ON THE NUMERICAL ALGORITHMS FORMULATED IN COMPUTER ARITHMETIC

Svetoslav Markov

Mathematical Institute of the Bulgarian Academy of Sciences

Sofia / BULGARIA

Abstract. We discuss some mathematical tools and techniques supporting the construction of rigorous bounds producing, numerically convergent algorithms, which are formulated in terms of computer arithmetic operations. Two important computer-arithmetic effects are considered and their application as stopping criteria is illustrated.

1. Introduction

In traditional numerical analysis numerical algorithms are formulated in terms of familiar real arithmetic operations. However, real arithmetic is alien to computers: they cannot perform it. Thus, there is a gap between the real-arithmetic form of the algorithms offered by classical numerical analysis and the disability of the computers to execute the arithmetic procedures prescribed by these algorithms. This gap permits rather arbitrary computer realizations of the arithmetic and conversion procedures necessary for the execution of the numerical algorithms. This arbitrariness basically leads to the effect that one and the same algorithm may produce (sometimes completely) different results when run on different type computers even when these computers operate in the same precision. This situation contradicts one of the basic ideas incorporated in the concept of an algorithm, namely the strict and accurate definition of the corresponding (computational) process.

Apart from the above mentioned uncertainty of the computational results, the numerical analyst is faced with the tedious problem of establishing a reliable connection between the correct solutions of the problem and the computational results practically obtained by running the numerical algorithm on a computer. The estimation of the global computational error is usually done through laborious independent estimations of both the truncation and the round-off errors.

The above difficulties can be completely overcome by means of a suitable formulation of the numerical algorithm in terms of computer-arithmetic operations. The usage of a well-defined computer arithmetic allows the construction of numerical algorithms which always produce well-defined intermediate and final results depending only on the chosen computational precision.

Moreover, computer arithmetic supports the construction of numerical algorithms possessing some important properties. Two such properties of particular interest will be formulated in section 3. We shall then discuss some approaches and technical means facilitating the construction of numerical algorithms satisfying these properties.

2. Standard computer arithmetic

For our purposes we shall make use of the standard floating-point computer-arithmetic operations with directed roundings as discussed in [1], [2], [5]. To be more specific let R be the set of reals and $S(b,p)$ be the set of floating-point numbers handled by the machine we are using; b is the base of the number system; p is the number of base b digits contained in the mantissa of the floating-point numbers (it will be further called computational precision); for simplicity we assume no bounds on the exponent.

If $a \in R$, denote by ∇a the largest number in $S(b,p)$ which is $\leq a$, and by Δa the smallest number in $S(b,p)$ which is $\geq a$. The mappings $\nabla, \Delta : R \to S(b,p)$ are called directed (downwardly and upwardly, resp.) roundings in R. These roundings generate the following computer arithmetic in $S(b,p)$: if $a,b \in S(b,p)$ and $\circ \in \{+,-,\times,/\}$, then $a \,\triangledown\, b = \nabla(a \circ b)$ and $a \,\triangle\, b = \Delta(a \circ b)$. We shall assume that the arithmetic operations $\triangle, \Delta, \triangle, \Delta, \nabla, \nabla, \nabla, \nabla$ are present on our computer. Assume also that programs for conversion of input data are available which for each $a \in R$ may produce ∇a or $\Delta a \in S(b,p)$; resp. programs for conversion of output data are available.

Example 1. Consider the Newton method for the computation of $\sqrt{a}$, $a \in R$:

(1) $x_0 = a$; $x_{n+1} = (x_n + a/x_n)/2$, $n = 0,1,2,\ldots$.

The result of the computations depend on the rounding mode of the operations. For instance, let $a=2$ and $b=10$, $p=2$. If we round all operations up we get on the se-

CH1892-9/83/0000/0082$01.00 © 1983 IEEE

cond iteration $x_2=1.5$, and if we round all operations down, then $x_2=1.4$ .Consider in some detail the algorithm (1) when all operations are rounded up:

