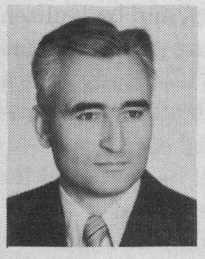


Ulrich Kulisch studied mathematics and physics at the Technical University, Munich, Germany, and the University of Munich, Munich, Germany, from 1953 to 1958. He received the Dr. rer. nat. degree in mathematics in 1961 and the Habilitation degree in mathematics in 1963, both from the Technical University of Munich.

Since 1963 he has taught mathematics at the Technical University of Munich, the University of Munich, and the University of Karlsruhe, Karlsruhe, Germany. Since 1967 he has been Full Professor of Mathematics and Director of the Institute of Applied Mathematics at the University of Karlsruhe. From 1966 to 1970 he was also Director of the Computer Center of this University. During the years 1969 and 1970 he was on academic leave with the Mathematics Research Center, The University of Wisconsin, Madison and 1972 to 1973 at the IBM T. J. Watson Research Center, Yorktown Heights, NY. He has published about 30 research articles in mathematics and computer sciences and two books, the first one in 1969 together with J. Heinhold on *Analog and Hybrid Computations* and the second one in 1976 about *Fundamentals of Numerical Computations—Mathematical Foundation of Computer Arithmetic*. Since 1968 he has been editor of the book series "Reihe Informatik" and since 1974 also of the series "Jahrbuch Überlicke Mathematik" by Bibliographisches Institut, Mannheim, West Germany.

Floating-Point Computation of Functions with Maximum Accuracy

GERD BOHLENDER

***Abstract*—Algorithms are given that compute multiple sums and products and arbitrary 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.**

***Index Terms*—Accuracy, errors, floating-point computations, multiple-length mantissas, roots of floating-point numbers, rounding.**

I. Introduction

OUR AIM is to approximate functions[1] $f: \mathbf{R}^n \to \mathbf{R}^p$ on a floating-point system T. For $b,l \in \mathbf{N}$, $b \geqq 2$, $l \geqq 1$, the floating-point system $T_{b,l}$ with base b and l-digit mantissa is defined by

$$T_{b,l} := \{0\} \cup \{x = *m \cdot b^e;\ * \in \{+,-\},\ m = 0 \cdot m[1] \cdots m[l],\ m[i] \in \{0,1,\cdots,b-1\},\ m[1] \neq 0,\ e \in \mathbf{Z}\}. \quad (1)$$

x is then called a floating-point number with sign $* = \operatorname{sgn}(x)$, mantissa $m = \operatorname{mant}(x)$, 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 that 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 $\square: \mathbf{R}^p \to T^p$ denotes a rounding.[2] We will restrict ourselves here to the roundings ∇, Δ and $\square_\mu$ ($\mu = 0(1)b$). For $p = 1$ these roundings are defined as follows:

$$\bigwedge_{x \in \mathbf{R}} \nabla x := \max\{y \in T; y \leqq x\} \quad (2)$$

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

$$\bigwedge_{x \geqq 0} \square_b x := \nabla x \wedge \bigwedge_{x < 0} \square_b x := \Delta x \quad (4)$$

$$\bigwedge_{x \geqq 0} \square_0 x := \Delta x \wedge \bigwedge_{x < 0} \square_0 x := \nabla x \quad (5)$$

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

$$\bigwedge_{x \geqq 0} \square_\mu x: \begin{cases} \nabla x & \text{for } x \in [\nabla x, S_\mu(x)) \\ \Delta x & \text{for } x \in [S_\mu(x), \Delta x] \end{cases}$$

$$\bigwedge_{x < 0} \square_\mu x := -\square_\mu(-x), \quad (6)$$

Manuscript received January 20, 1976; revised October 15, 1976.

The author is with the Institute of Applied Mathematics, University of Karlsruhe, Karlsruhe, Germany.

[1] **N, Z,** and **R** denote the sets of nonnegative integers, integers and reals, respectively. For any given set S, S^p denotes the set of p-tuples with components out of S. $\{x_i; P(x)\}$ denotes the set of all elements x with property $P(x)$.

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

wherein the function S_μ: $\mathbf{R} \to \mathbf{R}$ is defined by $S_\mu(x) := \nabla x + (\Delta x - \nabla x) \cdot \mu/b$. ∇, Δ, $\square_b$, $\square_0$, map a given real number x on the closest floating-point number which is equal to x or less, greater, absolutely less, absolutely greater, respectively, than x. Therefore, ∇ and Δ are called downwardly and upwardly directed rounding, respectively, $\square_b$ is called rounding towards zero, and $\square_0$ 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 $\square_\mu$ ($\mu = 0(1)b$) 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 $\square \in \{\nabla, \Delta, \square_\mu (\mu = 0(1)b)\}$. 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: $\mathbf{R}^n \to \mathbf{R}^p$ and a floating-point system $T = T_{b,l}$ the following properties hold:

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

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

$$\text{c)} \bigwedge_{x \in \mathbf{R}^n} \bigwedge_{\square \in \{\square_\mu; \mu = 0(1)b\}} (f(-x) = -f(x) \Rightarrow \square f(-x) = -\square f(x))$$

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

$$\text{d)} \bigwedge_{x \in \mathbf{R}^n} \bigwedge_{\square \in \{\nabla, \Delta, \square_\mu\}} |f(x) - \square f(x)| \leqq \epsilon^* \cdot |f(x)|$$

wherein absolute values are defined componentwise and

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

$$\text{e)} \bigwedge_{g: \mathbf{R}^n \to \mathbf{R}^p} \bigwedge_{x \in \mathbf{R}^n} \bigwedge_{\square \in \{\nabla, \Delta, \square_\mu\}} (f(x) \leqq g(x) \Rightarrow \square f(x) \leqq \square g(x)).$$

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

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 $\square \in \{\nabla, \Delta, \square_\mu (\mu = 0(1)b)\}$ 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 $\Sigma_{i=1}^n x_i$, $\Pi_{i=1}^n x_i$, $\sqrt[n]{x}$, , 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 the 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].

II. The Sum of n Floating-Point Numbers

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

$$\bigwedge_{(x_i) \in T^n} \bigwedge_{\square \in \{\nabla, \Delta, \square_\mu (\mu = 0(1)b)\}} \square \sum_{i=1}^{n} x_i = \square \widetilde{\sum_{i=1}^{n}} x_i. \tag{7}$$

We will give an algorithm that computes such an approximation; algorithms for the roundings ∇, Δ, $\square_\mu$ ($\mu = 0(1)b$) can be found in Kulisch [6].

