Numer. Math. 18, 224–242 (1971)

A Floating-Point Technique for Extending the Available Precision

T. J. DEKKER*

Received July 26, 1971

Abstract. A technique is described for expressing multilength floating-point arithmetic in terms of singlelength floating point arithmetic, i.e. the arithmetic for an available (say: single or double precision) floating-point number system. The basic algorithms are exact addition and multiplication of two singlelength floating-point numbers, delivering the result as a doublelength floating-point number. A straightforward application of the technique yields a set of algorithms for doublelength arithmetic which are given as ALGOL 60 procedures.

1. Introduction

Multilength floating-point arithmetic is often described in terms of integer arithmetic. The mantissa of a multilength floating-point number is then represented by means of some integers and another integer is used for the exponent.

In contrast with this approach, we present some basic algorithms which enable us to describe multilength floating-point arithmetic in terms of singlelength floating-point arithmetic. A multilength floating-point number is then represented by means of some singlelength floating-point numbers. In particular, we represent a doublelength floating-point number as the sum of two singlelength floating-point numbers, one of them being (almost) negligeable in singlelength precision. The basic algorithms are exact addition and multiplication of two singlelength floating-point numbers, delivering the result as a doublelength floating-point number.

After two introductory sections on floating-point systems and arithmetic, we deal with exact addition in Section 4.

Let x and y be singlelength floating-point numbers and let

$$z = fl(x+y);$$

i.e. z is the result of a singlelength floating-point addition of x and y. Let zz be the correction term exactly satisfying

$$z+zz = x+y.$$

It will be shown that, under various conditions, zz can be obtained by the formula

$$zz = fl((x-z)+y).$$

* Report MR 118/70, Computation Department, Mathematical Centre, Amsterdam. Part of this research was done while the author was visiting Bell Telephone Laboratories, Murray Hill, New Jersey.

For example, this is true if the singlelength floating-point system is binary, addition and subtraction are optimal and $|x| \geqq |y|$. Similar formulas have been used by Møller (1965), Kahan (1965), Babuška (1968) and Knuth (1969) for reducing the error in calculating the sum of several terms.

Møller (1965) gives a more elaborate formula which turns out to be a double application of ours. Knuth (1969) shows that this formula yields the correction term zz if floating-point addition and substraction are optimal.

In Sections 5 and 6, we deal with exact multiplication of two singlelength floating-point numbers.

First, each factor is splitted into two "halflength" numbers, and then the exact product is formed in a rather obvious way. Splitting the factors as well as forming the exact product are expressed in terms of singlelength floating-point operations which, under certain reasonable conditions, are shown to yield the correct results.

Veltkamp (1968) developed a similar technique with a slightly modified algorithm for exact multiplication.

In Section 7, we give a straightforward application of the foregoing results to obtain algorithms for doublelength arithmetic. In the appendix, the algorithms are given as ALGOL 60 procedures.

An important application (in fact, the main incentive for this research), is the calculation of the doublelength scalar product of two vectors of singlelength floating-point numbers.

Application of our results to multilength arithmetic we hope to present in a future publication.

A drawback of our technique is that it works only if the singlelength floating-point system and arithmetic satisfy certain conditions (which are fulfilled, for example, on the General Electric 635 and Philips Electrologica X8 computers).

Our technique is then, of course, most effective if we take as singlelength system the highest precision floating-point system available. Therefore, an essential feature of our algorithms is that, with respect to this singlelength system, no doublelength accumulator is used.

Our technique is especially attractive if the mantissa length of the highest precision floating-point system available is larger than the capacity of an available fixed-point (i.e. integer) system, because then less work is needed to obtain a certain required precision.

Moreover, since the floating-point operations used take care of the shifting and normalizing needed at various stages, a reasonable efficiency can be reached by writing our algorithms entirely in a high-level language such a ALGOL or FORTRAN.

2. Floating-Point Number Systems

The starting-point for our technique is a certain system, R, of floating-point numbers, and the floating-point arithmetical operations defined on R.

Let the base, β, of R be an integer larger than 1.

We assume that the mantissa as well as the exponent of the elements of R are integers within a certain range.

More precisely, we assume:

1) a floating-point number x has the form

$$x = mx\,\beta^{ex}, \tag{2.1}$$

where mx, or $m(x)$, denotes the *mantissa*, and ex, or $e(x)$, the *exponent* of x;

2) the system R of floating-point numbers is

$$R = \{x \mid x = mx\,\beta^{ex},\ |mx| < M,\ -D < ex < E\}, \tag{2.2}$$

where M is a positive integer (usually $M = \beta^t$, where t is the number of mantissa digits) and D, E are positive integers or infinite (the latter if we want to disregard underflow and/or overflow).

Floating-point number systems of the form (2.2) were proposed by Grau (1962) and are actually used in some computers, e.g. Burrough B 5500 (octal) and Philips Electrologica X8 (binary). Apart from overflow and underflow, the floating-point systems have this form in most, if not all, computers, since the mantissa can always be interpreted as an integer by subtracting a suitable constant from the exponent.

Remarks. 1) Representation (2.1) is not always unique. In most machines, a certain standardization is defined in order to make the respresentation unique. The most common standardization is the *normalization* in which the magnitude of a non-zero mantissa has a lower bound M/β. Some elements of R having an exponent (nearly) equal to $-D$ do not have a normalized representation due to underflow in the exponent range. A system of the form (2.2) without requiring normalization has the advantage, that addition and subtraction never lead to underflow.

2) We shall assume that M is not congruent to 1 (mod β), because this slightly simplifies some theorems and proofs. This assumption is no true restriction if we disregard overflow. Indeed, if $M = 1 + \beta K$ for some integer K, then M may be replaced by βK without changing the system R, except that (for finite E) the elements $\pm M\beta^E$ are removed from the system. This remark also justifies the restriction to a symmetric mantissa range, since, for $M = \beta K$, the not uncommon asymmetric mantissa range $-(1 + M) < mx < M$ defines the same system, apart from overflow, as the symmetric mantissa range given in (2.2).