(2) $\bar{x}_0=\bar{a}$; $\bar{x}_{n+1}=(\bar{x}_n \bar{\oplus} (\bar{a} \bar{\Delta} \bar{x}_n)) \bar{\Delta} 2$, $n=0,1,\ldots$

where $\bar{a}=\Delta a$ (rounding of a may be necessary due the fact that $a \bar{\in} S(b,p)$). The algorithm (2) produces a well-defined sequence of machine numbers as long as the system $S(b,p)$ is fixed.Thus for $a=2$ we obtain in $S(10,p)$ with $p=1,2,\ldots,7$ the following sequences ($\bar{x}_0=2.$ and $\bar{x}_1=1.5$ for all p):

p	$\bar{x}_2$ =	$\bar{x}_3$ =	$\bar{x}_4$ =	$\bar{x}_5$ =
1	$\bar{x}_0$ (=2.)	$\bar{x}_0$	$\bar{x}_0$	$\bar{x}_0$
2	$\bar{x}_1$(=1.5)	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$
3	1.42	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$
4	1.417	1.415	$\bar{x}_3$	$\bar{x}_3$
5	1.4167	1.4143	$\bar{x}_3$	$\bar{x}_3$
6	1.41667	1.41422	$\bar{x}_3$	$\bar{x}_3$
7	1.416667	1.414216	1.414214	$\bar{x}_4$

We shall point out some interesting properties of (2). It produces a sequence of numbers $\bar{x}_i$, $i=1,2,\ldots$, such that $\bar{x}_i \geq X=\sqrt{a}$, that is $\bar{x}_i$ is always a right bound for the true solution X. The finite computational precision p does not allow $\bar{x}_i$ to get arbitrarily close to X and we have $\bar{x}_j=\bar{x}_{j+1}=\ldots$ for some integer $j \geq 0$. Following Moore (see [8],p.36) we shall call this "finite convergence". It can be used as a natural stopping criterion, so that (2) becomes:

(2′) $\begin{cases} \bar{x}_0=\bar{a}, \ \bar{x}_{n+1}=(\bar{x}_n \bar{\oplus} (\bar{a} \bar{\Delta} \bar{x}_n)) \bar{\Delta} 2, n=0,1, \text{etc.} \\ \text{until relation } \bar{x}_{n+1} < \bar{x}_n \text{ is violated.} \end{cases}$

Note that for a specified input data a, the final computational result depends only on the computational precision p. For every positive integer p we obtain a well-defined final result $\bar{x}=\bar{x}(p)$, and $\bar{x}(p) \to X$ with $p \to \infty$. To summarize: i) the algorithm (2′) produces well-defined results depending on nothing but the computational precision p; ii) the algorithm produces a rigorous right bound for the correct result; iii) this bound converges to the solution with $p \to \infty$.
Remark. By rigorous bound we mean, in contrast to traditional numerical analysis, a reliably guaranteed, absolutely sure, practically secured bound.

3.Rigorous bounds producing, numerically convergent algorithms

We are particularly interested in numerical algorithms formulated in computer-arithmetic form, which satisfy the following two properties:
RB-property. For any arbitrarily chosen computational precision p the numerical algorithm produces well-defined rigorous bounds for the solution of the problem.
NC-property. The algorithm is numerically convergent to the solution, that is, the bounds computed with precision p tend to the correct solution with $p \to \infty$.
Remarks. The abbreviations RB and NC stand for rigorous (or: reliable) bounds and numerical convergence, respectively. The meaning of "solution" and "convergence" needs some explanation; however, we shall not go into detail, since the meaning of these terms is closely connected to the specific numerical problem in consideration (e.g.by solution we may mean the so-called interval hull of the solution set, the convergence is related to a certain metrics, etc.).

A numerical algorithm satisfying the RB- and the NC-properties will be further called a rigorous bounds producing, numerically convergent algorithm, briefly:an RBNC-algorithm. An RBNC-algorithm produces reliably guaranteed bounds containing the solution and these bounds tend to the solution when increasing the computational precision.

We shall next briefly discuss some mathematical means and techniques supporting the construction of RBNC-algorithms. The well-known simple interval arithmetic [1,8] is a very efficient tool for creating algorithms satisfying the RB-property. However, it is not so efficient when constructing algorithms satisfying the NC-property. Serious obstacles arrise in overcoming the simultaneity problem (see e.g. [9],p.210). A more sophisticated mathematical tool supporting the construction of RBNC-algorithms is extended interval arithmetic [6,7]. The technique of obtaining RBNC-algorithms by means of extended interval arithmetic is described and illustrated in [3].In the process of elaborating this technique we arrived to some other approaches for formulation of RBNC-algorithms without using interval arithmetic, like the "two-sided approach" and the "one-sided approach".