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

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

$$\bigwedge_{x,y \in T_b^*} (x \prec y :\Leftrightarrow x = 0 \vee y = 0 \vee (\exp(y) - \exp(x) \geqq 1 \wedge y \in T_{\exp(y) - \exp(x)})) \tag{8}$$

where $x \prec y$ means that $y = 0$, 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 $\bigcirc$: $\mathbf{R} \to T$ the rounding to the closest floating-point number. Then for all $x, y \in T$, the following properties hold:

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

$$\text{b)} \; r \prec s$$

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 $x = 0$ or $y = 0$, then $r = 0$. So a), b), c), d) are fulfilled. Otherwise, we define $d := \exp(x) - \exp(y)$ and assume without loss of generality that $d \geqq 0$.

Case 1:

$$d > l \Rightarrow s = x \Rightarrow r = y \Rightarrow \text{a), b), c), d).}$$

Case 2:

$$d \leqq l \wedge x + y = 0 \Rightarrow s = r = 0 \Rightarrow \text{a), b), c), d).}$$

Case 3:

$$d \leqq l \wedge x + y \neq 0 \Rightarrow x + y \in T_{b,2l}\backslash\{0\} \Rightarrow x + y = *0.m[1]\cdots m[2l]\cdot b^e,$$

with $* \in \{+,-\}$, $m[i] \in \{0,1,\cdots,b-1\}$, $m[1] \neq 0$, $e \in \mathbf{Z}$. Then s and r fulfill one of the following two properties:

$$\alpha)\ s = *0.m[1]\cdots m[l]\cdot b^e \qquad r = *0.m[l+1]\cdots m[2l]\cdot b^{e-l},$$

$$\beta)\ s = *(0.m[1]\cdots m[l] + b^{-l})\cdot b^e \qquad r = *(0.m[l+1]\cdots m[2l] - 1)\cdot b^{e-l}.$$

Therein r and s can be denormalized and r can 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

$$r = 0 \vee \exp(r) \leqq \min(\exp(x),\exp(y)). \qquad \blacksquare$$

Writing shortly $(s,r) := x + y$ for $s := \circ(x + y)$, $r := (x + y) - s$, 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 $x^{(0)} \in T^n$, a sequence $(x^{(k)})_{k=0,1,2,\cdots}$, $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}) \in T^n$ is recursively defined by

$$s_1 := x_1^{(k)}$$

$$(s_{p+1}, x_p^{(k+1)}) := s_p + x_{p+1}^{(k)}, \qquad p = 1(1)n-1, \tag{9}$$

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

wherein $s_p \in T$ are auxiliary variables.

Then the sequence $(x^{(k)})_{k=0,1,2,\cdots}$ has the following properties:

a) $\bigwedge_{k\in\mathbf{N}} \sum_{i=1}^{n} x_i^{(k)} = \sum_{i=1}^{n} x_i^{(0)}$

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

Proof: a) is an immediate consequence of Lemma 1-a). b) is proved by induction over k. It is trivial for $k = 0$, so let us assume that it is valid for any $k \in \mathbf{N}$.

Case 1: 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:

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

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

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

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

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

$\alpha)$ (11),(12) $\Rightarrow$

$$\bigwedge_{n-k-1\leqq i\leqq p-1} \left(x_i^{(k+1)} \prec s_p \wedge x_i^{(k+1)} \prec x_{p+1}^{(k)}\right) \underset{l1.c)}{\Longrightarrow} \bigvee_{n-k-1\leqq i\leqq p-1} \left(x_i^{(k+1)} \prec s_{p+1} \wedge x_i^{(k+1)} \prec x_p^{(k+1)}\right) \underset{l1.b),d),(10)}{\Longrightarrow} \left(\bigwedge_{n-k-1\leqq l\leqq p} x_l^{(k+1)} \prec s_{p+1} \wedge \bigwedge_{n-k-1\leqq i<j\leqq p} x_i^{(k+1)} \prec x_j^{(k+1)}\right)$$

$$\Leftrightarrow \text{(10),(11) for } p + 1.$$

$\beta)$ (13) $\Rightarrow \bigwedge_{p+2\leqq j\leqq n} x_{p+1}^{(k)} \prec x_j^{(k)} \underset{l1.(d)}{\Longrightarrow} \bigwedge_{p+2\leqq j\leqq n} x_p^{(k+1)} \prec x_j^{(k)}$

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

Therefore, (10)–(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.

Case 2: If $k \geqq n - 1$, then $x_i^{(k)} \prec x_j^{(k)}$ whenever $2 \leqq i < j \leqq n$. So the same assumptions hold as in the case $k = n - 2$, and the induction step can be done as in Case 1.

