FLOATING-POINT COMPUTATION OF FUNCTIONS WITH MAXIMUM ACCURACY

Gerd Bohlender
Institut für Angewandte Mathematik der Universität
D-75 Karlsruhe
Kaiserstr. 12
West Germany

Summary: Algorithms are given which compute multiple sums and products and arbitray roots of floating-point numbers with maximum accuracy. The summation algorithm can be applied to compute scalar products, matrix products, etc. For all these functions, simple error formulas and the smallest floating-point intervals containing the exact result can be obtained.

0. Introduction

Our aim is to approximate functions $f\colon \mathbb{R}^n \to \mathbb{R}^p$ on a floating point system T. For $b, l \in \mathbb{N}$, $b \geq 2$, $l \geq 1$, the floating-point system $T_{b,l}$ with base b and l-digit mantissa is defined by

$$T_{b,l} := \{o\} \cup \{x = *m\cdot b^e;\ * \in \{+,-\},\ m = o.m[1]\ldots m[l],\ m[i] \in \{o,1,\ldots,b-1\}, m[1] \neq o,\ e \in \mathbb{Z}\}. \quad (1)$$

x is then called a floating-point number with sign $* = \mathrm{sgn}(x)$, mantissa $m = \mathrm{mant}(x)$ and exponent $e = \exp(x)$. As the base b will be kept fixed throughout the paper, we will suppress the index b and write shortly T_l or T. For the present, we do not consider the finite exponent range which is available in practice, as this would necessitate complicated exponent-overflow and -underflow discussions. Instead, we give remarks on the influence of limiting the exponent range on our algorithms.

The best possible approximation for f(x) is $\square f(x)$, wherein $\square\colon \mathbb{R}^p \to T^p$ denotes a rounding*). We will restrict ourselves here to the roundings ∇, Δ and $\square_\mu$ ($\mu = o(1)b$). For p=1 these roundings are defined as follows:

$$\bigwedge_{x\in\mathbb{R}} \nabla x := \max\{y \in T; y \leq x\}, \quad (2)$$

$$\bigwedge_{x\in\mathbb{R}} \Delta x := \min\{y \in T; x \leq y\} = -\nabla(-x), \quad (3)$$

$$\bigwedge_{x\geq o} \square_b x := \nabla x \wedge \bigwedge_{x<o} \square_b x := \Delta x, \quad (4)$$

$$\bigwedge_{x\geq o} \square_o x := \Delta x \wedge \bigwedge_{x<o} \square_o x := \nabla x, \quad (5)$$

and for $\mu = 1(1)b-1$:

$$\bigwedge_{x\geq o} \square_\mu x := \begin{cases} \nabla x & \text{for } x \in [\nabla x, S_\mu(x)) \\ \Delta x & \text{for } x \in [S_\mu(x), \Delta x] \end{cases},$$

$$\bigwedge_{x<o} \square_\mu x := -\square_\mu(-x), \quad (6)$$

wherein the function $S_\mu\colon \mathbb{R} \to \mathbb{R}$ is defined by

$$S_\mu(x) := \nabla x + (\Delta x - \nabla x)\cdot\frac{\mu}{b}.$$

∇ and Δ are called downwardly and upwardly directed rounding resp., $\square_b$ is called rounding towards zero and $\square_o$ is called rounding away from zero. If b is an even number, then $\bigcirc := \square_{b/2}$ is the rounding to the closest floating-point number. For p > 1 the roundings ∇, Δ and $\square_\mu$ ($\mu = o(1)b$) are defined componentwise.

With these roundings, $[\nabla f(x), \Delta f(x)]$ is the smallest floating-point interval containing f(x); and if b is even, then $\square_{b/2} f(x)$ is an approximation of f(x) with maximum accuracy. So the computation of interval bounds for f(x) and the computation of optimal approximations of f(x) can be substituted by the more general task of computing $\square f(x)$ for all $\square \in \{\nabla, \Delta, \square_\mu (\mu = o(1)b)\}$. For these approximations $\square f(x)$ of f(x), the following theorem can be proved, simply by applying the properties of the roundings ∇, Δ and $\square_\mu$ that are studied by Kulisch [5].

Theorem 1: For a function $f\colon \mathbb{R}^n \to \mathbb{R}^p$ and a floating-point system $T = T_{b,l}$ the following properties hold:

(a) $\bigwedge_{x\in\mathbb{R}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} (f(x) \in T^p \Rightarrow \square f(x) = f(x))$,

(b) $\bigwedge_{x,y\in\mathbb{R}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} (f(x) \leq f(y) \Rightarrow \square f(x) \leq \square f(y))$,

(c) $\bigwedge_{x\in\mathbb{R}^n} \bigwedge_{\square\in\{\square_\mu;\mu=o(1)b\}} (f(-x) = -f(x) \Rightarrow \square f(-x) = -\square f(x))$,

$\bigwedge_{x\in\mathbb{R}^n} (f(-x) = -f(x) \Rightarrow \nabla f(-x) = -\Delta f(x) \wedge \Delta f(-x) = -\nabla f(x))$,

(d) $\bigwedge_{x\in\mathbb{R}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} |f(x) - \square f(x)| \leq \varepsilon^* \cdot |f(x)|$,

wherein absolute values are defined componentwise and

$$\varepsilon^* := \begin{cases} \frac{1}{2} b^{1-l}, & \text{if b is even and } \square = \square_{b/2} \\ b^{1-l}, & \text{else} \end{cases}$$

(e) $\bigwedge_{g\colon\mathbb{R}^n\to\mathbb{R}^p} \bigwedge_{x\in\mathbb{R}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} (f(x) \leq g(x) \Rightarrow \square f(x) \leq \square g(x))$.

*) As regards general definitions, we refer to Kulisch [5].

Remark: If the exponent range is limited by $e1,e2\in\mathbb{Z}$, then property (d) is only valid if no exponent-overflow and -underflow occur, i.e. if $|f(x)|\in[b^{e1-1},(1-b^{-1})b^{e2}]^p$.

As we want to execute the function f on a floating-point system T, only arguments $x\in T^n$ have to be considered. The task of computing $\square f(x)$ for all $\square\in\{\nabla,\Delta,\square_\mu\ (\mu=o(1)b)\}$ and all $x\in T^n$ was solved by Kulisch [6] and Yohe [13] for the arithmetic operations $+,-,\cdot,/$ and by Kulisch and Bohlender [7] for the sum of n floating-point numbers as well as for the scalar product. Yohe [12] finally gave a modification of Newton's method which allows the computation of $\nabla\sqrt{x}$ and $\Delta\sqrt{x}$ in the special case b=2.

In the present paper, algorithms shall be proposed for the functions $\sum_{i=1}^n x_i$, $\prod_{i=1}^n x_i$, $\sqrt[n]{x}$, wherein $x_i\in T$, $x\in T$, and for functions that are derived from the sum.

Several algorithms for improving accuracy in floating-point summation are known (e.g. Kahan [4], Linz [8], Malcolm [9], Pichat [10]). But these algorithms are not intended to deliver results with maximum accuracy, smallest interval bounds, and the properties of theorem 1. The algorithm of Kulisch and Bohlender [7] sorts the numbers x_i according to their exponent. For large n this consumes much time, as the time for sorting n numbers is at least proportional to $n\cdot\lceil\log_2 n\rceil$. The algorithm in the present paper is based on the one of Pichat [10]; it manages without sorting.

For the computation of roots of floating-point numbers, a modification of Newton's method is used which determines - under appropriate assumptions - the smallest floating-point interval containing the zero of a function. This method is applied on the polynomial x^n-a. In the case b=n=2 it is equivalent with the algorithm of Yohe [12].