We shall give some definitions. Assume that X is the solution of a given numerical problem (under suitable input data) in a suitable ordered metric space S.Suppose we have an iterative algorithm of the form:

(A) $\begin{cases} x_0,x_1,\ldots,x_k \text{ given,} \\ x_{n+1}=f(x_n,x_{n-1},\ldots,x_{n-k}), n=k,k+1,\ldots \end{cases}$

where $k \geq 0$ is a fixed integer, $x_i \in S$, $i=0,1,\ldots$, and f is a computable function defined on $S \times S \times \ldots \times S$.The algorithm (A) is called a left (resp. right) one-sided BC-algorithm if it satisfies the requirements:
B-property: $x_n \leq X$ (resp.$x_n \geq X$), $n=0,1,\ldots$;
C-property: $x_n \to \inf X$ (resp. $x_n \to \sup X$).
Remark. $[\inf X, \sup X]$ denotes the interval hull of X; the inf and sup signs can be omitted in the one-dimensional case.
A pair of two one-sided algorithms for solving a given problem, one of which is left and the other right, will be called a two-sided algorithm. A two-sided algorithm is a two-sided BC-algorithm if both one-sided algorithms involved are one-sided BC-algorithms.

4. Construction of RBNC-algorithms

4.1. Two-sided approach

The two-sided approach for constructing an RBNC-algorithm consists of two steps:
Step 1. Construct a two-sided BC-algorithm, defining a sequence $l_0, l_1, l_2, \ldots$ of left one-sided approximations and a sequence $r_0, r_1, r_2, \ldots$ of right one-sided approximations converging to the correct solution X.
Step 2. Perform a computer-arithmetic realization of the two-sided BC-algorithm transforming it into an RBNC-algorithm. The computer-arithmetic realization consists of a suitable replacement of the real-arithmetic operations involved in the two-sided algorithm, by computer-arithmetic operations, in such a manner that at each iteration (n) the computed (in computer-arithmetic) interval $[\underline{l}_n, \bar{r}_n]$ contains the true interval $[l_n, r_n]$ defined by the BC-algorithm.

Some applications of this approach and the corresponding technical details are given in [3,4]. The following simple example may serve as illustration of this approach.

Example 2. For the computation of e^x for small nonnegative x (say, $0 \le x \le 1$) a two-sided BC-algorithm may be constructed on the basis of the relations $s_n(x) \le e^x \le r_n(x)$, $s_n(x) = 1+x+x^2/2! + \ldots + x^n/n!$, $r_n(x) = (1+x+x^2/2! + \ldots + x^n/n!)/(1-x^{n+1}/(n+1)!)$.
This algorithm may be written in an iteration form as:

$a_0 = s_0 = 1$,
$a_n = a_{n-1}(x/n)$, $s_n = s_{n-1} + a_n$,
$n = 1,2,\ldots$,

for the left bound, and

$a_0 = s_0 = 1$,
$a_n = a_{n-1}(x/n)$, $r_{n-1} = s_{n-1}/(1-a_n)$,
$s_n = s_{n-1} + a_n$, $n = 1,2,\ldots$,

for the right bound.
Denote by $[\underline{x}, \bar{x}]$ the smallest machine interval for x. The computer-arithmetic realization of the above BC-algorithm yields the following RBNC-algorithm:

$$(3)\quad \begin{cases} \underline{a}_0 = \underline{s}_0 = 1, \\ \underline{a}_n = \underline{a}_{n-1} \underline{\nabla} (\underline{x} \underline{\nabla} n), \quad \underline{s}_n = \underline{s}_{n-1} \underline{\nabla} \underline{a}_n; \\ n = 1,2,\ldots; \\ \bar{a}_0 = \bar{s}_0 = 1, \\ \bar{a}_n = \bar{a}_{n-1} \bar{\Delta} (\bar{x} \bar{\Delta} n), \quad \bar{r}_{n-1} = \bar{s}_{n-1} \bar{\Delta} (1 \bar{\nabla} \bar{a}_n), \\ \bar{s}_n = \bar{s}_{n-1} \bar{\Delta} \bar{a}_n; \quad n = 1,2,\ldots . \end{cases}$$

We give below the results of the computation of e^x for x=1 by means of the above algorithm in S(10,3), that is in decimal floating-point arithmetic with precision 3. Let us emphasize once more that the computed values of the variables involved in the above formulas depend only on the specified precision p. Thereby it is of no consequence whether a computer is used or the necessary computations are done by hand. The computational results are given in the following table:

n	$\underline{a}_n$	$\underline{s}_n$	$\bar{a}_n$	$1 \bar{\nabla} a_{n+1}$	$\bar{r}_n$	$\bar{s}_n$
0	1.00	1.00	1.00	.000	4.00	1.00
1	1.00	2.00	1.00	.500	3.01	2.00
2	.500	2.50	.500	.833	2.79	2.50
3	.166	2.66	.167	.958	2.75	2.67
4	.0415	2.70	.418	.991	2.74	2.72
5	.00830	2.70	.00836	.998	2.75	2.73
6	.00138	2.70	.00140	.999	2.76	2.74

The effect of finite convergence, mentioned in sec.3, is clearly seen. Using it as stopping criterion we obtain as final result the interval [2.70, 2.74].