Thus b) is proved for all $k \in \mathbf{N}$. $\blacksquare$

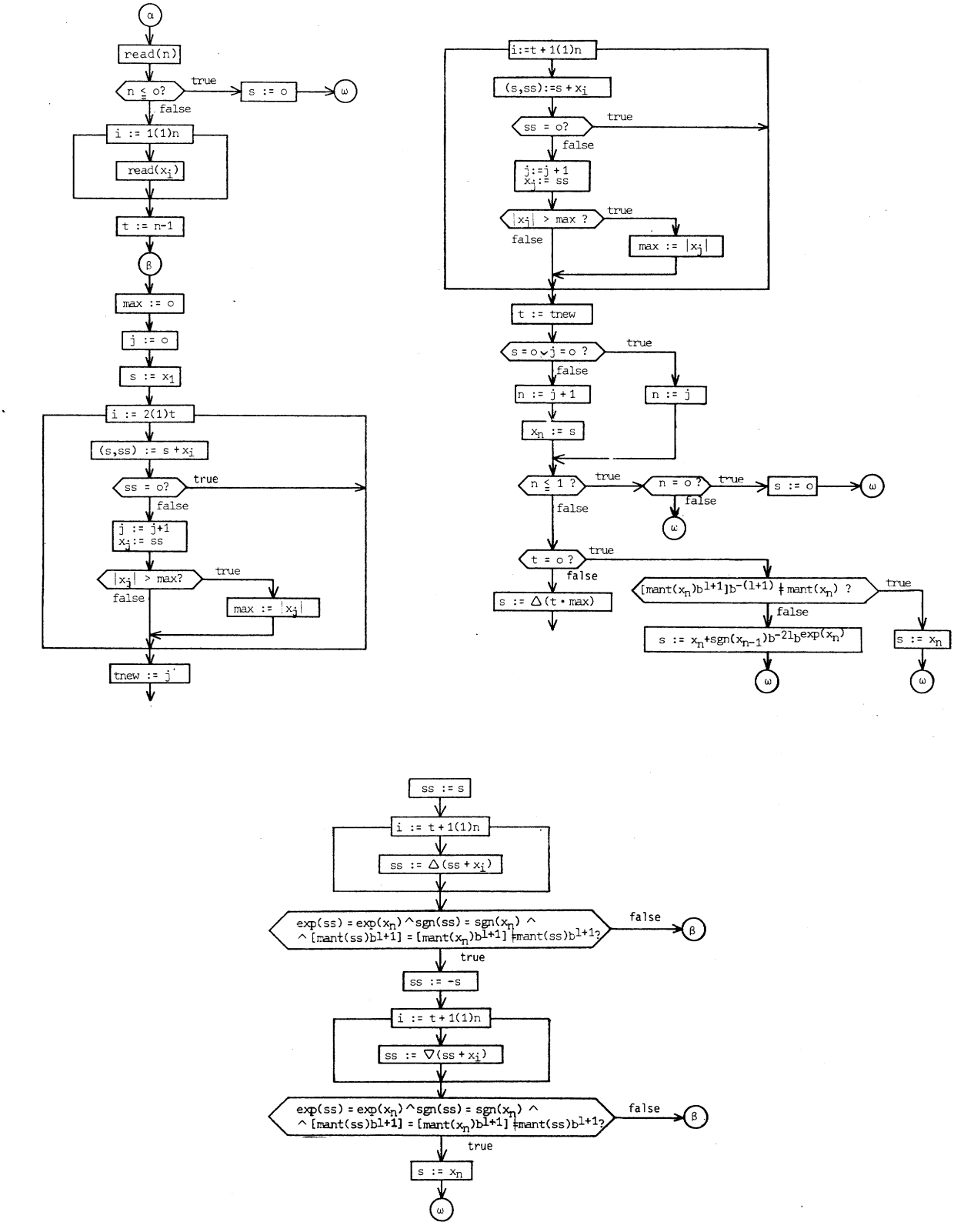
In Algorithm 1, the method from Lemma 2 is applied in $T_{b,2l}$ on the computation of the sum of n double-length floating-point numbers. Some modifications are made: the index t takes the role of $n - k$ in property b) of the lemma; zeros are eliminated; the iteration is stopped, if the result can be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

Theorem 2: Let $T_l = T_{b,l}$, $l \geqq 3$, be a floating-point system, $\square: \mathbf{R} \to T_{b,l}$, and $x_i \in T_{2l}$ $(i = (1)n)$ n double-length floating-point numbers. Then algorithm 1 determines an approximation $s \in T_{2l}$ of $\Sigma_{i=1}^n x_i$ with the property

a) $\bigwedge_{(x_i)\in T_{2l}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu(\mu=0(1)b)\}} \square \sum_{i=1}^{n} x_i = \square s.$

Furthermore, the following properties hold:

b) $\bigwedge_{(x_i)\in T_{2l}^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} \left(\sum_{i=1}^{n} x_i \in T_l \Rightarrow \square \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i\right)$



Algorithm 1. Sum of n floating-point numbers.

c) $$\bigwedge_{(x_i),(y_i)\in T_{2l}^n}\ \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}}\left(\sum_{i=1}^n x_i \leqq \sum_{i=1}^n y_i \Rightarrow \square\sum_{i=1}^n x_i \leqq \square\sum_{i=1}^n y_i\right)$$

d) $$\bigwedge_{(x_i)\in T_{2l}^n}\ \bigwedge_{\square\in\{\square_\mu(\mu=0(1)b)\}} \square\sum_{i=1}^n(-x_i) = -\square\sum_{i=1}^n x_i$$

$$\bigwedge_{(x_i)\in T_{2l}^n}\left(\nabla\sum_{i=1}^n(-x_i) = -\Delta\sum_{i=1}^n x_i \wedge \Delta\sum_{i=1}^n(-x_i) = -\nabla\sum_{i=1}^n x_i\right)$$

e) $$\bigwedge_{(x_i)\in T_{2l}^n}\ \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}}\left|\sum_{i=1}^n x_i - \square\sum_{i=1}^n x_i\right| \leqq \epsilon^*\left|\sum_{i=1}^n x_i\right|$$

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 $t = 0$, 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 $x_1,\cdots,x_{n-1}$ 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)–e) follow immedately from Theorem 1. ■

Remarks: 1) The approximation s is computed with double length and rounded to the single-length result $\square s$ afterwards. Therefore, it can be expected to be precise enough after the first pass. When catastrophic cancellation occurs, additional passes may be necessary to obtain the desired accuracy.

This result was confirmed by simulations performed on a UNIVAC 1108. We computed sums of up to 1000 summands and found that our algorithm was nearly constantly 2.5 times slower than a single precision algol for-statement.

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

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

then an exponent range $\{\overline{e1},\cdots,\overline{e2}\}$, $e1 := \overline{e1} - 2\cdot l + 1$, $\overline{e2} := e2\cdot\lceil\log_b(n)\rceil$, 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 $x_i, y_i \in T_{b,l}$ $(i = 1(1)n)$ be floating-point numbers. By applying Algorithm 1 on the products $x_i\cdot y_i \in T_{b,2l}$, the scalar product $\square\ \Sigma_{i=1}^n x_i\cdot y_i$ can be computed for all roundings $\square \in \{\nabla,\Delta,\square_\mu(\mu = 0(1)b)\}$. As matrix products and linear mappings are componentwise defined as scalar products, we can compute the following functions:

$$T^n \times T^n \ni (x,y) \to \square(x\cdot y) \in T \quad \text{(scalar product)}$$

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

$$T^n \ni x \to \square(C\cdot x + c) \in T^n \quad \text{(linear mapping)}$$

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. Furthermore, 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 at 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 $T = T_{b,l}$, $l \geqq 3$, be a floating-point system and $T^{n\times n}$ the set of $n\times n$ matrices with components in T. The floating-point matrix operations $\boxed{*}: T^{n\times n}\times T^{n\times n} \to T^{n\times n}$ are defined by

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

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

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

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

c) $$\bigwedge_{A,B\in T^{n\times n}}\ \bigwedge_{\square\in\{\square_\mu(\mu=0(1)b)\}}\Big((-A)\ \boxplus\ (-B) = -A\ \boxplus\ B \wedge (-A)\ \square\ B = A\ \square\ (-B) = -A\ \square\ B\Big)$$

$$\bigwedge_{A,B\in T^{n\times n}}\Big((-A)\ \overline{\boxplus}\ (-B) = -A\ \underline{\boxplus}\ B \wedge (-A)\ \underline{\boxplus}\ (-B) =$$

$$-A\ \overline{\boxplus}\ B \wedge (-A)\ \nabla\ B = A\ \nabla\ (-B) =$$
$$-A\ \Delta\ B \wedge (-A)\ \Delta\ B = A\ \Delta\ (-B) = -A\ \Delta\ B\Big)$$

d) $\bigwedge_{A,B\in T^{n\times n}} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu\}} |A * B - A \pm B| \leqq \epsilon^* |A * B|.$