3. Floating-Point Operations

We use the following notation from Wilkinson (1963, p. 4). If A is an expression involving floating-point numbers and the arithmetical operations $+$, $-$, $\times$, $/$, then $fl(A)$ is the corresponding expression obtained by replacing the arithmetical operations by the corresponding floating-point operations.

Let $*$ represent $+$, $-$, $\times$ or $/$, and let x and y be elements of R.

(3.1) **Definition.** The floating-point operation corresponding to $*$ is *faithful* if, for all x and y, the result $fl(x*y)$ equals either the largest element of R smaller than or equal to $x*y$, or the smallest element of R larger than or equal to $x*y$.

Thus, when $x*y$ lies between two successive elements of R, either one will do, but when $x*y \in R$, the result must be exact (and when $x*y$ is outside the range of R, the result must be the nearest, i.e. the largest or the smallest, element of R).

(3.2) **Definition.** The floating-point operation corresponding to $*$ is *optimal* (or *properly rounding*) if, for all x and y, the result $fl(x*y)$ is an element of R nearest to $x*y$.

Note that this definition uniquely determines the result, except when $x*y$ lies halfway between two successive elements of R, in which case an optimal operation may round up or down.

Related to this definition is the following notation which will be used below.

(3.3) *Notation.* For any real ξ, round (ξ) denotes an integer closest to ξ.

The result $z=fl(x*y)$ of a properly rounding operation can be represented in the form (2.1), where mantissa mz and exponent ez satisfy

$$mz = round\,(x*y\,\beta^{-ez}), \tag{3.4}$$

provided that no overflow or underflow occurs.

For addition and subtraction, we also need the following definitions.

(3.5) **Definition.** Floating-point addition is *properly truncating* if it is commutative (i.e. $fl(x+y)=fl(y+x)$) and, for all x and y satisfying $|x|\geqq|y|$, the result $fl(x+y)$ equals the largest element of R smaller than or equal to $x+y$ if $y\geqq 0$ or the smallest element of R larger than or equal to $x+y$ if $y<0$.

Thus, when $x+y$ does not belong to R, it is truncated in the direction of $-y$. Note that this definition uniquely determines the result.

(3.6) **Definition.** Floating-point subtraction is *properly truncating* if, for all x and y, we have $fl(x-y)=fl(x+y')$, where $y'=-y$ and the floating-point addition is properly truncating.

(3.7) **Definition.** Floating-point addition and subtraction are *superfaithful* if, for each x and y, the result $fl(x\pm y)$ is obtained by properly rounding or by properly truncating.

Remark. Arithmetical operations having these properties are not difficult to realize.

In fact, there are machines in which the floating-point addition and subtraction are optimal (e.g. General Electric 635, Philips Electrologica X8). In order to obtain faithful addition and subtraction, the result must only be normalized before it is truncated or rounded, cf. Kahan (1965). To obtain a super-faithful, or even optimal, addition and subtraction, it is by no means necessary to have a (nearly) doublelength accumulator. Optimal addition and subtraction can be perfectly well formulated using an accumulator having no more than two guarding digits; see Knuth (1969, p. 183 Algorithms A and N and p. 194 exercise 5).

4. Exact Addition

Let x and y be given elements of R and let

$$z=fl(x+y). \tag{4.1}$$

We want to find the correction term zz satisfying the (exact) relation

$$z+zz=x+y. \tag{4.2}$$

We shall derive some formulas for calculating zz which use only singlelength floating-point addition and subtraction.

First we consider the formula

$$w = fl(z - x), \quad zz = fl(y - w) \tag{4.3}$$

and prove some theorems stating sufficient conditions for the validity of this formula. For practical computation, formulas (4.1) and (4.3) can be written as the following sequence of ALGOL 60 statements:

$$\text{“}z := x + y; \quad zz := y - (z - x)\text{”}, \tag{4.4}$$

in which w remains anonymous.

Let $x, y \in R$ be representable such that their exponents satisfy

$$ex \geqq ey. \tag{4.5}$$

In particular, this holds if $x, y \in R$ satisfy

$$|x| \geqq |y|. \tag{4.6}$$

(4.7) **Theorem.** If R has the form (2.2), where $\beta = 2$ or 3 and M is a multiple of β, and if, moreover, floating-point addition is optimal and subtraction faithful, then, for all x and y satisfying (4.5) and for z obtained according to (4.1), formula (4.3) yields the correction term zz defined by (4.2).

Proof. According to (4.5), we may assume

$$x = mx\,\beta^{ex}, \quad y = my\,\beta^{ey}, \quad ex \geqq ey.$$

Since floating-point subtraction is faithful, we need to show only

1) $z - x \in R$,

2) $y - w \in R$.

Proof of (1). Obviously z can be represented such that $ez \leqq ex + 1$. If $ez \leqq ex$, then overflow may or may not occur in forming z; in both cases the result easily follows.

If $ez = ex + 1$, then obviously no overflow has occurred in forming z. Let $d = ex - ey$ and $\mu = \beta mz - mx$. Then

$$mz = round\,(mx/\beta + my/\beta^{d+1}), \quad z - x = \mu\beta^{ex},$$

where μ satisfies

$$|\mu| \leqq |\beta mz - mx - my/\beta^{d}| + |my/\beta^{d}| < \beta/2 + M.$$

Hence, since $\beta \leqq 3$ and μ is integral, we have $|\mu| \leqq M$.

From this and the fact that M is a multiple of β, the result easily follows.

Proof of (2). From $ex \geqq ey$, it follows that $y - w$ is equal to an integer times β^{ey}. Moreover, $|y - w| \leqq |y|$, because otherwise x would be closer to $x + y$ than z contradicting the assumption that floating-point addition is optimal. Hence, $y - w \in R$.

This completes the proof of (2) and the theorem.

Theorem (4.7) does not hold for $\beta \leqq 3$ if M is not a multiple of β (which, according to Section 2 Remark 2 can only occur for $\beta = 3$), nor for any $\beta > 3$.