A program system supporting the execution of RBNC algorithms should be able to carry out the calculations several times in different precision. If this can be automatically organized, then the user may prescribe the necessary accuracy of the final result. We give below the final results of the computation of e by means of formula (3) when using five different precisions p:

p	1	2	3	4	5
$\underline{s}$	2	2.6	2.70	2.716	2.7179
$\bar{r}$	4	2.9	2.74	2.721	2.7185

4.2. One-sided approach

For a large class of numerical problems it is often comparatively easy to construct a one-sided BC-algorithm (left or right), but is very difficult or impossible to construct another one-sided BC-algorithm producing approximations from the other side of the solution. We shall demonstrate that it is possible to obtain an RBNC-algorithm on the basis only of a one-sided BC-algorithm. using some purely computer-arithmetic effects.
Consider first the following computer-arithmetic realization of the one-sided BC-algorithm (1) producing computed $\underline{x}_i$, which are always to the left of the corresponding real x_i:

$$(4)\quad \underline{x}_0 = \underline{a}, \quad \underline{x}_{n+1} = (\underline{x}_n \underline{\nabla} (\underline{a} \underline{\nabla} \underline{x}_n)) \underline{\nabla} 2, \quad n = 0,1,\ldots,$$

where $\underline{a} = \underline{\nabla} a$. Some results of the computation of $\sqrt{a}$ by means of (4) for a=2, b=10 are presented below ($\underline{x}_0 = 2.$ for all p; $\underline{x}_1 = 1.$ for p=1 and $\underline{x}_1 = 1.5$ for all $p \ge 2$):

p	$\underline{x}_2$ =	$\underline{x}_3$ =	$\underline{x}_4$ =	$\underline{x}_5$ =
1	$\underline{x}_1$ (=1.)	$\underline{x}_1$	$\underline{x}_1$	$\underline{x}_1$
2	1.4	$\underline{x}_2$	$\underline{x}_2$	$\underline{x}_2$
3	1.41	$\underline{x}_2$	$\underline{x}_2$	$\underline{x}_2$
4	1.416	1.414	$\underline{x}_3$	$\underline{x}_3$
5	1.4166	1.4142	$\underline{x}_3$	$\underline{x}_3$
6	1.41666	1.41421	$\underline{x}_3$	$\underline{x}_3$
7	1.416666	1.414215	1.414213	$\underline{x}_4$

A computer-arithmetic effect, similar to the effect of finite convergence, is to be observed in the above calculations. As in algorithm (2) we see that for some $j \geq 0$, depending on p, we have $\underline{x}_j = \underline{x}_{j+1} = \ldots = \underline{x}(p)$. This effect is intrinsic for this particular algorithm and is mainly due to the fact that truncation error rapidly becomes insignificant. For every chosen p, $\underline{x}(p)$ presents a rigorous left bound for the solution. We thus have an algorithm producing a reliable left bound for $\sqrt{a}$:

$$(4') \begin{cases} \underline{x}_0 = \underline{a}, \quad \underline{x}_{n+1} = (\underline{x}_n \overline{\nabla} (\underline{a} \nabla \underline{x}_n)) \nabla 2, \quad n=0,1,2, \\ \text{until} \quad \underline{x}_{n+1} = \underline{x}_n. \end{cases}$$

Formulas (2') and (4') considered together present an RBNC-algorithm for the computation of $\sqrt{a}$. They serve as a good illustration of the fact that an RBNC-algorithm can be obtained only on the basis of a one-sided BC-algorithm.

Consider one more example in the same direction, where a more typical computer-arithmetic effect takes place. To this end we return to example 2. Using the expression $s_n(x) = 1 + x + x^2/2 + \ldots + x^n/n!$ we constructed the following one-sided BC-algorithm for computation of left bound for $e^x, x \geq 0$:

$$(5) \begin{cases} a_0 = s_0 = 1, \\ a_n = a_{n-1}(x/n), \quad s_n = s_{n-1} + a_n, \quad n=1,2,\ldots \end{cases}$$

