Augustin A. Dubrulle

A Class of Numerical Methods for the Computation of Pythagorean Sums

Moler and Morrison have described an iterative algorithm for the computation of the Pythagorean sum $(a^2 + b^2)^{1/2}$ *of two real numbers* a *and* b. *This algorithm is immune to unwarranted floating-point overflows, has a cubic rate of convergence, and is easily transportable. This paper, which shows that the algorithm is essentially Halley's method applied to the computation of square roots, provides a generalization to any order of convergence. Formulas of orders 2 through 9 are illustrated with numerical examples. The generalization keeps the number of floating-point divisions constant and should be particularly useful for computation in high-precision floating-point arithmetic.*

1. Introduction

As in [1], the Pythagorean sum h of two real numbers a and b is defined by

$$h = (a^2 + b^2)^{1/2}.$$

Pythagorean sums are a frequent occurrence in numerical computations, and their evaluation in floating-point arithmetic requires some precautions, often overlooked, to prevent unwarranted overflows or underflows. In their paper, Moler and Morrison [1] describe an elegant iterative algorithm for the computation of Pythagorean sums that is immune to unwarranted overflows, has a cubic rate of convergence, and is easily transportable to any machine.

This paper presents a summary description of the Moler-Morrison algorithm and shows that it is essentially the application of Halley's method [2] to the computation of square roots. A generalization to any convergence order higher than linear is then proposed that preserves the main properties of the Moler-Morrison algorithm. Algorithms for orders 2 through 9 are illustrated with numerical examples. Since the general algorithm is based on a rational iteration, the number of divisions required per iteration is constant (two), while the number of multiplications is proportional to the order of convergence. High-order algorithms should thus be particularly interesting for multiple-precision floating-point computations.

2. The Moler-Morrison algorithm

In a rectangular coordinate system $\{x, y\}$ of origin 0, we consider a sequence of points $\{A_n; n = 0, 1, 2, \cdots\}$, A_0 being in the first quadrant, at a distance h from the origin. A_{n+1} is derived from A_n as follows. Let H_n be the projection of A_n on the x axis and M_n the midpoint of A_nH_n. A_{n+1} is defined as the reflection of A_n in $0M_n$ (Fig. 1). Elementary geometric considerations show that A_{n+1} and A_n are in the same quadrant, at the same distance from the origin, and that A_{n+1} is closer than A_n to the x axis. Thus, from the definition of A_0, the set $\{A_n\}$ is on the quarter circle of radius h in the first quadrant, with A_{n+1} between A_n and A, the point of abscissa h on the x axis. The sequence $\{A_n\}$ converges monotonically towards A.

Let x_n and y_n be the coordinates of A_n. From the above considerations, the sequence $\{x_n\}$ converges monotonically to h from below, with

$$h = (x_n^2 + y_n^2)^{1/2}, \qquad n = 0, 1, 2, \cdots, \tag{1}$$

while the sequence $\{y_n\}$ converges monotonically to zero from above. Thus, $\{x_n\}$ can be considered as a sequence of approximants to the Pythagorean sum of two arbitrary numbers x_0 and y_0. From the relationship between A_n and A_{n+1}, we now derive the iteration formulas providing the pair (x_{n+1}, y_{n+1}) from (x_n, y_n).

Without loss of generality, we henceforth assume that

$$x_0 \geq y_0 > 0,$$

considering that the Pythagorean addition is commutative and that the case $y_0 = 0$ is trivial. This assumption guarantees that

$$x_n \geq y_n > 0 \tag{2}$$

for any finite value of n.

From Eq. (1), we have

$$x_n^2 + y_n^2 = x_{n+1}^2 + y_{n+1}^2 . \tag{3}$$

Since $A_{n+1}A_n$ is perpendicular to $0M_n$,

$$y_n(y_{n+1} - y_n) + 2x_n(x_{n+1} - x_n) = 0. \tag{4}$$

Combining Eqs. (3) and (4), we obtain

$$x_{n+1} = x_n\left(1 + \frac{2y_n^2}{4x_n^2 + y_n^2}\right),$$

$$y_{n+1} = \frac{y_n^3}{4x_n^2 + y_n^2}, \tag{5}$$

under the assumptions in the inequalities (2).