1. The sum of n floating-point numbers

For the computation of the sum $\square\sum_{i=1}^n x_i$ of n floating-point numbers x_i, we need not determine $\sum_{i=1}^n x_i$. Instead, we can compute a simpler approximation $\widetilde{\sum_{i=1}^n x_i}$ with the property

$$\bigwedge_{(x_i)\in T^n}\ \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu(\mu=o(1)b)\}} \square\sum_{i=1}^n x_i = \square\widetilde{\sum_{i=1}^n x_i}\,. \quad (7)$$

We will give an algorithm that computes such an approximation; algorithms for the roundings $\nabla,\Delta,\square_\mu$ $(\mu=o(1)b)$ can be found in Kulisch [6].

In the following lemmas, we introduce the notation and the iteration method which will be needed in the summation algorithm.

Let us first define a binary relation $\prec$ on $T_b^* := \bigcup_{l=1}^{\infty} T_{b,l}$:

$$\bigwedge_{x,y\in T_b^*}(x\prec y \;:\Longleftrightarrow\; x=o \vee y=o \vee y\in T_{\exp(y)-\exp(x)}) \quad (8)$$

$x\prec y$ means that y=o or all digits of x have smaller exponents than all nonzero digits of y.

Lemma 1: Let $T=T_{b,l}$ be a floating-point system and $\bigcirc:\mathbb{R}\to T$ the rounding to the closest floating-point number. Then for all $x,y\in T$ the following properties hold:

(a) $s:=\bigcirc(x+y)\in T,\quad r:=(x+y)-s\in T$,

(b) $r\prec s$,

(c) $\bigwedge_{z\in T}(z\prec x \wedge z\prec y \Rightarrow z\prec r \wedge z\prec s)$,

(d) $\bigwedge_{z\in T}(x\prec z \vee y\prec z \Rightarrow r\prec z)$.

Proof: If x=o or y=o, then r=o. So (a),(b),(c),(d) are fulfilled. Else we define d:=exp(x)-exp(y) and assume without loss of generality that $d\geq o$.

Case 1: $d>l \Rightarrow s=x \Rightarrow r=y \Rightarrow$ (a),(b),(c),(d).

Case 2: $d\leq l \wedge x+y=o \Rightarrow s=r=o \Rightarrow$ (a),(b),(c),(d).

Case 3: $d\leq l\wedge x+y\neq o \Rightarrow x+y\in T_{b,2l}\setminus\{o\} \Rightarrow$
$\Rightarrow x+y = *o.m[1]\ldots m[2l]\cdot b^e$, with $*\in\{+,-\}$, $m[i]\in\{o,1,\ldots,b-1\}, m[1]\neq o$, $e\in\mathbb{Z}$. Then s and r fulfil one of the following two properties:

(α) $s = *o.m[1]\ldots m[l]\cdot b^e,\ r = *o.m[l+1]\ldots m[2l]\cdot b^{e-l}$,

(β) $s = *(o.m[1]\ldots m[l]+b^{-l})\cdot b^e,\ r = *(o.m[l+1]\ldots m[2l]-1)\cdot b^{e-l}$.

Therein r and s may be denormalized and r may even be 0. In both cases, (a) and (b) are evident; (c) follows from

$z\prec x\wedge z\prec y \Rightarrow z\prec x+y$;

(d) finally follows from

$r=o\vee \exp(r)\leq\min(\exp(x),\exp(y))$. ∎

Writing shortly $(s,r) := x+y$ for $s:=\bigcirc(x+y)$, $r:=(x+y)-s$, we can now formulate the iteration method which is the basis of our summation algorithm.

Lemma 2: Let $T=T_{b,l}$ be a floating-point system. Starting with $x^{(o)}\in T^n$, a sequence $(x^{(k)})_{k=o,1,2,\ldots}$, $x^{(k)}=(x_1^{(k)},x_2^{(k)},\ldots,x_n^{(k)})\in T^n$ is recursively defined by

$$s_1 := x_1^{(k)},$$
$$(s_{p+1},x_p^{(k+1)}) := s_p + x_{p+1}^{(k)},\quad p=1(1)n-1, k=o,1,2,\ldots \quad (9)$$

$x_n^{(k+1)} := s_n$,

wherein $s_p \in T$ are auxiliary variables.
Then the sequence $(x^{(k)})_{k=o,1,2,\dots}$ has the following properties:

$$\text{(a)} \bigwedge_{k\in\mathbb{N}} \sum_{i=1}^{n} x_i^{(k)} = \sum_{i=1}^{n} x_i^{(o)} ,$$

$$\text{(b)} \bigwedge_{k\in\mathbb{N}} \bigwedge_{i,j\in\{1,\dots,n\}} (n-k\leqq i<j \Rightarrow x_i^{(k)} \prec x_j^{(k)}) .$$

Proof: (a) is an immediate consequence of lemma 1,(a). (b) is proved by induction over k. It is trivial for k=o, so let us assume that it is valid for any $k\in\mathbb{N}$.

Case 1: If $k \geqq n-1$, then $x_i^{(k+1)}=x_i^{(k)}$ $(i=1(1)n)$, so (b) is proved.

Case 2: If $k < n-1$, then $x_i^{(k)} \prec x_j^{(k)}$ whenever $n-k\leqq i<j\leqq n$. For $p=n-k-1(1)n$, the following properties can again be proved inductively:

$$n-k-1\leqq i<j\leqq p-1 \Rightarrow x_i^{(k+1)} \prec x_j^{(k+1)}, \tag{10}$$

$$n-k-1\leqq i\leqq p-1 \Rightarrow x_i^{(k+1)} \prec s , \tag{11}$$

$$n-k-1\leqq i\leqq p-1, p+1\leqq j\leqq n \Rightarrow x_i^{(k+1)} \prec x_j^{(k)}, \tag{12}$$

$$p+1\leqq i<j\leqq n \Rightarrow x_i^{(k)} \prec x_j^{(k)} \tag{13}$$

Properties (10),(11),(12) are clear for p=n-k-1 and property (13) is clear for all p=n-k-1(1)n. So let (10),(11),(12) be valid for any $p\in\{n-k-1,\dots,n-1\}$; then (10),(11),(12) can be proved for p+1 as follows:

$$(\alpha)\ (11),(12) \Rightarrow \bigwedge_{n-k-1\leqq i\leqq p-1} (x_i^{(k+1)} \prec s \wedge$$

$$\wedge\ x_i^{(k+1)} \prec x_{p+1}^{(k)}) \underset{11.(c)}{\Rightarrow} \bigwedge_{n-k-1\leqq i\leqq p-1} (x_i^{(k+1)} \prec s_{p+1} \wedge$$

$$\wedge\ x_i^{(k+1)} \prec x_p^{(k+1)}) \underset{11.(b),(d)}{\Rightarrow} \bigwedge_{n-k-1\leqq i\leqq p} (x_i^{(k+1)} \prec s_{p+1} \wedge$$

$$\wedge\ x_i^{(k+1)} \prec x_p^{(k+1)}) \underset{(10)}{\Rightarrow} (10),(11) \text{ for } p+1 .$$

$$(\beta)\ (13) \Rightarrow \bigwedge_{p+2\leqq j\leqq n} x_{p+1}^{(k)} \prec x_j^{(k)} \underset{11.(d)}{\Rightarrow} \bigwedge_{p+2\leqq j\leqq n} x_p^{(k+1)} \prec x_j^{(k)}$$