By means of suitable roundings we obtained the following one-sided algorithm producing rigorous left bound for exp(x):

$$(6) \begin{cases} \underline{a}_0 = \underline{s}_0 = 1, \\ \underline{a}_n = \underline{a}_{n-1} \overline{\nabla} (\underline{x} \nabla n), \quad \underline{s}_n = \underline{s}_{n-1} \overline{\nabla} \underline{a}_n, \\ n=1,2,\ldots, \text{ until } \underline{s}_{n+1} \leq \underline{s}_n. \end{cases}$$

Consider now the following algorithm which produces computed values $\bar{s}_n$ of s_n which are $\geq$ then the real s_n defined by (5):

$$(7) \begin{cases} \bar{a}_0 = \bar{s}_0 = 1, \\ \bar{a}_n = \bar{a}_{n-1} \overline{\Delta} (\bar{x} \Delta n), \quad \bar{s}_n = \bar{s}_{n-1} \overline{\Delta} \bar{a}_n, n=1,\ldots \end{cases}$$

Let us take a look at the table in sec.4.1. where the values $\bar{s}_n$ as computed by (7) are presented. From a certain index N onward the values $\bar{s}_N, \bar{s}_{N+1}, \ldots$ present rigorous bounds for the correct solution. This is precisely the place where $\bar{s}_n$ starts to increase with just one unit in the last position (the value of $\bar{s}_n$ jumps to next machine number). This effect is due to the fact that truncation error has become negligible, but since we have to add a very small positive term $\bar{a}_n$ to the positive sum $\bar{s}_{n-1}$ and the addition is performed in rounding up mode we have to increase $\bar{s}_{n-1}$ by one unit in the last position (or, in other words: to pass to the next machine number). It is to be noted that some published algorithms for round up addition produce $a \overline{\Delta} b = a$ for $a > 0$ and sufficiently small $b > 0$ (which is theoretically wrong, but may surve certain purposes). Such is the case with the algorithm of Yohe [10].

Having in mind the discussed computer-arithmetic effect, we obtain the following simple algorithm producing a rigorous right bound for the value of exp(x):

$$(7') \begin{cases} \bar{a}_0 = \bar{s}_0 = 1, \\ \bar{a}_n = \bar{a}_{n-1} \overline{\Delta} (\bar{x} \Delta n), \quad \bar{s}_n = \bar{s}_{n-1} \overline{\Delta} \bar{a}_n, \\ n=1,2,\ldots, \text{until } (\bar{s}_n, \bar{s}_{n+1}) \text{ contains no} \\ \text{machine numbers.} \end{cases}$$

The pair (6)-(7') presents a RBNC-algorithm. An important moment in the construction of RBNC-algorithms is the suitable computer-arithmetic realization of the corresponding one-sided algorithms. The technique of such a realization is presented in some detail in [3,4]. Both approaches discussed above proved to be extremely suitable for the construction of RBNC-algorithms for computation of Taylor and harmonic series (in particular: elementary and special functions). Computational results show that these approaches are practically very effective and suggest that they can be used for solving more general numerical problems.

The accomplishment of the computer-arithmetic realization of the algorithms opens a wide field for sophisticated usage of various computer-arithmetic effects such as the effects described above of "finite convergence" and of "passing to the next machine number". We believe that the mastery of these techniques will lead to an extension of the classes of problems, which can be reliably solved numerically.

REFERENCES

1. G.Alefeld, J.Herzberger. Einführung in die Intervallrechnung. Mannheim, Bibl. Inst., 1974.
2. J.Coonen, et al. A proposed standard for binary floating-point arithmetic. ACM Signum Letters, Special Issue, Oct. 1979, 4-12.
3. N.Dimitrova, S.Markov. Interval methods of Newton type for nonlinear equations, Pliska Studia math. bulg. 5, 1982, 105-117.
4. N.Kjurkchiev, S.Markov. Two interval methods for algebraic equations with real roots. Pliska 5, 1982, 118-131.
5. U.Kulisch, W.Miranker. Computer arithmetic in theory and practice. Acad. Press, 1981.
6. S.Markov. Some applications of the extended interval arithmetic to interval iterations. Computing Suppl. 2, 1980, 69-84.
7. S.Markov. On an interval arithmetic and its applications. Proc. of the 5th Symposium on computer arithmetic, Ann Arbor, May 18-19, 1981, IEEE Computer Society Press, 1981, 274-277.
8. R.Moore. Methods and applications of interval analysis. SIAM Studies in Appl. Math., SIAM, Philadelphia, 1979.
9. P.Sterbenz. Floating-point computation, Prentice-Hall, Englewood Cliffs, N.J., 1974.
10. J.Yohe. Roundings in floating-point computations, IEEE Trans. Comp. C-22, 1973, 577-586.