For example, if $\beta = 3$, $M = \frac{1}{2}(3^t + 1)$ (this being the most natural range in the "balanced" ternary system using the digits $+1$, 0, -1, see Knuth (1969, p. 173)) and $x = y = M - 1$, then $z = 3^t$ and $z - x = \frac{1}{2}(3^t + 1)$, which is not an element of R; if $\beta = 10$, $M = 100$ and $x = y = 98$, then $z = 200$ and $z - x = 102$, which is not an element of R.

The theorem can be extended to any β and M, provided that an enlarged mantissa range (an extra digit, say) is available for w. Such an extended theorem has practical applications, because w may remain anonymous (see 4.4) and some systems have an enlarged mantissa range for the anonymous floating-point values.

Therefore, we define an enlarged system R^* for w, analogous to (2.2), as follows.

$$R^* = \{x \mid x = mx\,\beta^{ex},\ |mx| < M^*,\ -D < ex < E\}, \tag{4.8}$$

where $M^* \geqq M + \beta/2$. (This is certainly true if R^* has at least one more digit for the mantissas of its elements than R.)

Then the following theorem holds without any restriction on β and M.

(4.9) **Theorem.** Let R have the form (2.2) and R^* the form (4.8). Let $x, y \in R$ satisfy (4.5) and let $z \in R$ be obtained according to (4.1), where the floating-point addition is optimal. Furthermore, let formula (4.3) be calculated such that the subtraction producing $w \in R^*$ is faithful with respect to R^* and the other operation is faithful with respect to R. Then formula (4.3) yields the correction term $zz \in R$ defined by (4.2).

Proof. The theorem is proved by showing that $z - x$ is an element of R^* and $y - w$ is an element of R. The proof is identical to that of Theorem (4.7) apart from an obvious modification in the first part of the proof.

For properly truncating addition, the following theorem holds without any restriction on β and M and without requiring an enlarged mantissa range for w.

(4.10) **Theorem.** If R has the form (2.2), floating-point addition is properly truncating and subtraction is faithful, then, for all x and y satisfying (4.5) and for z obtained according to (4.1), formula (4.3) yields the correction term zz defined by (4.2).

Proof. The theorem is proved, in a similar way as Theorem (4.7), by showing that both $z - x$ and $y - w$ are elements of R.

Remarks. 1) As pointed out in Section 2 Remark 1, addition and subtraction never lead to underflow in a system of the form (2.2). If we restrict ourselves to normalized representations, then theorems (4.7), (4.9) and (4.10) do not remain valid, since underflow may occur in forming zz. Of course, the theorems remain valid in cases where no underflow occurs.

2) The theorems do not hold if addition is only faithful, because the correction term zz is then not always an element of R. For example, if $\beta = 2$, $M = 16$, $x = 15$, $y = 15/32$ and $z = 16$, then $zz = -17/32$, which is not an element of R.

For super-faithful addition (see definition 3.7), we immediately obtain by combining theorems (4.7) and (4.10):

(4.11) **Corollary.** Theorem (4.7) remains valid, if "optimal" is replaced by "super-faithful".

Similarly, combining Theorems (4.9) and (4.10), we obtain

(4.12) **Corollary.** Theorem (4.9) remains valid, if "optimal" is replaced by "super-faithful".

Instead of formula (4.3), we now consider the formula

$$w = fl(x - z), \quad zz = fl(w + y). \tag{4.13}$$

For practical computation, formulas (4.1) and (4.13) can be written as the following sequence of ALGOL 60 statements:

$$\text{"}z := x + y;\ zz := x - z + y\text{"}, \tag{4.14}$$

in which w again remains anonymous. Here, we make use of the fact that, in ALGOL 60, subsequent additions and subtractions are performed from left to right, so that the expression "$x - z + y$" is equivalent to "$(x - z) + y$", see Naur (1962, 3.3.5.). For applications, we prefer this formula above (4.4), because many compilers produce a slightly faster code for (4.14) than for (4.4).

Since the floating-point number system R is *symmetric* (i.e. $x \in R$ implies $-x \in R$, see (2.2) and Section 2 Remark 2), and R^* also (see 4.8), and since optimal, properly truncating or super-faithful addition is certainly faithful, we immediately obtain

(4.15) **Corollary.** Theorems (4.7), (4.9) and (4.10), and corollaries (4.11) and (4.12) remain valid if formula (4.3) is replaced by formula (4.13).

The formulas (4.3) and (4.13) are numerically equivalent but for the sign of w. (Since the second subtraction in (4.3) corresponds to the addition in (4.13), we talk about "other operation" in Theorem (4.9).)

The pair of formulas (4.1) and (4.13), or the equivalent formulation (4.14), is our basic algorithm for *exact addition* of two floating-point numbers. In the applications, it will often be necessary to interchange the roles of the terms x and y when $|x| < |y|$, in order to ensure that (4.5) holds. In the subsequent sections, we shall use this basic algorithm to obtain algorithms for doublelength arithmetical operations.

We conclude this section with a formula for the correction term zz satisfying (4.2), in the case that floating-point addition and subtraction are optimal, R has the form (2.2) with arbitrary β and M, but no enlarged mantissa range is available for anonymous real values (in particular, w). Since we cannot guarantee that $w = z - x$ in (4.3), we apply formula (4.13) to obtain the correction term, $z2$, for w. This leads to the formula

$$\begin{aligned} w &= fl(z - x), \quad z1 = fl(y - w), \\ v &= fl(z - w), \quad z2 = fl(v - x), \\ zz &= fl(z1 - z2). \end{aligned} \tag{4.16}$$

Since we assume that R is symmetric, this formula is equivalent to that given by Møller (1965, p. 42 process A) and Knuth (1969, p. 203 formula 48), the only difference being that there $-z2$ instead of $z2$ is calculated and added to $z1$.

For this formula, the following theorem holds even without requiring (4.5).

(4.17) **Theorem** of Møller-Knuth.

If R has the form (2.2) and floating-point addition and subtraction are optimal, then for all $x, y \in R$ and for z obtained according to (4.1), formula (4.16) yields the correction term zz defined by (4.2).