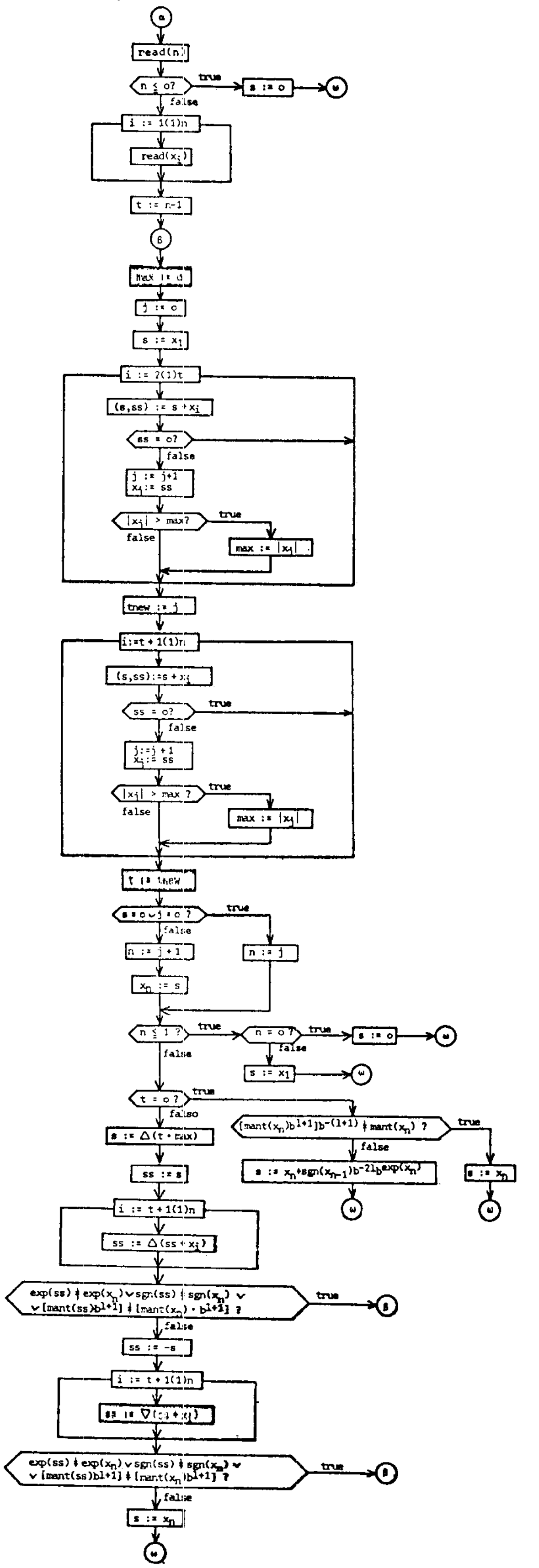
$$\underset{(12)}{\Rightarrow} (12) \text{ for } p+1 .$$

Therefore (10),(11),(12) are valid for all p=n-k-1(1)n. In the case p=n we get from (10) and (11):

$$n-(k+1)\leqq i<j\leqq n \Rightarrow x_i^{(k+1)} \prec x_j^{(k+1)},$$

which concludes the induction step over k. Thus (b) is proved for all $k\in\mathbb{N}$. ■

In algorithm 1, the method from lemma 1 is applied in $T_{b,2l}$ on the computation of the sum of n double-length floating-point numbers. Some modifications are made: the index t takes the role of n-k in property (b) of the lemma; zeros are eliminated; the iteration is stopped, if the result can be determined from the summands easily.



Algorithm 1: Sum of n floating-point numbers

Theorem 2: Let $T_1=T_{b,1}$ be a floating-point system and $x_i \in T_{21}$ ($i=(1)n$) n double-length floating-point numbers. Then algorithm 1 determines an approximation $s = \widetilde{\sum_{i=1}^{n} x_i} \in T_{21}$ of $\sum_{i=1}^{n} x_i$ with the property

(a) $\bigwedge_{(x_i)\in T_{21}^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu(\mu=o(1)b)\}} \square \sum_{i=1}^{n} x_i = \square s.$

Furthermore the following properties hold:

(b) $\bigwedge_{(x_i)\in T_{21}^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} (\sum_{i=1}^{n} x_i \in T_1 \Rightarrow \square \sum_{i=1}^{n} x_i =$

$= \sum_{i=1}^{n} x_i),$

(c) $\bigwedge_{(x_i),(y_i)\in T_{21}^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} (\sum_{i=1}^{n} x_i \leqq \sum_{i=1}^{n} y_i \Rightarrow$

$\Rightarrow \square \sum_{i=1}^{n} x_i \leqq \square \sum_{i=1}^{n} y_i),$

(d) $\bigwedge_{(x_i)\in T_{21}^n} \bigwedge_{\square\in\{\square_\mu(\mu=o(1)b)\}} \square \sum_{i=1}^{n} (-x_i) = -\square \sum_{i=1}^{n} x_i,$

$\bigwedge_{(x_i)\in T_{21}^n} (\nabla \sum_{i=1}^{n} (-x_i) = -\triangle \sum_{i=1}^{n} x_i \wedge \triangle \sum_{i=1}^{n} (-x_i) =$

$= -\nabla \sum_{i=1}^{n} x_i)$,

(e) $\bigwedge_{(x_i)\in T_{21}^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} |\sum_{i=1}^{n} x_i - \square \sum_{i=1}^{n} x_i| \leqq$

$\leqq \varepsilon^* \, |\sum_{i=1}^{n} x_i|$,

wherein ε^* is defined as in theorem 1.

Proof: As we have seen in lemma 2, the sum of the x_i does not change and they are gradually ordered wrt. the relation $\prec$. Therefore the iteration stops after at most n-1 steps.

Case 1: If the algorithm stops because $n \leqq 1$, then (a) is trivially fulfilled.

Case 2: If the algorithm stops because t=o, then all x_i are ordered according to $\prec$ and the greatest summand x_n is the result s, unless the last l-1 digits of x_n are all zero. In this case, b^{-2l} has to be added to or subtracted from the mantissa of x_n, to take account of the influence of x_{n-1} on the result.

Case 3: If the algorithm stops because the summands $x_1,\dots,x_{n-1}$ have no significant influence on x_n, i.e. if they cannot change sign, exponent and first l+1 digits of x_n, then x_n is the result s.

The properties (b),(c),(d),(e) follow immediately from theorem 1. ■

Remarks: 1) In nearly all cases, the result is precise enough after the first pass; so no iteration occurs. Therefore the execution time is roughly proportional to n.

2) If the exponents of the $x_i \in T_{b,21}$ ($i=1(1)n$) are bounded by constants e1,e2, i.e. if

$\bigwedge_{i=1(1)n} \exp(x_i) \in \{e1, e1+1, \dots, e2\},$

then an exponent range $\{\overline{e1},\dots,\overline{e2}\}$, $\overline{e1} := e1-2\cdot l+1$, $\overline{e2} := e2 \cdot \lceil \log_b(n) \rceil$, is needed for intermediate results in x_i and for s.

This can be achieved by first decomposing the input data x_i into signed mantissas m_i and integer exponents e_i.

3) Let $x_i, y_i \in T_{b,1}$ ($i=1(1)n$) be floating-point numbers. By applying algorithm 1 on the products $x_i \cdot y_i \in T_{b,21}$, the scalar product $\square \sum_{i=1}^{n} x_i \cdot y_i$ can be computed for all roundings $\square \in \{\nabla,\triangle,\square_\mu(\mu=o(1)b)\}$.

As matrix products and linear mappings are componentwise defined as scalar products, we can compute the following functions:

$T^n \times T^n \ni (x,y) \longmapsto \square(x \cdot y) \in T$ (scalar product),

$T^{n\times n} \times T^{n\times n} \ni (A,B) \mapsto \square(A \cdot B) \in T^{n\times n}$ (matrix product),

$T^n \ni x \longmapsto \square(C \cdot x + c) \in T^n$ (linear mapping),

wherein $C \in T^{n\times n}$ is a fixed floating-point matrix and $c \in T^n$ a fixed floating-point vector.

Grüner [2] applied these functions on several matrix inversion algorithms and got error bounds that are better than those for single precision computation by a factor of about n. Furthermose these error bounds are valid in all cases, whereas for single precision computation (as well as for ordinary double precision computation) additional assumptions are needed.