III. 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 $\Pi_{i=1}^n x_i$ by a double-length approximation $\widetilde{\Pi_{i=1}^n x_i} \in T_{b,2l}$ and return to using $\Pi_{i=1}^n x_i$ only if this double-length result turns out to be too inaccurate. The following lemma gives a criterion for the property

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

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 ∇_{2l}: $\mathbf{R} \to T_{b,2l}$ from (2), an approximation $p_2 := \widetilde{\Pi_{i=1}^n x_i}$ can be defined by

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

With $\operatorname{mant}(p_2) = m = 0 \cdot m[1] \cdots m[2l]$, (14) holds if

$$0 \cdot m[1] \cdots m[l+1] < m \wedge 0 \cdot m[1] \cdots m[l+1] + b^{-(l+1)} > m + 2(n-2)b^{-(2l-1)}. \tag{16}$$

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 inequality delivers

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

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

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

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

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

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

$$(16) \Rightarrow (0 \cdot m[1] \cdots m[l+1] < m \wedge 0 \cdot m[1] \cdots m[l+1] + b^{-(l+1)} > m \cdot (1 + 2(n-2)b^{-(2l-1)})) \Rightarrow I \cap T_{b,l+1} = \phi \Rightarrow (14)] \blacksquare \tag{17}$$

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

a) $sp \in \{+1,-1\}$, $mp = 0 \cdot mp[1] \cdots mp[l+2]$, $ep \in \mathbf{Z}$
variables for the sign, mantissa, and exponent of the product p, respectively; 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 $(mp[l+2] = 0)$;

b) $m_i = m_i[0] \cdot m_i[1] \cdots m_i[l]$, $i = 1(1)n$
auxiliary variables with a carry digit $m_i[0]$; ma, mb, mc, md alike; $m = 0 \cdot m[1] \cdots m[2l]$ is a double-length mantissa;

c) $m := m_j \times m_i$
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) $(ma, mb) := m$
the decomposition of the double-length mantissa m into two single-length mantissas $ma := 0 \cdot m[1] \cdot m[l]$, $mb := 0 \cdot m[l+1] \cdots m[2l]$.

Then the following theorem holds:

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

a) $\bigwedge_{(x_i)\in T^n} \bigwedge_{\square\in\{\nabla,\Delta,\square_\mu(\mu=0(1)b)\}} \square \prod_{i=1}^n x_i = \square p.$

Furthermore, the following properties hold:

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

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

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

Proof: b)–d) follow immediately from Theorem 1, so we only have to prove a).

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

Case 2: If "error = 0" 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 $md > 0 \wedge 1 > md + 2(n-2)b^{-(l-2)}$, 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 $\neq 0$ 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 $\square \in \{\nabla,\Delta,\square_\mu\}$, 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 longer used. So, for all $j = 2(1)n$, we have

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

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

The double loop in the normalization algorithm ν can be run through at most $n \cdot l$ times, because the product is not zero. After this loop the jth 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 $\neq 0$ truncated off. ■

Remarks: 1) Condition (16) of Lemma 3 shows that for reasonable n the approximation p_2 is mostly 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 $T = T_{10,2}$ by the following example:

$$0.2 \in T \quad 0.5 \in T \quad 0.2^{14} \cdot 0.5^{14} = 0.1 \cdot 10^{-13} \in T$$

but the intermediate result $0.2^{14} = 0.16384 \cdot 10^{-9}$ is no double-length floating-point number; i.e., $0.2^{14} \notin T_{10,4}$. Therefore, the product $0.2^{14} \cdot 0.5^{14}$ cannot be computed exactly by recursive double-length multiplication.

2) If $\exp(x_i) \in \{e1, \cdots, e2\}$ $(i = 1(1)n)$, then

$$\exp\left(\prod_{i=1}^{n} x_i\right) \in \{n \cdot e1 - n + 1, \cdots, n \cdot e2\}.$$

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 loss could be avoided at the cost of storage space and complexity of π_n.

IV. Roots of Floating-Point Numbers

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

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

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

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

$$\bigwedge_{X,Y\in \mathbf{I}T} X \diamondsuit Y := \Diamond (X * Y),\ * \in \{+,-,\times,/\}. \tag{19}$$[3]

In the case of the division, $0 \notin Y$ 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, $x_1^{(0)}, x_2^{(0)} \in T$, $0 < x_1^{(0)} < x_2^{(0)}$, two positive floating-point numbers, and $f: X^{(0)} := [x_1^{(0)}, x_2^{(0)}] \to \mathbf{R}$ a real-valued function with the following properties:

$$\text{a)} \bigwedge_{\xi\in X^{(0)}} f(\xi) = 0$$

$$\text{b)} \bigwedge_{m_1,m_2\in T} \bigwedge_{x\in X^{(0)}\setminus\{\xi\}} 0 < m_1 \leqq \frac{f(x)}{x-\xi} \leqq m_2 < \infty, \qquad M := [m_1,m_2]$$

c) there is a function $F: X^{(0)} \cap T \to \mathbf{I}T$, with

$$\text{c1)} \bigwedge_{x\in X^{(0)}\cap T} f(x) \in F(x)$$

$$\text{c2)} \bigwedge_{x\in X^{(0)}\cap T} (F(x) \leqq [0,0] \vee F(x) \geqq [0,0])$$

$$\text{c3)}\ \xi \in T \Rightarrow F(\xi) = [0,0]$$

where ξ is the zero from a).