Møller (1965) has a weaker assumption for floating-point addition and subtraction and obtains a weaker result. Knuth (1969, p. 203 Theorem B) assumes that floating-point addition and subtraction are optimal (p. 197, formula 11), but excludes overflow and underflow. In fact, overflow does not invalidate the theorem, whereas underflow does not occur in a number system R of the form (2.2) (see Section 2 Remark 1).

For a proof of this theorem we refer to Knuth (1969, p. 201-203). An alternative proof can be given along the lines of the proof of Theorem (4.7).

5. Exact Multiplication

Let x and y be given elements of R. We want to calculate their exact product and to deliver it as a pair (z, zz) of elements of R satisfying the (exact) relation

$$z + zz = x \times y, \tag{5.1}$$

with some extra condition that zz be (almost) negligeable within machine precision with respect to $x \times y$.

For simplicity, we assume that the system R of floating-point numbers has the form (2.2), with the restrictions

$$\beta = 2, \quad M = 2^t, \tag{5.2}$$

where t is the number of binary digits in the mantissa, and D and E are infinite. Thus R obtains the form

$$R = R(t) = \{x \mid x = m\,x\,2^{ex},\ |m\,x| < 2^t\}. \tag{5.3}$$

In other words, we restrict ourselves to binary floating-point t-digit arithmetic disregarding overflow and underflow. The following results on exact multiplication can be generalized, however, to nonbinary systems. We use the notation $R(t)$, because we shall also refer to binary floating-point k-digit number systems $R(k)$ for some values $k \neq t$.

Moreover, we assume that floating-point addition and subtraction are optimal and multiplication is faithful (see Definitions 3.1 and 3.2).

In order to form the exact product of x and y, we first split x and y each into two "halflength" numbers. Let $t1$ and $t2$ be (roughly equal) integers such that

$$t = t1 + t2. \tag{5.4}$$

Let hx (the "head" of x) be an element of $R(t2)$ as near to x as possible; i.e. if $x=mx2^{ex}$, where mx is normalized when $x\neq 0$ (thus, ex is minimal and $|mx|\geqq 2^{t-1}$), then

$$hx=round(mx\,2^{-t1})\,2^{ex+t1}. \tag{5.5}$$

(For the definition of round see 3.3.) Let tx (the "tail" of x) be the remaining part of x:

$$tx=x-hx. \tag{5.6}$$

Since hx is obtained by rounding, tx is an element of $R(t1-1)$. Let y similarly split into hy and ty.

Then we obviously have

$$\begin{aligned} &hx\times hy\in R(2t2),\\ &hx\times ty, tx\times hy\in R(t-1),\\ &tx\times ty\in R(2t1-2). \end{aligned}$$

So, in order to make sure, that these numbers fit in $R=R(t)$, we choose (cf. 5.4)

$$t2=entier(t/2),\quad t1=t-t2. \tag{5.7}$$

Then $2t2\leqq t$ and $2t1-2\leqq t-1$. Since $hx\times ty$ and $tx\times hy$ are representable as elements of $R(t-1)$ with the same exponent, their sum is an element of $R(t)$. So, the calculations

$$\begin{aligned} p&=fl(hx\times hy),\\ q&=fl(hx\times ty+tx\times hy),\\ r&=fl(tx\times ty) \end{aligned} \tag{5.8}$$

are exact, because the floating-point operations involved are faithful. We find z and the correction term $z1$ (say) by performing an exact addition of p and q:

$$z=fl(p+q),\quad z1=fl((p-z)+q). \tag{5.9}$$

Since certainly $|p|\geqq|q|$, we have $ep\geqq eq$, in other words, the relation corresponding to (4.5) holds. Hence, $z+z1=p+q$ according to Theorem (4.7) and Corollary (4.15), since we assume that R has the form (5.3) and floating-point addition and subtraction are optimal.

From these assumptions, it also follows that $z1$ is representable as an element of $R(t-1)$ with the same exponent as r. Hence, $z1+r\in R$, so that zz defined by (5.1) is obtained from

$$zz=fl(z1+r). \tag{5.10}$$

So we have proved the following

(5.11) **Theorem.** If R has the form (5.3), floating-point addition and subtraction are optimal and multiplication is faithful, then for all $x, y\in R$ splitted into head and tail according to (5.5) and (5.6), where $t1$ and $t2$ are given by (5.7), the formulas (5.8), (5.9) and (5.10) yield z and zz satisfying (5.1).

Formulas (5.8), (5.9) and (5.10) can be written as the following sequence of ALGOL 60 statements (cf. 4.14):

$$\begin{aligned} &\text{“}p:=hx\times hy;\ q:=hx\times ty+tx\times hy;\\ &z:=p+q;\ zz:=p-z+q+tx\times ty\text{”}. \end{aligned} \tag{5.12}$$

Note that, in the last assignment statement, the additions and subtractions are performed from left to right (see (4.14) and Naur (1962, 3.3.5.)). We use (5.12) in our algorithm for exact multiplication (see ALGOL 60 procedure “*mul12*” in Appendix).

As to the extra condition that “zz be (almost) negligeable within machine precision with respect to $x\times y$”, we shall show for $t\geqq 2$ that

$$|zz|\leqq|x\times y|\,\frac{\tau 2^{-t}}{1+\tau 2^{-t}}, \tag{5.13}$$

where $\tau=2$ if t even and $\tau=3$ otherwise.

Proof. If $x=0$ or $y=0$, then obviously $zz=z=0$ and (5.13) holds. So we may assume that x and y are nonzero, and thus also hx, hy and $p+q$. Let

$$\varepsilon=tx/x,\quad \eta=ty/y,\quad \delta=z1/(p+q).$$

Then a simple calculation yields

$$zz=(x\times y)((1-\varepsilon\eta)\delta+\varepsilon\eta).$$

Since floating-point addition is optimal, we have

$$|\delta|\leqq\frac{2^{-t}}{1+2^{-t}}$$

(cf. Wilkinson (1963, p. 17–19) who gives the bound 2^{-t}). Similarly, it follows from (5.5) and (5.6) that

$$|\varepsilon|,\ |\eta|\leqq\frac{2^{-t2}}{1+2^{-t2}}.$$

Hence

$$|zz|\leqq|x\times y|\,\frac{2^{-t}}{1+2^{-t}}\left[1+\frac{2^{t-2t2}(1+2^{1-t})}{(1+2^{-t2})^2}\right].$$

Since, according to (5.7),

$$2^{t-2t2}=\tau-1$$

and, for $t\geqq 2$,

$$1+2^{-t2}\geqq 1+\tau 2^{-t}\geqq 1+2^{1-t},$$

we obtain

$$|zz|\leqq|x\times y|\,\frac{2^{-t}}{1+2^{-t}}\left[1+\frac{\tau-1}{1+\tau 2^{-t}}\right]$$

from which (5.13) immediately follows.