4) Algorithm 1 was intended mainly for the computation of scalar products. Some storage space is wasted, if only the sum of n single-precision floating-point numbers $x_i \in T$ is needed. This could be avoided on the cost of more complicated termination criteria.

As matrix multiplication is the most interesting application of algorithm 1, we want to mention some properties that were proved by Kulisch and Bohlender [7] in abstract spaces. In the context of the present paper, these properties follow immediately from theorem 2.

Theorem 3: Let $T=T_{b,l}$ be a floating-point system and $T^{n\times n}$ the set of n×n-matrices with components in T. The floating-point matrix operations

$\boxed{*} : T^{n\times n} \times T^{n\times n} \Rightarrow T^{n\times n}$ are defined by

$$\bigwedge_{A,B\in T^{n\times n}} A \boxed{*} B := \square(A * B),\ * \in\{+,-,\cdot\}, \square\in\{\nabla,\triangle,\square_\mu\}.$$

Then the following properties hold for all $*\in\{+,-,\cdot\}$:

(a) $\bigwedge_{A,B\in T^{n\times n}} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} (A * B \in T^{n\times n} \Rightarrow A\boxed{*}B = A * B)$,

(b) $\bigwedge_{A,B,C,D\in T^{n\times n}} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} (A * B \leqq C * D \Rightarrow A\boxed{*}B \leqq C\boxed{*}D)$,

(c) $\bigwedge_{A,B\in T^{n\times n}} \bigwedge_{\square\in\{\square_\mu(\mu=o(1)b)\}} ((-A)\boxed{+}(-B) = -A\boxed{-}B \ \wedge$
$\wedge\ (-A)\boxed{\cdot}B = A\boxed{\cdot}(-B) = -A\boxed{\cdot}B)$,

$\bigwedge_{A,B\in T^{n\times n}} ((-A)\nabla\!\!\!\!\!+(-B) = -A\triangle\!\!\!\!\!-B \wedge (-A)\triangle\!\!\!\!\!+(-B) = -A\nabla\!\!\!\!\!-B\ \wedge$
$\wedge\ (-A)\nabla\!\!\!\!\cdot\,B = A\nabla\!\!\!\!\cdot\,(-B) = -A\triangle\!\!\!\!\cdot\,B\ \wedge$
$\wedge\ (-A)\triangle\!\!\!\!\cdot\,B = A\triangle\!\!\!\!\cdot\,(-B) = -A\nabla\!\!\!\!\cdot\,B)$,

(d) $\bigwedge_{A,B\in T^{n\times n}} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} |A * B - A\boxed{*}B| \leqq \varepsilon^*|A * B|$.

2. The product of n floating-point numbers

If $x_i \in T_{b,l}$ $(i=1(1)n)$ are floating-point numbers, then their product can be computed exactly in $T_{b,n\cdot l}$ and rounded afterwards. But this computation requires $n\cdot(n-1)/2$ multiplications of single-length mantissas. Therefore we will replace the exact result $\prod_{i=1}^{n} x_i$ by a double-length approximation $\widetilde{\prod_{i=1}^{n}} x_i \in T_{b,2l}$ and return to using $\prod_{i=1}^{n} x_i$ only if this double-length result turns out to be too inaccurate. The following lemma gives a criterion for the property

$$\bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} \square \prod_{i=1}^{n} x_i = \square \widetilde{\prod_{i=1}^{n}} x_i \quad . \tag{14}$$

Lemma 3: Let $T=T_{b,l}$ be a floating-point system, $2\leqq n\leqq b^{2l-2}$ and $x_i \in T$ $(i=1(1)n)$ floating-point numbers. With the downwardly directed rounding $\nabla_{2l}: \mathbb{R} \to T_{b,2l}$ from (2), an approximation $p_2 := \widetilde{\prod_{i=1}^{n}} x_i$ can be defined by

$$p_2 := \prod_{i=1}^{n} \operatorname{sgn}(x_i)\cdot \nabla_{2l}(\dots\nabla_{2l}(\nabla_{2l}(|x_1|\cdot|x_2|)\cdot |x_3|)\dots\cdot|x_n|)\ . \tag{15}$$

With $\operatorname{mant}(p_2)=m=o.m[1]\dots m[2l]$, (14) holds if

$$\begin{aligned} &o.m[1]\dots m[l+1] < m \ \wedge \\ &\wedge\ o.m[1]\dots m[l+1] + b^{-(l+1)} > m + 2(n-2)b^{-(2l-1)} \end{aligned} \tag{16}$$

Proof: For $x,y\in T_{b,2l}$, we get from (2) and theorem 1, (d):

$$\nabla_{2l}(|x|\cdot|y|) \leqq |x|\cdot|y| \leqq \nabla_{2l}(|x|\cdot|y|)/(1-b^{-(2l-1)}).$$

Considering that the innermost product in (15) can be executed exactly, multiple application of this inequation delivers

$$|p_2| \leqq |\prod_{i=1}^{n} x_i| \leqq |p_2|/(1-b^{-(2l-1)})^{n-2}$$

Using Bernoulli's inequation and the assumption $n\leqq b^{2l-2}$, we find like Wilkinson [11]:

$$1/(1-b^{-(2l-1)})^{n-2} \leqq 1 + \sum_{i=1}^{\infty}((n-2)b^{-(2l-1)})^i \leqq$$
$$\leqq 1 + 2(n-2)b^{-(2l-1)}.$$

Therefore, the following interval $I\in\mathbb{IR}$ contains the exact product:

$$|\prod_{i=1}^{n} x_i| \in I := [|p_2|, |p_2|\cdot(1+2(n-2)b^{-(2l-1)})] \tag{17}$$

With these preparations, the lemma can be proved as follows:

$(16) \Rightarrow (o.m[1]\dots m[l+1] < m \wedge o.m[1]\dots m[l+1] + b^{-(l+1)} > m\cdot(1+2(m-2)b^{-(2l-1)})) \Rightarrow I \cap T_{b,l+1} = \emptyset \underset{(17)}{\Rightarrow} (14)$. ■

As it is convenient to treat signs, exponents and mantissas of the floating-point numbers x_i separately, we introduce the following notations:

(a) $sp\in\{+1,-1\}, mp=o.mp[1]\dots mp[l+2], ep\in\mathbb{Z}$ are variables for the sign, mantissa and exponent of the product p resp.; the last digit mp[l+2] of mp may be a dual digit which indicates whether the result is truncated (mp[l+2] =1) or exact (mp[l+2]=o).

(b) $m_i = m_i[o].\ m_i[1]\dots m_i[l]$, $i=1(1)n$, are auxiliary variables with a carry digit $m_i[o]$; ma,mb,mc,md alike. $m=o.m[1]\dots m[2l]$ is a double length mantissa.

(c) $m:=m_j \times m_i$ denotes the exact double-length product of the mantissas m_j and m_i, provided that neither m_j nor m_i has a carry.

(d) (ma,mb):=m denotes the decomposition of the double-length mantissa m into two single-length mantissas ma:=o.m[1]...m[l],mb:=o.m[l+1]...m[2l].