From Eqs. (5), the Moler-Morrison algorithm for the Pythagorean sum of two real numbers a and b,

$$h = (a^2 + b^2)^{1/2},$$

can be expressed as follows:

$$\left.\begin{aligned} x_0 &= \max(|a|, |b|), \\ y_0 &= \min(|a|, |b|), \\ r_n &= (y_n/x_n)^2, \\ s_n &= r_n/(4 + r_n), \\ x_{n+1} &= x_n + 2s_n x_n, \\ y_{n+1} &= s_n y_n, \end{aligned}\right\} n = 0, 1, 2, \cdots. \tag{6}$$

Equations (6) represent the formulation in [1]. For computations in floating-point arithmetic, the iteration is stopped when

$$4 \simeq r_n + 4,$$

where $\simeq$ denotes equality of machine floating-point representations. The monotonic convergence of x_n to h from below and the formulation in terms of r_n exclude the possibility of unwarranted overflows. The asymptotic rate of convergence is cubic, and very few iterations are required to obtain results accurate to the working precisions of current computers. The simplicity, compactness, and accuracy of the algorithm make it an attractive choice for inclusion in mathematical software libraries.

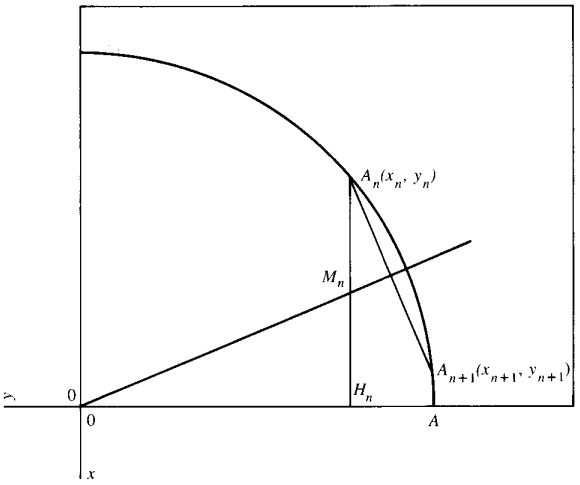

Figure 1 Geometric interpretation of the Moler-Morrison algorithm.

3. Relationship with Halley's method

Let $f(x)$ be a real function twice differentiable of the real variable x. Halley's method (1694) is an iterative scheme for finding an approximation to a root of the equation

$$f(x) = 0$$

given an initial guess x_0 of that root. It is based on the iteration formula

$$x_{n+1} = x_n - \frac{\dfrac{f_n}{f'_n}}{1 - \dfrac{1}{2}\dfrac{f_n f''_n}{(f'_n)^2}}, \tag{7}$$

where f_n, f'_n, and f''_n are the values of $f(x)$ and its first two derivatives at x_n. The asymptotic rate of convergence of the method is cubic for simple roots and linear for multiple roots. More details on this iteration can be found in [2].

We now apply Halley's method to the computation of Pythagorean sums. For a given pair (x_0, y_0) such that

$$0 < y_0 \leq x_0 , \tag{8}$$

the Pythagorean sum

$$h = (x_0^2 + y_0^2)^{1/2} \tag{9}$$

is a root of the equation

$$f(x) = x^2 - h^2 = 0. \tag{10}$$

Replacing $f(x)$ by its expression (10) in formula (7), we obtain

$$x_{n+1} = x_n\left(1 + 2\frac{h^2 - x_n^2}{h^2 + 3x_n^2}\right). \tag{11}$$

From Eq. (11), we can derive that

$$0 \leq x_n \leq h \tag{12}$$

is a sufficient condition for the double inequality

$$x_n \leq x_{n+1} \leq h \tag{13}$$

to hold true. Since x_0 satisfies the inequalities (12) from the definitions (8) and (9), the inequalities (13) guarantee that y_n exists such that

$$y_n^2 = h^2 - x_n^2 . \tag{14}$$

The combination of Eqs. (11) and (14) finally yields

$$x_{n+1} = x_n \left(1 + 2 \frac{y_n^2}{4x_n^2 + y_n^2} \right)$$

and

$$y_{n+1} = \frac{y_n^3}{4x_n^2 + y_n^2},$$

which are precisely the equations (5) derived for the Moler-Morrison algorithm.

4. A generalization

While the Moler-Morrison algorithm is likely to be of optimal efficiency for the working precision of current computers, higher-order algorithms may be desirable (under certain conditions of computational cost) for calculations requiring higher levels of precision. In this section, we propose such algorithms for which the number of divisions is constant and the number of additions and multiplications varies linearly with the rate of convergence (rational iteration formulas).

The cubic rate of convergence of the Moler-Morrison algorithm is revealed by forming

$$h - x_{n+1} = \frac{(h - x_n)^3}{h^2 + 3x_n^2}, \tag{15}$$

from Eq. (11).

