and executed.

# 6. Conclusions

The floating-point arithmetic is a fast way of doing scientific and engineering calculations. Today, individual floating-point operations are, generally, of maximum accuracy. However, after just two or some few operations, the result may be completely wrong. Computers today perform over $10^{11}$ floating-point operations in a second. Thus, special attention must be given to the validity of the calculated results. Recently, some techniques have been developed in numerical analysis, which make possible that the computer itself verifies the validity of the results calculated for numerous problems and applications. Besides of this, the computer frequently establishes the existence and the uniqueness of the solution in this way.

The result verification is performed by means of mathematical fixed-point theorems, such as Brouwer fixed-point theorem and its generalizations.

The scientific calculation with automatic result verification is of fundamental importance for many applications, for example, for mathematical simulation and modeling. Models that are frequently developed by heuristic methods can only be refined systematically if the computational errors can be completely excluded.

This article gives an introduction for all the scientific computation field with automatic result verification.

# References

[BOH90] Bohlender, G. **What Do We Need Beyond IEEE Arithmetic?** In: Ullrich, C (ed.). Computer Arithmetic and Self-Validating Numerical Methods. Academic Press, 1990.

[IEEE85] American National Standards Institute. Institute of Electrical and Electronics Engineers: **IEEE Standard for Binary Floating-Point Arithmetic.** ANSI/IEEE Std 754-1985, New York, 1985.

[IGR89] IMACS, GAMM: **Resolution on Computer Arithmetic.** Mathematics and Computers in Simulation 31, 297-298, 1989.

[KIR88] Kirchner, R.; Kulisch, U. **Accurate Arithmetic for Vector Processing**. Journal of Parallel and Distributed Computing (Academic Press), vol. 5, N. 3, June 1988.

[KUL81] Kulisch, U.; Miranker, W. L. **Computer Arithmetic in Theory and Practice**. Academic Press, New York, 1981.

[KUL83] Kulisch, U.; Miranker, W. L. (eds.): **A New Approach to Scientific Computation**. Notes and Reports in Computer Science and Applied Mathematics, Academic Press, Orlando, 1983.

[KUL93] Kulisch, U.; Rall, L.B. **Numerics with automatic result verification. Mathematics and Computers in Simulation**. (North Holland). v.35, p.435-450, 1993.

[ULL90] Ullrich, Ch. (ed.) **Computer Arithmetic and Self-Validating Numerical Methods**. Proceedings of SCAN-89, Basel (Oct. 2-6, 1989), Academic Press, 1990.