A slightly smaller bound for zz is obtained if the algorithm is modified as follows. After calculating z and zz as above (see 5.12), an exact addition of z and zz is performed:

$$\text{“}u:=z;\ z:=u+zz;\ zz:=u-z+zz\text{”}. \tag{5.14}$$

Then z and zz still satisfy (5.1) and, since addition is optimal, we now have

$$|zz| \leqq |x \times y| \frac{2^{-t}}{1+2^{-t}}. \tag{5.15}$$

This bound is only about 2 or 3 times smaller than the bound given by (5.13). Therefore we did not include (5.14), requiring three extra additions or subtractions, in our algorithm "*mul12*".

The following algorithm for exact multiplication, due to Veltkamp (1968), also works under the conditions stated in Theorem (5.11).

$$z = fl(x \times y),$$
$$zz = fl((((hx \times hy - z) + hx \times ty) + tx \times hy) + tx \times ty).$$

If multiplication is optimal, then zz satisfies (5.15), otherwise the bound for zz is about twice as large, because multiplication is assumed to be faithful. Veltkamp's algorithm requires one more multiplication and one less addition than ours.

6. Splitting into Halflength Numbers

Let x be a given element of R. We want to calculate hx and tx as defined by (5.5) and (5.6) by means of some simple arithmetical operations. We consider the formula

$$p = fl(x \times c), \quad q = fl(x - p), \quad hx = fl(q + p), \tag{6.1}$$

where c is some constant, and we try to find a value for c such that the formula yields hx defined by (5.5). The most obvious choices $c = \pm 2^{t1}$ do not always work.

For example, if t is even, $c = 2^{t1}$ and $|mx| = 2^{t-1} + 2^{t1-1}$, then $fl(q+p)$ is not an element of $R(t2)$; if $c = -2^{t1}$ and $|mx| = 2^t - 1$, then $fl(q+p)$ is not properly rounded to $t2$ bits.

On the other hand, defining

$$c = 2^{t1} + 1, \tag{6.2}$$

the following theorem holds.

(6.3) **Theorem.** If R has the form (5.3), floating-point addition and subtraction are optimal, multiplication is faithful, and $t1$, $t2$ and c are defined by (5.7) and (6.2), then, for all $x \in R$, formula (6.1) yields hx defined by (5.5).

Proof. If $x = 0$, then the theorem is obvious.

If $x \neq 0$, then we may and shall assume (for definiteness) that the floating-point numbers involved are normalized, i.e. the absolute value of the mantissas are $\geqq 2^{t-1}$. Since addition is optimal and, according to (5.5), hx is an element of $R(t2)$ and, thus, certainly of $R = R(t)$, we need to show only that $q+p$ obtained from (6.1) equals hx. Obviously ep equals either $ex + t1$ or $ex + t1 + 1$ which cases we now consider separately.

a) $ep = ex + t1$. Then obviously eq equals either ep or $ep - 1$. However, $eq = ep - 1$ is impossible, because this would imply $|mq| \geqq 2^t$ which is not within the mantissa range.

Hence $eq = ep$ and the result easily follows.

b) $ep = ex + t1 + 1$. Then obviously eq equals $ep - 1$ or ep. If $eq = ep - 1$, then the result easily follows. If $eq = ep$, then we have

$$|mq| < |mx|/2 + 3/2.$$

This is within the normalized mantissa range $2^{t-1} \leqq |mq| < 2^t$ only if $|mx| = 2^t - \varepsilon$, where ε equals 1 or 2, but then we obtain

$$q + p = round\,(m\,x\,2^{-t1-1})\,2^{ep} = round\,(m\,x\,2^{-t1})\,2^{ep-1}$$

and the result follows.

This completes the proof.

After calculating hx according to (6.1), we obtain tx defined by (5.6) from

$$tx = fl(x - hx) \tag{6.4}$$

which is equivalent to (5.6), because $tx \in R$ and subtraction is optimal. Formulas (6.1) and (6.4) can be written as the following sequence of ALGOL 60 statements, where c is assumed to have the value given by (6.2):

$$\text{"}p := x \times c;\ hx := x - p + p;\ tx := x - hx\text{"}. \tag{6.5}$$

Note that, in ALGOL 60, "$x - p + p$" is equivalent to "$(x - p) + p$" (see (4.14) and Naur (1962, 3.3.5.)). We use formula (6.5) in our algorithm for exact multiplication (see ALGOL procedure "*mul12*" in Appendix).

7. Doublelength Arithmetic

We now give a straightforward application of the results of the previous sections to obtain algorithms for doublelength addition, subtraction, multiplication, division and square root. In the appendix, the algorithms are given as ALGOL 60 procedures. For simplicity, we assume that the conditions of Section 5 are satisfied, viz. R has the form (5.3), floating-point addition and subtraction are optimal, and multiplication is faithful. The algorithms for addition and subtraction, however, would also work correctly under the weaker assumptions stated in Theorems (4.7), (4.9) and (4.10) and Corollaries (4.11), (4.12) and (4.15), provided that no overflow occurs.

In several computers, more than one floating-point system is available, e.g. a "single" and a "double" precision system. Our technique is, of course, most effective if we start from the highest precision system available (provided that it satisfies the requirements). Thus, starting from a "double" precision system and arithmetic, our technique yields a "quadruple" precision system and arithmetic.

We shall, however, call the system R used as starting-point for our technique "*singlelength* floating-point number system", and the arithmetical operations defined on R "*singlelength* floating-point arithmetical operations".