An obvious generalization of Eq. (15) to an asymptotic convergence rate k is

$$h - x_{n+1} = \frac{(h - x_n)^k}{F_k(x_n)}, \tag{16}$$

where $F_k(x)$ is a polynomial to be defined. The requirements we choose to impose upon $F_k(x)$ relate to computational considerations and the properties of the Moler-Morrison method; they are

- Monotonic convergence to h from below:

  $0 \leq x_n \leq h$

  is a sufficient condition for

  $x_n \leq x_{n+1} \leq h$

  to hold true;
- Stability for positive x iterates: the coefficients of $F_k(x)$ are positive (sufficient condition);
- Rational computability of the x iterates: x_{n+1} is a rational function of h^2 and x_n;
- Compatibility with the Moler-Morrison algorithm: Eqs. (15) and (16) coincide when $k = 3$.

The choice

$$F_k(x) = \frac{(h + x)^k + (h - x)^k}{2h}$$

satisfies the above conditions and defines the following iteration formulas:

$$x_{n+1} = h \frac{(h + x_n)^k - (h - x_n)^k}{(h + x_n)^k + (h - x_n)^k},$$

$$y_{n+1} = \frac{2hy_n^k}{(h + x_n)^k + (h - x_n)^k}, \tag{17}$$

with

$$x_{n+1}^2 + y_{n+1}^2 = h^2. \tag{18}$$

An analysis of Eqs. (17) shows that two cases must be considered, based on the parity of k:

- k is even; then
  The numerator of x_{n+1} is an odd polynomial in x_n and an even polynomial in h. The denominator is an even polynomial in x_n and h. Thus, x_{n+1} is the product of x_n and a rational function of x_n^2 and h^2; using a similar approach, we find y_{n+1} to be the product of h and a rational function of x_n^2 and h^2;
- k is odd; then
  x_{n+1} is again the product of x_n and a rational function of x_n^2 and h^2, while y_{n+1} is the product of y_n and a rational function of x_n^2 and h^2.

From these considerations, we see that a rational iteration mapping the pair (x_n, y_n) into the pair (x_{n+1}, y_{n+1}) exists only when k is odd, since h is not a quantity available for computation. Similar analyses show that

$$r_{n+1} = (y_{n+1}/x_{n+1})^2$$

always has a rational expression in terms of r_n and h^2. When k is even, an iteration can thus be derived that maps the pair (x_n, r_n) into (x_{n+1}, r_{n+1}). These two classes of iterations, together with Eqs. (17), constitute the basis for our generalization of the Moler-Morrison algorithm. We derive in the next sections the corresponding iteration formulas. As much as possible, the outline of the general algorithms follows that of the cubic algorithm defined by Eqs. (6).

5. Formulas of even order

For an even rate of convergence,

$$k = 2m,$$

let us first look at the derivation of the x iterate from the first of Eqs. (17). Using the formula for binomial expansion, we have

$$\begin{aligned}(h + x)^{2m} + (h - x)^{2m} &= 2\sum_{i=0}^{m}\binom{2m}{2i}h^{2i}x^{2m-2i},\\ (h + x)^{2m} - (h - x)^{2m} &= 2\sum_{i=0}^{m}\binom{2m}{2i-1}h^{2i-1}x^{2m-2i+1},\end{aligned} \tag{19}$$

with the convention

$$\binom{u}{v} = 0 \quad \text{when} \quad v < 0.$$

As in Section 2, Eq. (6), we define

$$r_n = (y_n/x_n)^2$$

which with a transformation of Eq. (18) gives

$$h^2 = x_n^2(1 + r_n). \tag{20}$$

Combining Eqs. (19), (6), and (20), we get

$$\begin{aligned}(h + x_n)^{2m} + (h - x_n)^{2m} &= 2x_n^{2m}\sum_{i=0}^{m}\binom{2m}{2i}(1 + r_n)^i,\\ h[(h + x)^{2m} - (h - x)^{2m}] &= 2x_n^{2m+1}\sum_{i=0}^{m}\binom{2m}{2i-1}(1 + r_n)^i.\end{aligned} \tag{21}$$

The replacement of the expressions (21) in the first of Eqs. (17) yields

$$x_{n+1} = x_n\frac{\sum_{i=0}^{m}\binom{2m}{2i-1}(1 + r_n)^i}{\sum_{i=0}^{m}\binom{2m}{2i}(1 + r_n)^i}$$

or

$$x_{n+1} = x_n + x_n\frac{\sum_{i=0}^{m}\left[\binom{2m}{2i-1} - \binom{2m}{2i}\right](1 + r_n)^i}{\sum_{i=0}^{m}\binom{2m}{2i}(1 + r_n)^i} \tag{22}$$

for a formula analogous in form to that of the cubic algorithm. Expressing the fraction in Eq. (22) as a rational function of r_n after replacement of the binomials by their expressions, we finally obtain

$$x_{n+1} = x_n + x_n\frac{\sum_{p=1}^{m}r_n^p\sum_{i=p}^{m}\binom{i}{p}\left[\binom{2m}{2i-1} - \binom{2m}{2i}\right]}{\sum_{p=0}^{m}r_n^p\sum_{i=p}^{m}\binom{i}{p}\binom{2m}{2i}}. \tag{23}$$