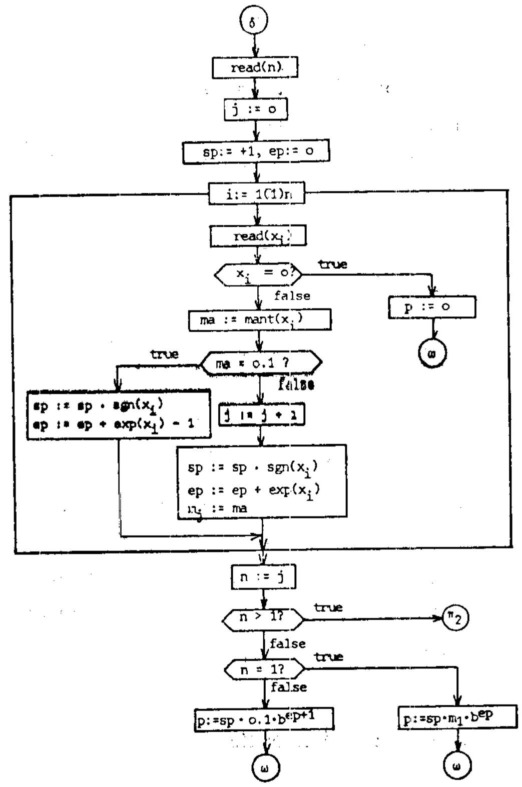
Then the following theorem holds:

Theorem 4: Let $T=T_{b,l}$ be a floating-point system, $n\leqq b^{2l-2}$, $x_i\in T$ $(i=1(1)n)$. Then algorithm 2 determines an approximation $p = \widetilde{\prod_{i=1}^{n}} x_i \in T_{b,l+2}$ of $\prod_{i=1}^{n} x_i$ with the property

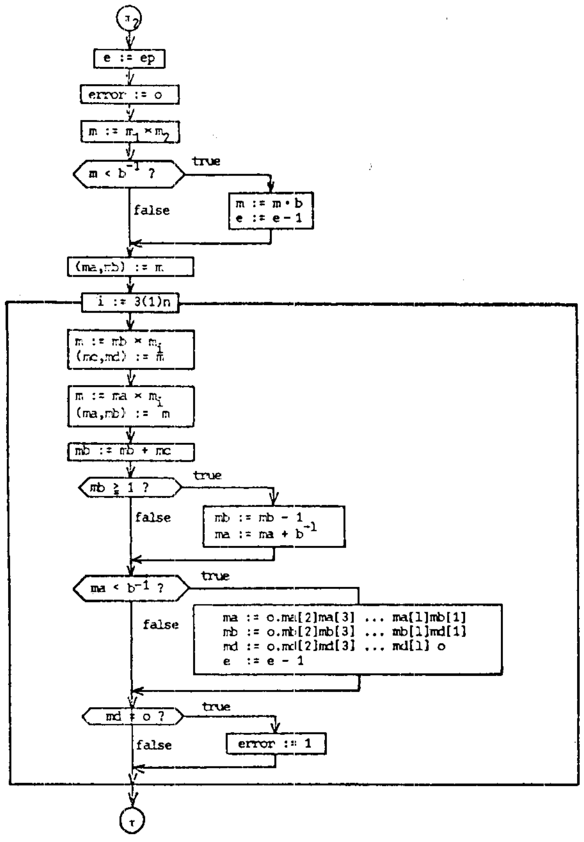
(a) $\bigwedge_{(x_i)\in T^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu(\mu=o(1)b)\}} \square\prod_{i=1}^{n} x_i = \square p.$

Furthermore the following properties hold:

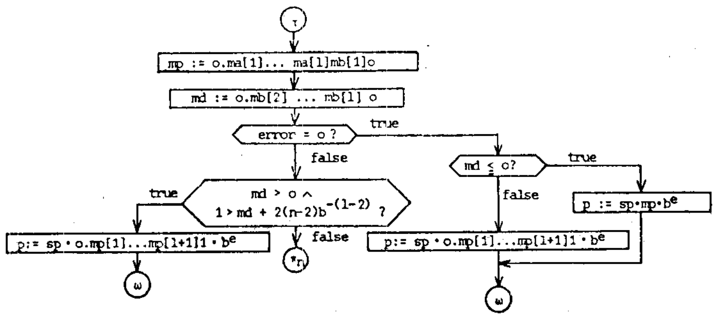
(b) $\bigwedge_{(x_i)\in T^n} \bigwedge_{\square\in\{\nabla,\triangle,\square_\mu\}} (\prod_{i=1}^{n} x_i \in T \Rightarrow \square\widetilde{\prod_{i=1}^{n}} x_i = \prod_{i=1}^{n} x_i)$,



2.1: Decomposition

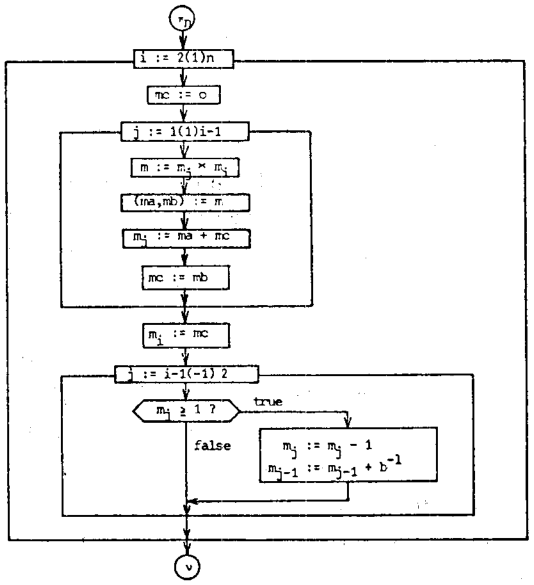


2.2: Double-length product

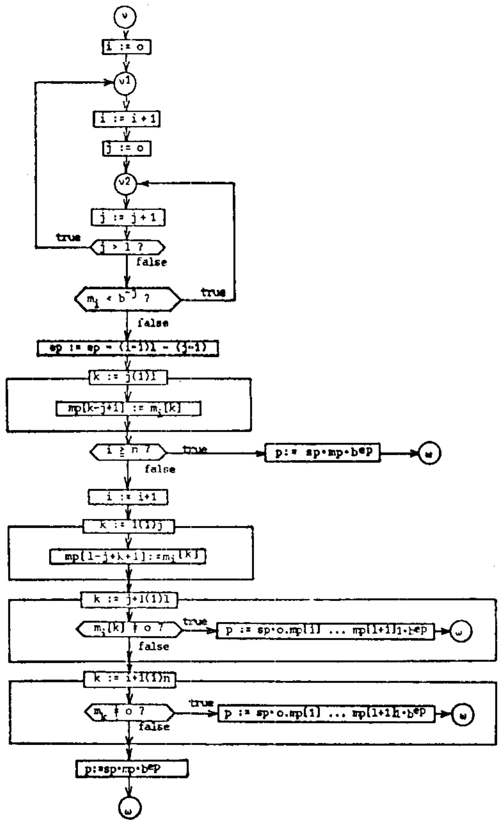


τ
mp := o.ma[1]... ma[l]mb[1]o
md := o.mb[2] ... mb[l] o
error = o ?
true
false
md ≤ o?
true
false
p := sp·mp·b^e
md > o ∧
1 > md + 2(n-2)b^{-(l-2)} ?
true
false
p:= sp · o.mp[1]...mp[l+1]1 · b^e
π_n
p:= sp · o.mp[1]...mp[l+1]1 · b^e
ω
ω

2.3: Test of the precision of the double length result



2.4: Exact product



2.5: Normalization

Algorithm 2: Product of n floating-point numbers

(c) $\bigwedge_{(x_i),(y_i)\in T^n} \bigwedge_{\Box\in\{\nabla,\Delta,\Box_\mu\}} \left(\prod_{i=1}^{n} x_i \leqq \prod_{i=1}^{n} y_i \Longrightarrow \Box\prod_{i=1}^{n} x_i \leqq \Box\prod_{i=1}^{n} y_i\right),$

(d) $\bigwedge_{(x_i)\in T^n} \bigwedge_{\Box\in\{\nabla,\Delta,\Box_\mu\}} \left|\prod_{i=1}^{n} x_i - \Box\prod_{i=1}^{n} x_i\right| \leqq \varepsilon^*\cdot\left|\prod_{i=1}^{n} x_i\right|$

Proof: (b),(c),(d) follow immediately from theorem 1, so we only have to prove (a).

Case 1: (a) is trivial if the algorithm stops because $n\leq 1$ in the decomposition δ .