Starting with $X^{(0)}$, let a sequence $(X^{(k)})_{k=0,1,2,\ldots}$ of intervals $X^{(k)} = [x_1^{(k)}, x_2^{(k)}] \in \mathbf{I}T$ be generated by

$$\text{d)}\ X^{(k+1)} := X^{(k)} \cap ([m(X^{(k)}), m(X^{(k)})] \diamondsuit F(m(X^{(k)})) \diamondsuit M), \qquad k = 0,1,2,\cdots$$

wherein $m(X^{(k)})$ fulfills the condition e) $m(X^{(k)}) \in X^{(k)} \cap T$ and

$$\text{e1)}\ m(X^{(k)}) \neq x_1^{(k)}, x_2^{(k)}$$

if $X^{(k)} \cap T$ has at least 3 elements

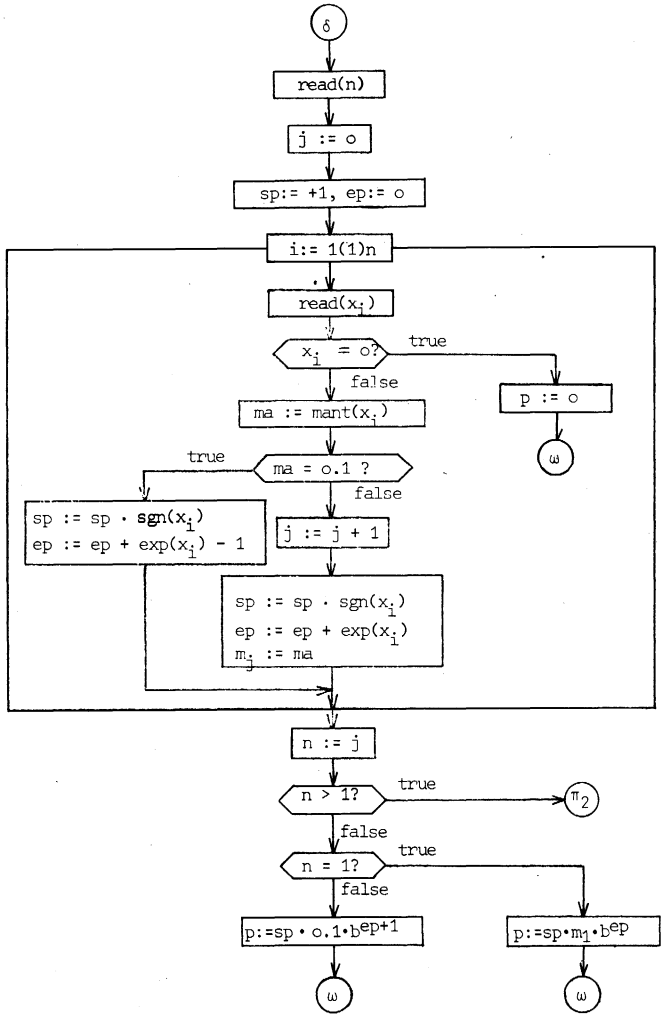
$$\text{e2)}\ m(X^{(k)}) = \begin{cases} x_1^{(k)}, & \text{if } k \text{ is even} \\ x_2^{(k)}, & \text{if } k \text{ is odd} \end{cases}, \qquad \text{otherwise.}$$

For this sequence $(X^{(k)})_{k=0,1,\ldots}$, the following properties hold:

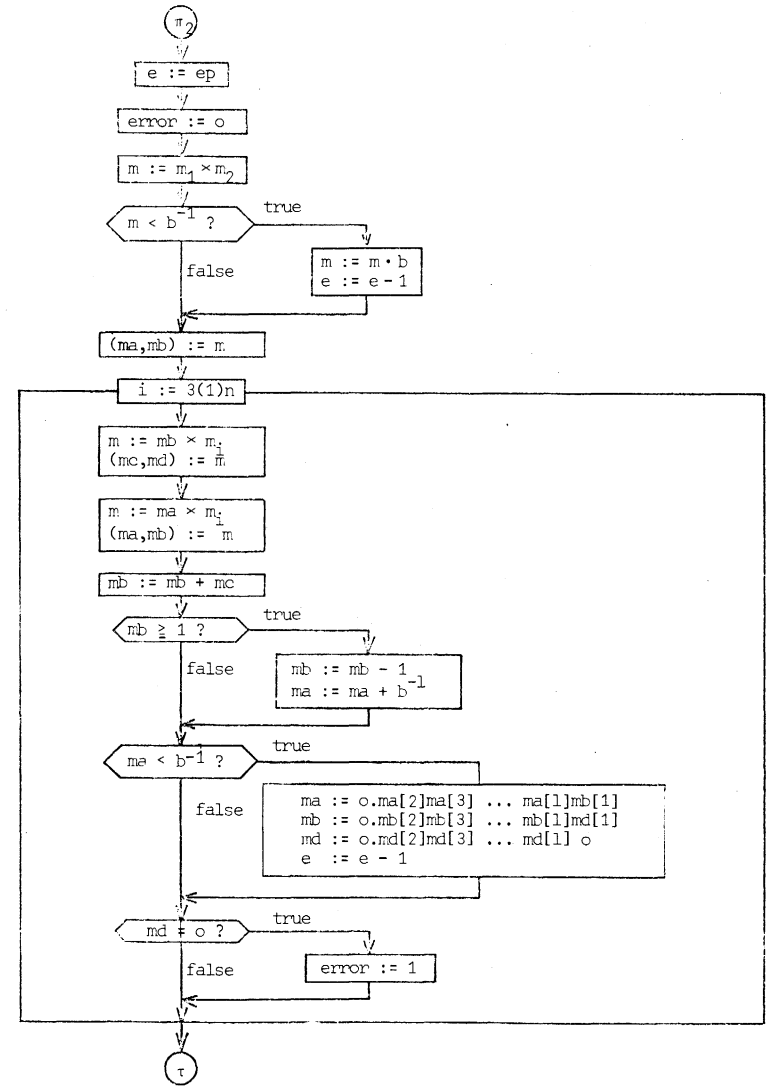
$$\text{f)} \bigwedge_{k\in \mathbf{N}} \Diamond [\xi,\xi] \subseteq X^{(k)},$$

$$\text{g)} \bigwedge_{k\in \mathbf{N}} X^{(k)} \cap T \neq \phi,$$

[3] For real intervals $X, Y \in \mathbf{IR}$ the operations $*$ are defined by $X * Y := \{x * y; x \in X, y \in Y\}$; the order relations $\leqq$ and $<$ are defined by $[x_1,x_2] \leqq [y_1,y_2] :\Leftrightarrow (x_1 \leqq y_1 \wedge x_2 \leqq y_2)$, $X < Y :\Leftrightarrow (X \leqq Y \wedge X \neq Y)$.