(7.1) **Definition.** A *doublelength* floating-point number is a pair (r, s) of single-length floating-point numbers (i.e. $r, s \in R$) satisfying

$$|s| \leqq |r + s| \frac{2^{-t}}{1 + 2^{-t}}. \tag{7.2}$$

The value of the doublelength number (r, s) is, by definition, equal to $r+s$. We call r the *head* and s the *tail* of (r, s).

In particular, any pair $(r, 0)$ is a doublelength floating-point number, and, since addition is optimal, also any pair (z, zz) obtained by performing an exact addition (see Section 4).

Sometimes we replace (7.2) by the weaker condition

$$|s| \leqq |r+s| C 2^{-t}, \tag{7.3}$$

where C is some constant not much larger than 1; we call a pair (r, s) satisfying (7.3) a *nearly doublelength* floating-point number.

In particular, a pair (z, zz) obtained by exact multiplication (5.12) is a nearly doublelength floating-point number, because, according to (5.13) and Theorem (5.11), we can take $C = \tau/(1+\tau 2^{-t})$.

On the other hand, the magnitude of the tail of a doublelength number may be much smaller than the bound given by (7.2). Thus, the mantissa of a doublelength floating-point number cannot always be represented by means of a multilength integer of fixed maximum length. This is in contrast with the usual approach in which the mantissa is represented by means of some integers.

The doublelength sum of two (nearly) doublelength floating-point numbers (x, xx) and (y, yy) is calculated as follows (see ALGOL 60 procedure "*add2*" in Appendix).

First, the heads x and y are added exactly (4.14). Here the roles of x and y are interchanged when $|x|<|y|$, in order to ensure that (4.5) holds. Thus, we obtain a doublelength number (r, rr) such that $r+rr = x+y$. Subsequently, the tails are added to rr:

$$s = fl((rr+yy)+xx), \tag{7.4}$$

so that $r+s$ approximately equals the sum of (x, xx) and (y, yy). Here again the roles of xx and yy are interchanged when $|x|<|y|$, in order to reduce the maximum error in (7.4) and to ensure commutativity. Finally, an exact addition of r and s is performed. Although not always $|r| \geqq |s|$ (cancellation in forming r may cause $|r|$ to be slightly smaller than $|s|$), the relation corresponding to (4.5) certainly holds, so that, for this final exact addition, we never need interchange the roles of r and s.

Since singlelength addition is optimal, the final exact addition transforms the approximate sum into a doublelength floating-point number having the same value.

Doublelength subtraction is performed in a completely analogous fashion (see ALGOL 60 procedure "*sub2*" in Appendix).

The calculation of doublelength product, quotient and square root is rather obvious and can be sketched as follows (for details see error analysis below and ALGOL 60 procedures "*mul2*", "*div2*" and "*sqrt2*" in Appendix). First a nearly doublelength approximation, (c, cc), of the required result is calculated. Here, besides some singlelength operations, exact multiplication is used. The pair (c, cc) satisfies the relation corresponding to (7.3), but not always (7.2). Therefore, an exact addition is performed, which transforms the result obtained into a doublelength floating-point number having the same value.

Error Analysis. The only error in the doublelength addition is committed in forming s (7.4). Assuming that (x, xx) and (y, yy) are nearly doublelength numbers satisfying

$$|xx| \leqq |x+xx| C_1 2^{-t}, \quad |yy| \leqq |y+yy| C_2 2^{-t}, \tag{7.5}$$

where C_1 and C_2 are constants not much larger than 1, we shall show that, for sufficiently large t, the error, E^+, of the doublelength addition satisfies

$$|E^+| \leqq \{|x+xx|(1+C_1)+|y+yy|(1+C_2)\}2^{1-2t}. \tag{7.6}$$

Proof. From (7.4), it follows that

$$s = (rr+yy)(1+\varepsilon)(1+\varepsilon')+xx(1+\varepsilon'),$$

where $|\varepsilon|$, $|\varepsilon'| \leqq 2^{-t}/(1+2^{-t}) < 2^{-t}$, because addition is optimal, cf. Wilkinson (1963, p. 9). Similarly,

$$|rr| \leqq |x+y| 2^{-t}.$$

From these relations and (7.5), we obtain

$$\begin{aligned} |E^+| &\leqq |rr+yy|(2^{1-t}+2^{-2t})+|xx|2^{-t} \\ &\leqq |x+xx|\{(2^{1-2t}+2^{-3t})(1+C_1 2^{-t})+C_1 2^{-2t}\} \\ &\quad +|y+yy|(2^{1-2t}+2^{-3t})(1+C_2 2^{-t}+C_2). \end{aligned}$$

Since $|y| \leqq |x|$ (otherwise the roles ot x and y would be interchanged), the sum of the terms of order of magnitude 2^{-3t} and lower is smaller than $|x+xx|C_1 2^{-2t}$ for sufficiently large t. This establishes (7.6).

If (x, xx) and (y, yy) are doublelength numbers, then $C_1, C_2 < 1$ so that (7.6) reduces to

$$|E^+| \leqq (|x+xx|+|y+yy|)2^{2-2t}. \tag{7.7}$$

If, however, doublelength addition is used, in combination with our exact multiplication algorithm, for calculating the doublelength scalar product of two vectors of singlelength floating-point numbers, then one of the constants C_1, C_2 is smaller than τ (i.e. 2 if t even and 3 otherwise (5.13)) and the other is smaller than 1. Repeated application of (7.6) then yields an upper bound for the error of the doublelength scalar product, cf. Wilkinson (1963, p. 18).

Formula (7.6) means that the error is small in doublelength precision with respect to the sum of the magnitudes of the terms, the loss being at most a few bits. The relative error of the doublelength sum, however, is not always small, because severe cancellation may take place in forming r. Since this can happen only if $rr = 0$, the algorithm for doublelength addition can easily be modified such that a small relative error is ensured (Veltkamp, 1968). We did not include this modification in our algorithm, because it is of only limited value if the terms are not exact.