Case 2: If "error = o" in the test τ , then no rounding error has occurred in π_2. Therefore mp can be determined in τ from ma and mb. Otherwise some mantissas md have been neglected in the course of π_2. Then condition (16) from the preceding lemma determines whether the double length result $p_2:=sp\cdot(ma+mb\cdot b^{-l})\cdot b^{e}$ is precise enough.

Case 3: If $md>o \wedge 1 > md+2(n-2)b^{-(l-2)}$, then condition (16) is satisfied; therefore the first l+1 digits of the mantissa mp are correct and mp[l+2]=1 indicates that some digits $\neq$o are cut off.

Case 4: If neither of the preceding cases applies, then generally the double length result p_2 can no longer be rounded correctly for all $\Box\in\{\nabla,\Delta,\Box_\mu\}$, and the product has to be computed exactly by the algorithm π_n.

In this algorithm the mantissas m_i of the floating-point numbers x_i are multiplied recursively and their products are stored as multiple length mantissas in those m_i which are no more used. So for all j=2(1)n we have

$$\prod_{i=1}^{n} x_i = sp\cdot[(m_1^{(j)}+m_2^{(j)}\cdot b^{-l}+m_3^{(j)}\cdot b^{-2l}+\ldots+m_j^{(j)}\cdot b^{-(j-1)l})$$
$$\cdot m_{j+1}\cdot m_{j+2}\cdot\ldots\cdot m_n]\cdot b^{ep}$$

Therein $m_{j+1},\ldots,m_n$ are the mantissas of the floating-point numbers $x_{j+1},\ldots,x_n$ and $m_1^{(j)},\ldots,m_j^{(j)}$ represent the multiple length product $\prod_{i=1}^{j} m_i$.

The double loop in the normalization algorithm ν can be run through at most n·l times, because the product is not zero. After this loop the j-th digit of m_i is the first digit which is not zero. Therefore the result can be determined from the following mantissas, and mp[l+2] again indicates whether there are digits $\neq$o truncated off. ■

Remarks: 1) In most cases, p_2 is precise enough for the computation of the product p. But no double-length result can be sufficient in all cases to get the correctly rounded result. This is shown in the floating-point system $T=T_{1o,2}$ by the following example:

$$o.2\in T,\ o.5\in T,\ o.2^{14}\cdot o.5^{14}=o.1\cdot 1o^{-13}\in T\ ,$$

but the intermediate result $o.2^{14}=o.16384\cdot 1o^{-9}$ is no double-length floating-point number, i.e. $o.2^{14}\notin T_{1o,4}$. Therefore the product $o.2^{14}\cdot o.5^{14}$ cannot be computed exactly by recursive double-length multiplication.

2) If $\exp(x_i)\in\{e1,\ldots,e2\}$ (i=1(1)n), then $\exp(\prod_{i=1}^{n} x_i)\in\{n\cdot e1-n+1,\ldots,n\cdot e2\}$.

3) The condition $n\leqq b^{2l-2}$ is no serious restriction, as the constant b^{2l-2} is very large for ordinary floating-point systems. It can be dropped if the product is computed exactly for $n>b^{2l-2}$.

4) In the course of the double-length product π_2, the mantissas md are lost. Therefore the algorithm π_n cannot use the results of π_2 and has to start anew. By storing the mantissas md, this could be avoided on the cost of storage space and complexity of π_n.

3. Roots of floating-point numbers

For a floating-point number $a\in T$, we will determine $\Box\sqrt[n]{a}$ for $\Box\in\{\nabla,\Delta,\Box_\mu\}$ using a modification of Newton's method. At first, we introduce some notations:

$\mathbb{IR}:=\{[a,b];\ a,b\in\mathbb{R},\ a\leqq b\}$ denotes the set of all closed real intervals and $\mathbb{I}T:=\{[a,b];\ a,b\in T,\ a\leqq b\}\subseteq\mathbb{IR}$ denotes the subset of all closed floating-point intervals. Then the monotone, outwardly directed rounding $\Diamond:\mathbb{IR}\to\mathbb{I}T$ can be defined by

$$\bigwedge_{X=[x_1,x_2]\in\mathbb{IR}} \Diamond X = \Diamond[x_1,x_2] := [\nabla x_1,\Delta x_2] \tag{18}$$

and arithmetic operations $\circledast:\mathbb{I}T\times\mathbb{I}T\to\mathbb{I}T$, $*\in\{+,-,\times,/\}$, by

$$\bigwedge_{X,Y\in\mathbb{I}T} X\circledast Y:=\Diamond(X*Y),\ *\in\{+,-,\times,/\}. \tag{19}$$

In the case of the division, $o\notin Y$ is assumed.

Then the following theorem delivers - under appropriate conditions - the smallest floating-point interval containing the zero of a function.

Theorem 5: Let $T=T_{b,l}$ be a floating-point system, $x_1^{(o)},x_2^{(o)}\in T$, $o<x_1^{(o)}<x_2^{(o)}$, two positive floating-point numbers and $f:X^{(o)}:=[x_1^{(o)},x_2^{(o)}]\to\mathbb{R}$ a real-valued function with the following properties:

(a) $\bigvee_{\xi\in X^{(o)}} f(\xi) = o$,

(b) $\bigvee_{m_1,m_2\in T} \bigwedge_{x\in X^{(o)}\setminus\{\xi\}} o<m_1\leqq\frac{f(x)}{x-\xi}\leqq m_2<\infty$,
$M:=[m_1,m_2]$,

(c) there is a function $F: X^{(o)} \cap T \to \mathbb{I}T$, with

(c1) $\bigwedge_{x \in X^{(o)} \cap T} f(x) \in F(x)$,

(c2) $\bigwedge_{x \in X^{(o)} \cap T} (F(x) \geqq o \vee F(x) \leqq o)$,

(c3) $\xi \in T \Rightarrow F(\xi) = [o,o]$ (where ξ is the zero from (a)).

Starting with $X^{(o)}$, let a sequence $(X^{(k)})_{k=o,1,2,\ldots}$ of intervals $X^{(k)} = [x_1^{(k)}, x_2^{(k)}] \in \mathbb{I}T$ be generated by

(d) $X^{(k+1)} := X^{(k)} \cap ([m(X^{(k)}), m(X^{(k)})] \ominus F(m(X^{(k)})) \oslash M)$ $k=o,1,2,\ldots,$

wherein $m(X^{(k)})$ fulfils the condition

(e) $m(X^{(k)}) \in X^{(k)} \cap T$ and

(e1) $m(X^{(k)}) \neq x_1^{(k)}, x_2^{(k)}$, if $X^{(k)} \cap T$ has at least 3 elements,

(e2) $m(X^{(k)}) = \begin{cases} x_1^{(k)}, & \text{if } k \text{ is even} \\ x_2^{(k)}, & \text{if } k \text{ is odd} \end{cases}$, else.

For this sequence $(X^{(k)})_{k=o,1,\ldots}$ the following properties hold:

(f) $\bigwedge_{k \in \mathbb{N}} \Diamond[\xi,\xi] \subseteq X^{(k)}$,

(g) $\bigwedge_{k \in \mathbb{N}} X^{(k)} \cap T \neq \emptyset$, i.e. condition (e) can always be satisfied,

(h) $\bigvee_{k_o \in \mathbb{N}} X^{(o)} \supset X^{(1)} \supset \ldots \supset X^{(k_o)} = X^{(k_o+1)} \supseteq X^{(k_o+2)} = X^{(k_o+3)} = \ldots,$

(i) $X^{(k_o+2)} = \Diamond[\xi,\xi] = [\nabla\xi, \triangle\xi]$.

<u>Proof:</u> Properties (f) and (g) have to be proved simultaneously by induction. As the properties trivially hold for k=o, we can assume that they hold for any given $k \in \mathbb{N}$. Because of property (g) for k, $m(X^{(k)})$ can be chosen according to condition (e), therefore $X^{(k+1)}$ exists.