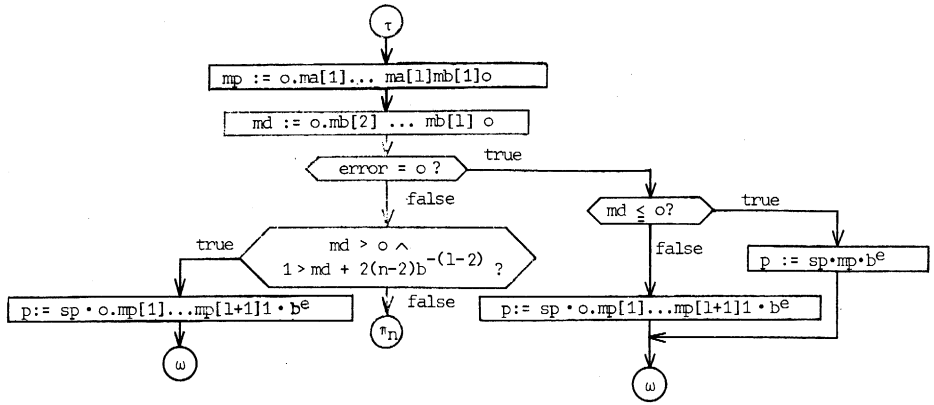


2.1: Decomposition.

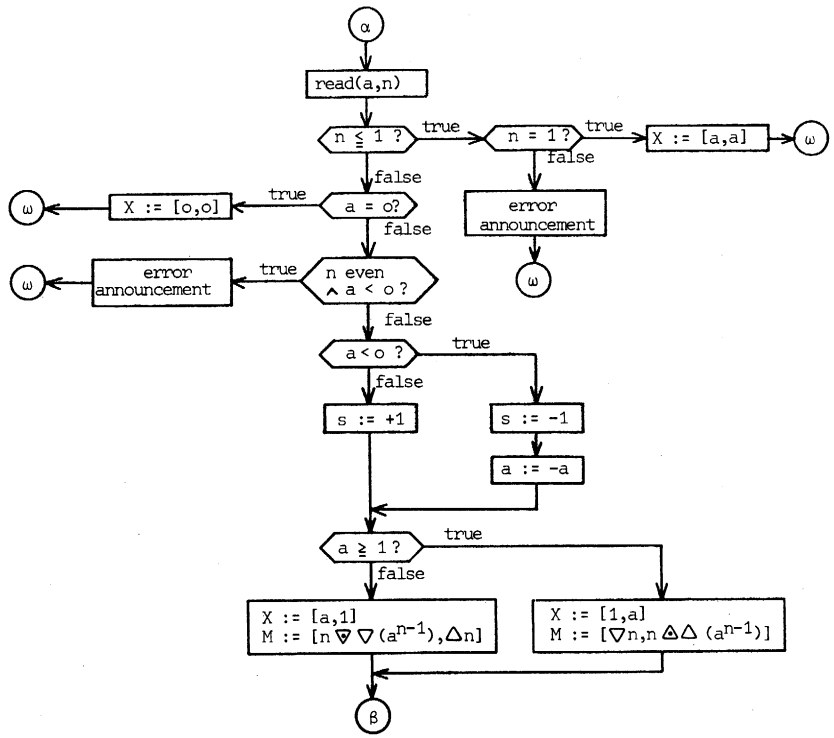


2.2: Double-length product.

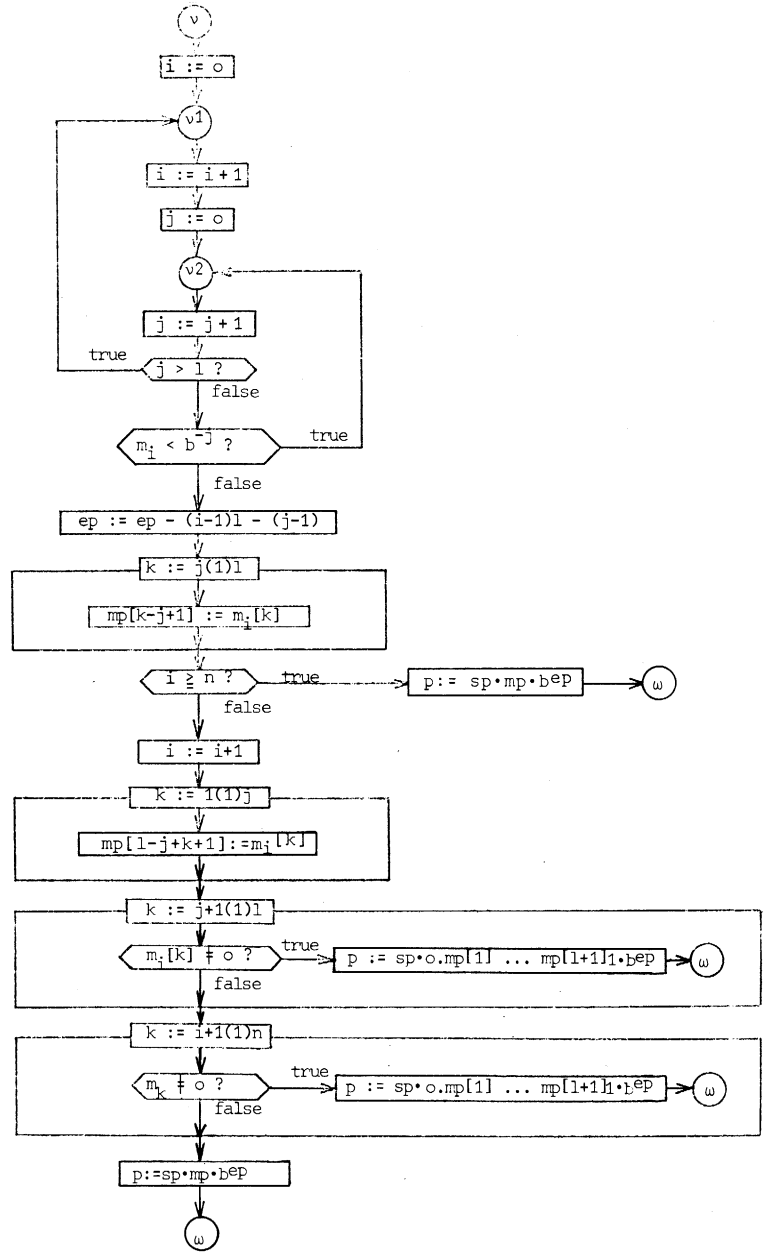
Algorithm 2. Product of n floating-point numbers.



2.3: **Test of precision.**



2.4: **Exact product.**



2.5: **Normalization.**

Algorithm 2. Product of n floating-point numbers.

h) $\bigwedge_{k_0 \in \mathbf{N}} X^{(0)} \supset X^{(1)} \supset \cdots \supset X^{(k_0)}$

$$= X^{(k_0+1)} \supseteq X^{(k_0+2)} = X^{(k_0+3)} = \cdots,$$

i) $X^{(k_0+2)} = \Diamond[\xi,\xi] = [\nabla\xi,\Delta\xi]$.