The doublelength product, quotient and square root produced by our algorithms all have a small relative error. In other words, if the operands are (nearly) doublelength numbers, then the corresponding absolute error, $E^\times$, $E^/$

and $E^{\sqrt{\ }}$, satisfy

$$|E^{\times}| \leqq (|x+xx| \times |y+yy|)\, C^{\times} 2^{-2t},$$
$$|E^{/}| \leqq (|x+xx|/|y+yy|)\, C^{/} 2^{-2t}, \tag{7.8}$$
$$|E^{\sqrt{\ }}| \leqq \sqrt{(x+xx)}\, C^{\sqrt{\ }} 2^{-2t},$$

where $C^{\times}$, $C^{/}$ and $C^{\sqrt{\ }}$ are constants not much larger than 1. (Here we assume, of course, that the denominator of the quotient is nonzero and the argument of the square root is nonnegative.)

If the operands are doublelength floating-point numbers (7.1) and singlelength division and square root are faithful, then, for $t \geqq 10$ (say), the constants are bounded by

$$C^{\times} \leqq 9+\tau, \quad C^{/} \leqq 21.1, \quad C^{\sqrt{\ }} \leqq 12.7, \tag{7.9}$$

where $\tau = 2$ if t even and $\tau = 3$ otherwise (5.13). If, moreover, singlelength multiplication and division are optimal (which holds for the Philips Electrologica X8 computer), then

$$C^{\times} \leqq 7+\tau, \quad C^{/} \leqq 12.1, \quad C^{\sqrt{\ }} \leqq 10.2. \tag{7.10}$$

Proof. Let

$$P = (x+xx) \times (y+yy), \quad P' = x \times y + x \times yy + xx \times y.$$

Since the operands are doublelength numbers (7.1), we have

$$|P'-P| = |xx \times yy| \leqq |P| 2^{-2t}. \tag{7.11}$$

In "*mul2*" the following approximation, P'', is calculated:

$$P'' = c + fl((x \times yy + xx \times y) + cc),$$

where c and cc, obtained by exactly multiplying x and y, satisfy (cf. Section 5)

$$c + cc = x \times y, \quad |cc| \leqq |x \times y| \frac{\tau 2^{-t}}{1+\tau 2^{-t}}.$$

Hence, we obtain (cf. Wilkinson (1963, p. 7–11))

$$P'' = c + [\{(x \times yy)(1+\varepsilon) + (xx \times y)(1+\varepsilon')\}(1+\varepsilon_1) + cc](1+\varepsilon_2),$$

where

$$|\varepsilon_1|, \ |\varepsilon_2| \leqq 2^{-t}/(1+2^{-t}),$$

because addition is optimal, and, for some constant μ,

$$|\varepsilon|, \ |\varepsilon'| \leqq \mu 2^{-t}/(1+2^{-t}).$$

From the assumption that multiplication is faithful, it follows that $\mu \leqq 2$; if multiplication is optimal, then $\mu = 1$.

From the formulas for P' and P'', we easily derive

$$|P''-P'| \leqq |P|(2\mu+4+\tau) 2^{-2t}. \tag{7.12}$$

So, combining (7.11) and (7.12), we obtain

$$|E^{\times}| = |P''-P| \leqq |P|\,(2\mu+5+\tau) 2^{-2t}. \tag{7.13}$$

This establishes the relations given above for $C^{\times}$.

Let

$$Q=(x+xx)/(y+yy), \quad Q'=(x+xx-x\times yy/y)/y.$$

Since the operands are doublelength numbers (7.1), we have

$$|Q'-Q|\leqq|Q|2^{1-2t}. \tag{7.14}$$

In "*div2*" the following approximation, Q'', is calculated:

$$Q''=c+fl((((x-u)-uu)+xx)-c\times yy)/y),$$

where

$$c=fl(x/y)=(x/y)(1+\varepsilon)\ \text{(say)}, \quad u+uu=c\times y,$$

u and uu being obtained by exactly multiplying c and y. Since cancellation occurs in calculating $fl(x-u)$, this is exact. Moreover, since division is faithful, we have $x-c\times y\in R$, so that

$$fl((x-u)-uu)=x-u-uu=-x\varepsilon.$$

Hence (cf. Wilkinson (1963, p. 7–11)),

$$\begin{aligned}Q''=(x/y)(1+\varepsilon)+[\{(x/y)(-\varepsilon)+xx/y\}(1+\varepsilon_1)\\ -(x/y)(yy/y)(1+\varepsilon)(1+\varepsilon'')](1+\varepsilon_2)(1+\varepsilon'),\end{aligned}$$

where

$$|\varepsilon_1|,\ |\varepsilon_2|\leqq 2^{-t}/(1+2^{-t}),$$

because addition and subtraction are optimal, and, for some constant μ,

$$|\varepsilon|,\ |\varepsilon'|,\ |\varepsilon''|\leqq\mu 2^{-t}.$$

Since we assume that multiplication and division are faithful, we obviously have $\mu<2$; if these operations are optimal, then $\mu<1$. Denoting the sum of the terms of order 2^{-3t} and lower by $\theta 2^{-2t}$, we obtain

$$|Q''-Q'|\leqq Q\ (\mu^2+6\mu+3+\theta)2^{-2t}. \tag{7.15}$$

Further analysis shows that, for $t\geqq 10$, we can take $\theta=0.1$. So, from this and (7.14), we obtain

$$|E'|=|Q''-Q|\leqq|Q|\,(\mu^2+6\mu+5.1)2^{-2t}. \tag{7.16}$$

This establishes the relations given above for C'.

Let

$$R=\sqrt{x+xx}, \quad R'=\sqrt{x}+xx/(2\sqrt{x}).$$

Since the argument is a doublelength number, we have

$$|R'-R|\leqq\tfrac{1}{8}|R|2^{-2t}. \tag{7.17}$$

In a similar way as above, one shows that the approximation R'' calculated in "*sqrt2*" satisfies

$$|R''-R'|\leqq|R|\,(2.5\mu+7.5+\theta)2^{-2t}, \tag{7.18}$$

where $\mu<2$ since division is faithful ($\mu<1$ if division is optimal) and $\theta 2^{-2t}$ again denotes the sum of the terms of order 2^{-3t} and lower.