The induction step can be executed similarly as in Alefeld/Herzberger [1]:

From the identity

$$\xi = m(X^{(k)}) - \frac{f(m(X^{(k)}))}{\dfrac{f(m(X^{(k)}))}{m(X^{(k)})-\xi}} \qquad (\xi \neq m(X^{(k)}))$$

and property (b), we get with the inclusion property of interval arithmetic

$$[\xi,\xi] \subseteq [m(X^{(k)}), m(X^{(k)})] - [f(m(X^{(k)})), f(m(X^{(k)}))] \,/\, M.$$

Note that $o \notin M$ and that this property holds also for $m(X^{(k)}) = \xi$. With (c1) and the definition of the rounding $\Diamond$ we get

$$\begin{aligned} [\xi,\xi] &\subseteq [m(X^{(k)}), m(X^{(k)})] - F(m(X^{(k)})) \,/\, M \\ &\subseteq [m(X^{(k)}), m(X^{(k)})] - F(m(X^{(k)})) \oslash M \\ \Rightarrow \Diamond[\xi,\xi] &\subseteq \Diamond([m(X^{(k)}), m(X^{(k)})] - F(m(X^{(k)})) \oslash M) \\ &= [m(X^{(k)}), m(X^{(k)})] \ominus F(m(X^{(k)})) \oslash M . \\ \Rightarrow \Diamond[\xi,\xi] &\subseteq X^{(k+1)}. \end{aligned}$$

So we have proved (f) for k+1. Property (g) is a trivial consequence of (f).

For the proof of properties (h) and (i), we first define functions $f_1, f_2: X^{(o)} \cap T \to T$ by $F(x) =: [f_1(x), f_2(x)]$ for all $x \in X^{(o)} \cap T$. Then method (d) can be expressed componentwise:

$$x_1^{(k+1)} = \begin{cases} \max\{x_1^{(k)}, m(X^{(k)}) \nabla f_2(m(X^{(k)})) \triangle m_1\}, \\ \text{if } f_2(m(X^{(k)})) > o \\ m(X^{(k)}) \nabla f_2(m(X^{(k)})) \triangle m_2 , \\ \text{if } f_2(m(X^{(k)})) \leqq o \end{cases}$$

$$x_2^{(k+1)} = \begin{cases} m(X^{(k)}) \triangle f_1(m(X^{(k)})) \nabla m_2, \\ \text{if } f_1(m(X^{(k)})) \geqq o \\ \min\{x_2^{(k)}, m(X^{(k)}) \triangle f_1(m(X^{(k)})) \nabla m_1\}, \\ \text{if } f_1(m(X^{(k)})) < o. \end{cases} \tag{20}$$

Now we distinguish the two cases of (e):

<u>Case 1:</u> $X^{(k)} \cap T$ has at least 3 elements.
Then $x_1^{(k)} < m(X^{(k)}) < x_2^{(k)}$.
Because of property (c2), we get:

(α) $F(m(X^{(k)})) \geqq o \Rightarrow f_1(m(X^{(k)})) \geqq o \Rightarrow x_2^{(k+1)} = m(X^{(k)}) \triangle f_1(m(X^{(k)})) \nabla m_2 \leqq m(X^{(k)}) < x_2^{(k)}$, or

(β) $F(m(X^{(k)})) \leqq o \Rightarrow f_2(m(X^{(k)})) \leqq o \Rightarrow x_1^{(k+1)} = m(X^{(k)}) \nabla f_2(m(X^{(k)})) \triangle m_2 \geqq m(X^{(k)}) > x_1^{(k)}$.

So in each case

$x_2^{(k+1)} - x_1^{(k+1)} < x_2^{(k)} - x_1^{(k)}$

$\Rightarrow X^{(k+1)} \subset X^{(k)}$.

As $X^{(o)} \cap T$ is a finite set, an index k_o' must exist, so that $X^{(k_o')} \cap T$ has at most 2 elements.

<u>Case 2:</u> $X^{(k)} \cap T$ has at most 2 elements.
Then one of the following two cases occurs:

(α) $X^{(k)} \cap T$ has exactly one element. Then

$$\begin{aligned} &X^{(k+1)} = X^{(k)} = \Diamond[\xi,\xi] = [\xi,\xi] \\ \Rightarrow &X^{(k+1)} = X^{(k+2)} = \ldots = X^{(k)} = \Diamond[\xi,\xi] . \end{aligned}$$

(β) $X^{(k)} \cap T$ has exactly two elements. Then one of the following three cases occurs:

$$(\beta 1)\ \xi = x_1^{(k)} \Longleftrightarrow f(x_1^{(k)}) = o \underset{(c)}{\Longleftrightarrow} F(x_1^{(k)}) = [o,o],$$

$$(\beta 2)\ \xi = x_2^{(k)} \Longleftrightarrow f(x_2^{(k)}) = o \underset{(c)}{\Longleftrightarrow} F(x_2^{(k)}) = [o,o],$$

$$(\beta 3)\ x_1^{(k)} < \xi < x_2^{(k)} \Longleftrightarrow f(x_1^{(k)}) < o < f(x_2^{(k)}) \underset{(c)}{\Longleftrightarrow}$$

$$\Longleftrightarrow F(x_1^{(k)}) < [o,o] < F(x_2^{(k)}).^{*)}$$

In case $(\beta 3)$ we have

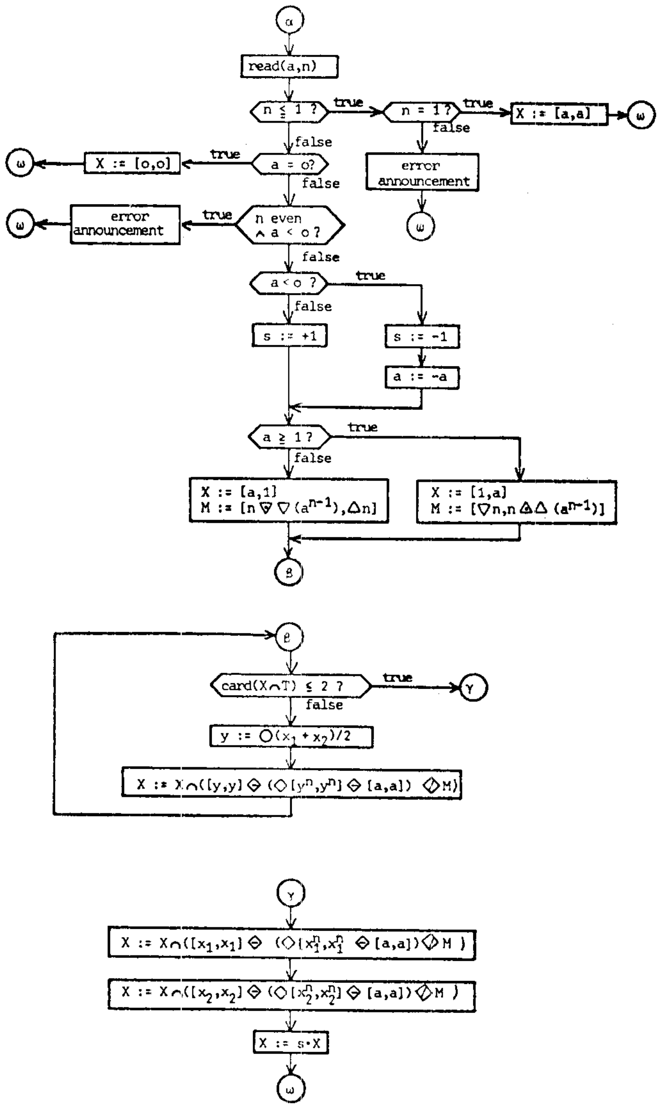
$$X^{(k)} = X^{(k+1)} = \dots = \Diamond[\xi,\xi]\ ,$$

in the two other cases, (e2) and (20) imply that either

$$X^{(k)} \supset X^{(k+1)} = X^{(k+2)} = \dots = [\xi,\xi] = \Diamond[\xi,\xi],$$

or

$$X^{(k)} = X^{(k+1)} \supset X^{(k+2)} = \dots = [\xi,\xi] = \Diamond[\xi,\xi].\ \blacksquare$$



Algorithm 3: Root of a floating-point number

Remarks: 1) Method (d) delivers $\Diamond[\xi,\xi] = [\bigtriangledown\xi,\bigtriangleup\xi]$, even if F is a bad approximation of f (provided that condition (c) is satisfied) and even if $m(X^{(k)})$ is a bad choice (provided that condition (e) is satisfied). By an inappropriate choice of F and $m(X^{(k)})$, method (d) may degenerate into a trial-and-error method.