Proof: Properties f) and g) have to be proved simultaneously by induction. As the properties trivially hold for $k = 0$, we can assume that they hold for any given $k \in \mathbf{N}$. 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 and Herzberger [1]:

From the identity

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

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 $0 \notin M$ and that this property holds also for $m(X^{(k)}) = \xi$. With c1) and the definition of the rounding $\Diamond$, we get

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

$$\Rightarrow \Diamond[\xi,\xi] \subseteq \Diamond([m(X^{(k)}), m(X^{(k)})] - F(m(X^{(k)})) \Diamond M)$$
$$= [m(X^{(k)}), m(X^{(k)})] \Diamond F(m(X^{(k)})) \Diamond M.$$

$$\Rightarrow \Diamond[\xi,\xi] \subseteq X^{(k+1)}.$$

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: $X^{(0)} \cap T \to T$ by $F(x) =: [f_1(x), f_2(x)]$ for all $x \in X^{(0)} \cap T$. Then method d) can be expressed componentwise

$$x_1^{(k+1)} = \begin{cases} \max\{x_1^{(k)}, m(X^{(k)}) \nabla f_2(m(X^{(k)})) \Delta m_1\}, \\ \qquad \text{if } f_2(m(X^{(k)})) > 0 \\ m(X^{(k)}) \nabla f_2(m(X^{(k)})) \Delta m_2, \\ \qquad \text{if } f_2(m(X^{(k)})) \leqq 0 \end{cases}$$

$$x_2^{(k+1)} = \begin{cases} m(X^{(k)}) \Delta f_1(m(X^{(k)})) \nabla m_2, \\ \qquad \text{if } f_1(m(X^{(k)})) \geqq 0 \\ \min\{x_2^{(k)}, m(X^{(k)}) \Delta f_1(m(X^{(k)})) \nabla m_1\}, \\ \qquad \text{if } f_1(mX^{(k)})) < 0. \end{cases} \quad (20)$$

Now we distinguish the two cases of e).

Case 1: $X^{(k)} \cap T$ has at least 3 elements. Then $x_1^{(k)} < m(X^{(k)}) < x_2^{(k)}$.

Because of property c2), we get

$\alpha)$ $F(m(X^{(k)})) \geqq 0 \Rightarrow f_1(m(X^{(k)})) \geqq 0 \Rightarrow x_2^{(k+1)}$
$= m(X^{(k)}) \Delta f_1(m(X^{(k)})) \nabla m_2 \leqq m(X^{(k)}) < x_2^{(k)}$

or

$\beta)$ $F(m(X^{(k)})) \leqq 0 \Rightarrow f_2(m(X^{(k)})) \leqq 0 \Rightarrow X_1^{(k+1)}$
$= M(X^{(k)}) \nabla f_2(m(X^{(k)})) \Delta m_2 \geqq m(X^{(k)}) > x_1^{(k)}$

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 $X^{(0)} \cap T$ is a finite set, an index k_0' must exist, so that $X^{(k_0')} \cap T$ has at most 2 elements.

Case 2: $X^{(k)} \cap T$ has at most 2 elements. Then one of the following two cases occurs. $\alpha)$ $X^{(k)} \cap T$ has exactly one element. Then

$$X^{(k+1)} = X^{(k)} = \Diamond[\xi,\xi] = [\xi,\xi]$$
$$\Rightarrow X^{(k+1)} = X^{(k+2)} = \cdots = X^{(k)} = \Diamond[\xi,\xi].$$

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

$$\beta 1)\ \xi = x_1^{(k)} \Leftrightarrow f(x_1^{(k)}) = 0 \Leftrightarrow F(x_1^{(k)}) = [0,0]$$

$$\beta 2)\ \xi = x_2^{(k)} \Leftrightarrow f(x_2^{(k)}) = 0 \Leftrightarrow F(x_2^{(k)}) = [0,0]$$

$$\beta 3)\ x_1^{(k)} < \xi < x_2^{(k)} \Leftrightarrow f(x_1^{(k)}) < 0 < f(x_2^{(k)})$$
$$\Leftrightarrow F(x_1^{(k)}) < [0,0] < F(x_2^{(k)})$$
$$(c)$$

In case $\beta 3)$, we have

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

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

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

or

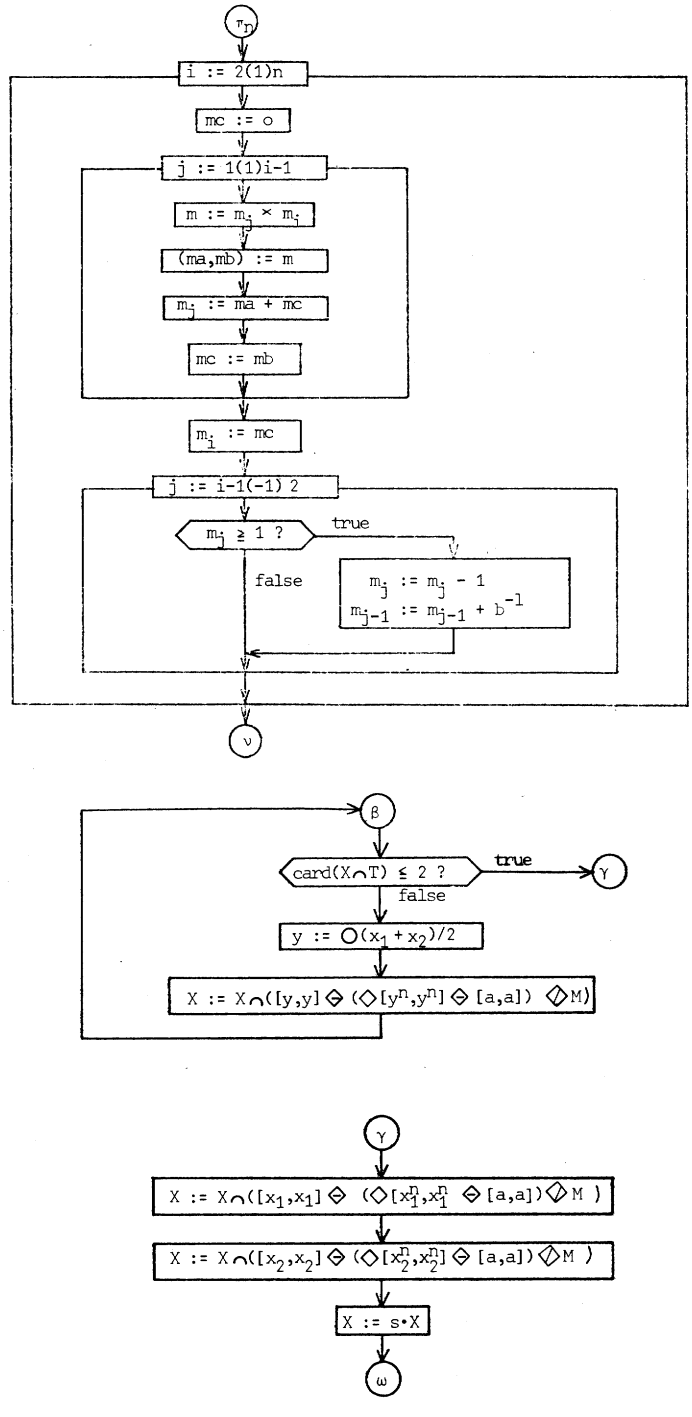
$$X^{(k)} = X^{(k+1)} \supset X^{(k+2)} = \cdots = [\xi,\xi] = \Diamond[\xi,\xi]. \quad \blacksquare$$