For $t \geqq 10$, we can take $\theta = 0.075$. So, from this and (7.17), we obtain

$$|E^{\vee}| = |R'' - R| \leqq |R|(2.5\mu + 7.7)2^{-2t}.$$

This establishes the relations given above for $C^{\vee}$ and completes the proof.

The doublelength operations defined by our algorithms "*add2*", "*sub2*", "*mul2*", "*div2*" and "*sqrt2*" are not faithful. To obtain faithful doublelength operations, one would have to calculate the result in triplelength (or maybe even quadruplelength) precision and then to round or truncate it to doublelength. This would, of course, require considerably more work and be preferable only in exceptional situations.

8. Appendix. ALGOL 60 Procedures

In this appendix, we give a set of ALGOL 60 procedures for doublelength arithmetic and exact multiplication.

The procedures work correctly if the singlelength floating-point system is binary, singlelength floating-point addition and subtraction are optimal (3.2), multiplication is faithful (3.1) and no overflow or underflow occurs (cf. Section 5).

The procedures for doublelength addition and subtraction also work correctly in a nonbinary system, provided that a guarding digit is available for the mantissas of anonymous quantities (i.e. quantities which are not assigned to a variable; for details see Section 4).

The procedures have been tested on the Philips Electrologica X8 computer at Mathematical Centre, Amsterdam. This is a binary machine having $t = 40$ binary digits in the mantissa and 12 bits for the exponent (i.e. $D = E = 2048$ in (2.2)).

In particular, the algorithms described in "*add2*" and "*mul12*" have been used extensively for calculating doublelength scalar products of vectors of singlelength floating-point numbers.

Procedure "*add2*" has been tested also on the GE635 computer at Bell Telephone Laboratories, Murray Hill. This is a binary machine having two floating-point systems, viz. single precision $t = 27$ and double precision $t = 63$. Procedure "*add2*" worked correctly in both systems.

In the comments, (x, xx), (y, yy) and (z, zz) denote (nearly) doublelength numbers (see Section 7).

```
comment add2 calculates the doublelength sum of (x, xx) and (y, yy), the result
        being (z, zz);

procedure add2 (x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real r, s;
    r := x + y;
    s := if abs (x) > abs (y) then
        x - r + y + yy + xx else y - r + x + xx + yy;
    z := r + s;
   zz := r - z + s
end add2;
```

```
comment sub2 calculates the doublelength difference of (x, xx) and (y, yy), the
        result being (z, zz);

procedure sub2 (x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real r, s;
      r:=x-y;
      s:=if abs (x)>abs (y) then
          x-r-y-yy+xx else -y-r+x+xx-yy;
      z:=r+s;
     zz:=r-z+s
end sub2;

comment mul12 calculates the exact product of x and y, the result being the
        nearly doublelength number (z, zz). The constant should be chosen
        equal to 2↑(t-t÷2)+1, where t is the number of binary digits in
        the mantissa;

procedure mul12 (x, y, z, zz);
value x, y; real x, y, z, zz;
begin real hx, tx, hy, ty, p, q;
       p:=x×constant;
      hx:=x-p+p; tx:=x-hx;
       p:=y×constant;
      hy:=y-p+p; ty:=y-hy;
       p:=hx×hy;
       q:=hx×ty+tx×hy;
       z:=p+q;
      zz:=p-z+q+tx×ty
end mul12;

comment mul2 calculates the doublelength product of (x, xx) and (y, yy), the
        result being (z, zz);

procedure mul2 (x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real c, cc;
      mul12 (x, y, c, cc);
      cc:=x×yy+xx×y+cc;
       z:=c+cc;
      zz:=c-z+cc
end mul2;

comment div2 calculates the doublelength quotient of (x, xx) and (y, yy), the
        result being (z, zz). If y=0, the effect of this procedure is undefined;
```

```
procedure div2 (x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real c, cc, u, uu;
      c:=x/y;
      mul12 (c, y, u, uu);
      cc:=(x-u-uu+xx-c×yy)/y;
      z:=c+cc;
      zz:=c-z+cc
end div2;
```

comment *sqrt2* calculates the doublelength square root of (x, xx), the result being (y, yy). If (x, xx) is not positive, then the result equals (0, 0);

```
procedure sqrt2 (x, xx, y, yy);
value x, xx; real x, xx, y, yy;
begin real c, cc, u, uu;
      if x>0 then
      begin c:=sqrt (x);
            mul12 (c, c, u, uu);
            cc:=(x-u-uu+xx)×0.5/c;
            y:=c+cc;
            yy:=c-y+cc
      end
      else y:=yy:=0
end sqrt2.
```

Acknowledgements. The author expresses his gratitude to Professor Dr. F. E. J. Kruseman Aretz, formerly at Mathematical Centre, Amsterdam, to Professor Dr. G. W. Veltkamp at Technological University, Eindhoven, and to Dr. W. S. Brown and Dr. P. L. Richman at Bell Telephone Laboratories, Murray Hill, for many valuable suggestions and inspiring discussions. The author is also grateful to Mr. H. J. W. ten Hagen, Mr. H. N. Glorie and Mr. D. T. Winter at Mathematical Centre, Amsterdam, for their assistance in testing the algorithms.

References

Babuška, I.: Numerical stability in mathematical analysis. IFIP congr. 68, Invited papers, 1–13 (1968).

Grau, A. A.: On a floating-point number representation for use with algorithmic languages. Comm. ACM **5**, 160–161 (1962).

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

Knuth, D. E.: The art of computer programming, vol. 2. Addison Wesley (1969).

Møller, O.: Quasi double-precision in floating-point addition. BIT **5**, 37—50 (1965).

Naur, P. (ed.): Revised report on the algorithmic language ALGOL 60 (1962).

Veltkamp, G. W.: Private communications (see also RC Informatie Nr. 21 & 22, Technological University, Eindhoven). (1968).

Wilkinson, J. H.: Rounding errors in algebraic processes. Her Majesty's Stationary Office (1963).

Dr. T. J. Dekker
Mathematical Centre
2 E Boerhaavestraat 49
NL–Amsterdam
Netherlands