2) A similar method was given by Herzberger [3]. But without the assumptions (c2,3) and (e), property (i) could not be proved. In fact (c2) is the crucial additional assumption compared with Herzberger's version of Newton's method. Without (c2) we cannot know, whether a given $x \in T$ is left or right of the zero. Therefore we cannot expect that (i) is valid and in general we could not even find $[\bigtriangledown\xi,\bigtriangleup\xi]$ by trying out all floating-point numbers in $X^{(o)}$.

3) The assumptions (a) and (b) imply that ξ is the only zero of the function f in the interval $X^{(o)}$ and that f is Lipschitz-continuous at the place ξ . Apart from this, f need neither be continuous nor monotone.

In the following theorem, the results of theorem 5 are applied on the computation of arbitrary roots of floating-point numbers:

Theorem 6: Let $T_{b,l}$ be a floating-point system, $a \in T$ a floating-point number and $n \leqq b^{2l-2}$ a positive integer. Then algorithm 3 terminates after a finite number of iterations and if $a \geqq o$ or n is odd, then it delivers a floating-point interval $X \in IT$ with the property

$$(a)\ \bigwedge_{a \in T} X = \Diamond[\sqrt[n]{a},\sqrt[n]{a}] = [\bigtriangledown\sqrt[n]{a},\bigtriangleup\sqrt[n]{a}]\ .$$

Provided that $\sqrt[n]{a} \in \mathbb{R}$ exists, the following properties hold:

$$(b)\ \bigwedge_{a \in T}\ \bigwedge_{\square \in \{\bigtriangledown,\bigtriangleup,\square_\mu\}} (\sqrt[n]{a} \in T \Rightarrow \square\sqrt[n]{a} = \sqrt[n]{a}),$$

$$(c)\ \bigwedge_{a,b \in T}\ \bigwedge_{\square \in \{\bigtriangledown,\bigtriangleup,\square_\mu\}} (a \leqq b \Rightarrow \square\sqrt[n]{a} \leqq \square\sqrt[n]{b}),$$

$$(d)\ \bigwedge_{a \in T}\ \bigwedge_{\square \in \{\bigtriangledown,\bigtriangleup,\square_\mu\}} |\sqrt[n]{a} - \square\sqrt[n]{a}| \leqq \varepsilon^* \cdot |\sqrt[n]{a}|\ ,$$

and if n is odd:

$$(e)\ \bigwedge_{a \in T}\ \bigwedge_{\square \in \{\square_\mu ; \mu = o(1)b\}} \square\sqrt[n]{-a} = -\square\sqrt[n]{a}\ ,$$

$$\bigwedge_{a \in T} (\bigtriangledown\sqrt[n]{-a} = -\bigtriangleup\sqrt[n]{a} \wedge \bigtriangleup\sqrt[n]{-a} = -\bigtriangledown\sqrt[n]{a}).$$

Proof: (a) is trivial for a=o. For a> o, theorem 5 is used with

$$X^{(o)} := \begin{cases} [1,a]\ , & \text{if } a \geqq 1 \\ [a,1]\ , & \text{if } a < 1 \end{cases}\ ,$$

$$M := \begin{cases} [\bigtriangledown n, n \,\triangle\!\!\!\!\cdot\, \bigtriangleup(a^{n-1})], & \text{if } a \geqq 1 \\ [n \,\bigtriangledown\!\!\!\!\cdot\, \bigtriangledown(a^{n-1}),\bigtriangleup n], & \text{if } a < 1 \end{cases}\ ,$$

*) i.e. $F(x_1^{(k)}) \leqq [o,o] \leqq F(x_2^{(k)})$ and $F(x_1^{(k)}) \neq [o,o],\ F(x_2^{(k)}) \neq [o,o]$

$f(x) := x^n - a,$

$F(x) := \Diamond[x^n, x^n] \ominus [a,a]$,

wherein $\Diamond[x^n, x^n] = [\nabla x^n, \Delta x^n]$ is computed with the product from section 2. Then assumptions (a),(b),(c) of theorem 5 are satisfied, therefore the iteration stops after a finite number of steps and delivers $\Diamond[\xi,\xi] = \Diamond[\sqrt[n]{a}, \sqrt[n]{a}] = [\nabla\sqrt[n]{a}, \Delta\sqrt[n]{a}]$. For $a < o$, $\sqrt[n]{a}$ does only exist, if n is odd. This case can be easily reduced to the case $a > o$.
(b),(c),(d),(e) follow immediately from theorem 1. ■

Remarks: 1) Algorithm 3 delivers the smallest interval containing $\sqrt[n]{a}$, whereas in properties (b),(c),(d), (e) the rounded result $\Box\sqrt[n]{a}$ for $\Box \in \{\nabla, \Delta, \Box_\mu\}$ is needed. As can be seen from (2),(3),(4),(5),(18), $\Box\sqrt[n]{a}$ for $\Box \in \{\nabla, \Delta, \Box_o, \Box_b\}$ can be determined from $\Diamond[\sqrt[n]{a}, \sqrt[n]{a}]$. For $\Box_\mu \sqrt[n]{a}$ ($\mu=1(1)b-1$), algorithm 3 has to be executed in $T=T_{b,l+1}$.

2) If $\exp(a) \in \{e1,\dots,e2\}$, then an exponent range $\{n \cdot e1-n+1,\dots, n \cdot e2\}$ has to be provided for intermediate results.

Literature:

1. Alefeld, G., Herzberger, J.: Einführung in die Intervallrechnung. Bibliographisches Institut, Mannheim (1974)

2. Grüner, K.: Fehlerschranken für lineare Gleichungssysteme. Talk at the MRI Oberwolfach (1975)

3. Herzberger, J.: Über die Nullstellenbestimmung bei näherungsweise berechneten Funktionen. Computing 10, 23-31 (1972)

4. Kahan, W.: Further remarks on reducing truncation errors. Comm. ACM 8, 1 (Jan. 1965), 40

5. Kulisch, U.: An axiomatic approach to rounded computations. Numer. Math. 18, 1-17 (1971)

6. Kulisch, U.: Über die Arithmetik von Rechenanlagen. Jahrbuch Überblicke der Mathematik, Bibliographisches Institut, Mannheim (1975)

7. Kulisch, U., Bohlender, G.: Formalization and implementation of floating-point matrix operations. To appear in Computing

8. Linz, P.: Accurate floating-point summation. Comm. ACM 13, 6 (June 1970), 361-362

9. Malcolm, M.A.: On accurate floating-point summation. Comm. ACM 14, 11 (Nov. 1971), 731-736

10. Pichat, M.: Correction d'une somme en arithmetique a virgule flottante. Numer. Math. 19, 400-406 (1972)

11. Wilkinson, J.H.: Rundungsfehler. Springer, Berlin-Heidelberg - New York (1969)

12. Yohe, J.M.: Interval bounds for square roots and cube roots. Computing 11, 51-57 (1973)

13. Yohe, J.M.: Roundings in floating-point arithmetic. IEEE Trans. on Comp., Vol. C 12, No. 6 (June 1973), 577-586