Remarks: 1) Method d) delivers $\Diamond[\xi,\xi] = [\nabla\xi,\Delta\xi]$, 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 $[\nabla\xi,\Delta\xi]$ by trying out all floating-point numbers in $X^{(0)}$.

3) The assumptions a) and b) imply that ξ is the only zero of the function f in the interval $X^{(0)}$ and that f is Lipschitz-continuous at the place ξ. Apart from this, f need neither be monotone nor continuous.

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



Algorithm 3. Root of a floating-point number.

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 $a \geqq 0$ or n is odd, then it delivers a floating-point interval $X \in \mathbf{I}T$ with the property

$$\text{a)} \bigwedge_{a \in T} X = \Diamond [\sqrt[n]{a}, \sqrt[n]{a}] = [\nabla \sqrt[n]{a}, \Delta \sqrt[n]{a}].$$

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

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

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

$$\text{d)} \bigwedge_{a \in T} \bigwedge_{\square \in \{\nabla, \Delta, \square_\mu\}} |\sqrt[n]{a} - \square \sqrt[n]{a}| \leqq \epsilon^* \cdot |\sqrt[n]{a}|$$

and, if n is odd

e) $\bigwedge_{a\in T} \bigwedge_{\square\in\{\square_\mu;\mu=0(1)b\}} \square\sqrt[n]{-a} = -\square\sqrt[n]{a}$

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

Proof: a) is trivial for $a = 0$. For $a > 0$, Theorem 5 is used with

$$X^{(0)}: = \begin{cases}[1,a], & \text{if } a \geqq 1\\ [a,1], & \text{if } a < 1\end{cases}$$

$$M: = \begin{cases}[\nabla n, n \mathbin{\blacktriangle} \Delta(a^{n-1})], & \text{if } a \geqq 1\\ [n \mathbin{\blacktriangledown} \nabla(a^{n-1}), \Delta n], & \text{if } a < 1\end{cases}$$

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

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

wherein $\Diamond[x^n,x^n] = [\nabla x^n, \Delta x^n]$ is computed with the product from Section III. Then assumptions a)–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 $a < 0$, $\sqrt[n]{a}$ does only exist, if n is odd. This case can be easily reduced to the case $a > 0$. Properties b)–e) follow immediately from Theorem 1. ■

Remarks: 1) Algorithm 3 delivers the smallest interval containing $\sqrt[n]{a}$, whereas in properties b)–e), the rounded result $\square\sqrt[n]{a}$ for $\square\in\{\nabla,\Delta,\square_\mu\}$ is needed. As can be seen from (2)–(5) and (18), $\square\sqrt[n]{a}$ for $\square \in \{\nabla,\Delta,\square_0,\square_b\}$ can be determined from $\Diamond[\sqrt[n]{a}, \sqrt[n]{a}]$. For $\square_\mu \sqrt[n]{a}$ $(\mu = 1(1)b - 1)$, Algorithm 3 has to be executed in $T = T_{b,l+1}$.

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

3) The speed of the algorithm could be improved by computing better starting values of $X^{(0)}$ and M and by replacing the interval M by a sequence of intervals $(M^{(k)})_{k=0,1,2,\ldots}$ (see, e.g., Herzberger [3]).

References

[1] G. Alefeld and J. Herzberger, *Einführung in die Intervallrechnung.* Mannheim: Bibliographisches Institut, 1974.

[2] K. Grüner, "Fehlerschranken für lineare Gleichungssysteme," presented at the MRI Oberwolfach, 1975.

[3] J. Herzberger, "Über die Nullstellenbestimmung bei näherungsweise berechneten Funktionen," *Comput.*, vol. 10, pp. 23–31, 1972.

[4] W. Kahan, "Further remarks on reducing truncation errors," *Commun. ACM*, vol. 8, p. 40, Jan. 1965.

[5] U. Kulisch, "An axiomatic approach to rounded computations," *Numer. Math.*, vol. 18, pp. 1–17, 1971.

[6] ——, *Über die Arithmetik von Rechenanlagen. Jahrbuch Überblicke der Mathematik.* Mannheim: Bibliographisches Institut, 1975.

[7] U. Kulisch and G. Bohlender, "Formalization and implementation of floating-point matrix operations," *Comput.*, vol. 16, pp. 239–261, 1976.

[8] P. Linz, "Accurate floating-point summation," *Commun. ACM*, vol. 13, pp. 361–362, June 1970.

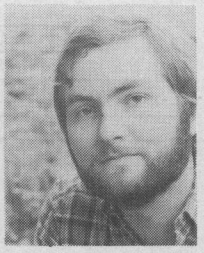
[9] M. A. Malcolm, "On accurate floating-point summation," *Commun. ACM*, vol. 14, pp. 731–736, Nov. 1971.

[10] M. Pichat, "Correction d'une somme en arithmetique a virgule flottante," *Numer. Math.*, vol. 19, pp. 400–406, 1972.

[11] J. H. Wilkinson, *Rounding Errors in Algebraic Processes.* Englewood Cliffs, NJ: Prentice-Hall, 1963.

[12] J. M. Yohe, "Interval bounds for square roots and cube roots," *Comput.*, vol. 11, pp. 51–57, 1973.

[13] ——, "Roundings in floating-point arithmetic," *IEEE Trans. Comput.*, vol. C-12, pp. 577–586, June 1973.



Gerd Bohlender was born in Kandel, Germany, on November 3, 1950. He received the Diplome in mathematics from the University of Karlsruhe, Karlsruhe, Germany, in 1973.

After receiving his degree he joined the Institute of Applied Mathematics, University of Karlsruhe. His current interests are in floating-point computation and the evaluation of mathematical functions.