Note that the numerator of this rational function has a null constant term and that the denominator is monic. Similarly, starting with the equation

$$r_{n+1}\left[\frac{2y_n^k}{(h + x_n)^k - (h - x_n)^k}\right]^2$$

obtained from Eqs. (17), we derive the iteration formula

$$r_{n+1} = (1 + r_n)\left[\frac{r_n^m}{\sum_{p=0}^{m}r^p\sum_{i=p}^{m}\binom{i}{p}\binom{2m}{2i-1}}\right]^2. \tag{24}$$

Assuming that the coefficients

$$\alpha_p^{(2m)} = \sum_{i=p}^{m}\binom{i}{p}\left[\binom{2m}{2i-1} - \binom{2m}{2i}\right], \qquad p = 1, \cdots, m,$$

and

$$\beta_p^{(2m)} = \sum_{i=p}^{m}\binom{i}{p}\binom{2m}{2i}, \qquad p = 0, \cdots, m,$$

are available, the numerical algorithm for the computation of $(a^2 + b^2)^{1/2}$ can be expressed as follows:

$$\left.\begin{aligned}x_0 &= \max(|a|, |b|),\\ r_0 &= [\min(|a|, |b|)/x_0]^2,\\ P_n^{(2m)} &= \sum_{p=1}^{m}\alpha_p^{(2m)}r_n^p,\\ Q_n^{(2m)} &= \sum_{p=0}^{m}\beta_p^{(2m)}r_n^p,\\ x_{n+1} &= x_n + \frac{P_n^{(2m)}}{Q_n^{(2m)}}x_n,\\ r_{n+1} &= (1 + r_n)\left[\frac{r_n^m}{P_n^{(2m)} + Q_n^{(2m)}}\right]^2.\end{aligned}\right\} \quad n = 0, 1, 2, \cdots, \tag{25}$$

The iteration is stopped when the equality of machine representations

$$\text{fl}(1) = \text{fl}(1 + r_n)$$

is satisfied.

One iteration of this algorithm requires the following amount of floating-point arithmetic:

$(2m + 2)$ additions, $(3m + 1)$ multiplications, and 2 divisions.

6. Formulas of odd order

The analysis of Section 4 showed that when k is odd,

$$k = 2m + 1,$$

we have an iteration on the pair (x_n, y_n) that we formulate below as a straight generalization of the Moler-Morrison algorithm. Using an approach similar to that of the previous section, we derive our iteration formulas from Eqs. (17). Defining first the coefficients

$$\alpha_p^{(2m+1)} = \sum_{i=p+1}^{m} \binom{i}{p+1}\left[\binom{2m+1}{2i} - \binom{2m+1}{2i+1}\right], \qquad p = 0, \cdots, m-1,$$

and

$$\beta_p^{(2m+1)} = \sum_{i=p}^{m} \binom{i}{p}\binom{2m+1}{2i+1}, \qquad p = 0, \cdots, m,$$

we express as follows the algorithm for the Pythagorean sum $(a^2 + b^2)^{1/2}$:

$$\left.\begin{aligned} x_0 &= \max(|a|, |b|), \\ y_0 &= \min(|a|, |b|), \\ r_n &= (x_n/y_n)^2, \\ P_n^{(2m+1)} &= \sum_{p=0}^{m-1} \alpha_p^{(2m+1)} r_n^p, \\ S_n^{(2m+1)} &= r_n\left[\sum_{p=0}^{m} \beta_p^{(2m+1)} r_n^p\right]^{-1}, \\ x_{n+1} &= x_n + S_n^{(2m+1)} P_n^{(2m+1)} x_n, \\ y_{n+1} &= r_n^{m-1} S_n^{(2m+1)} y_n. \end{aligned}\right\} n = 0, 1, 2, \cdots. \quad (26)$$

The iteration is stopped when the equality of machine representations

$$\text{fl}(1) = \text{fl}(1 + r_n)$$

is satisfied, except for the case of the cubic iteration ($m = 1$), where the criterion

$$\text{fl}(4) = \text{fl}(4 + r)$$

is more economical and as effective.

One iteration of this algorithm requires the following amount of floating-point arithmetic:

$(2m + 1)$ additions, $(3m + 1)$ multiplications, and 2 divisions.

(with one less addition for the cubic iteration). We can see that the odd-order iterations are more efficient than their even-order counterparts.

The formula of order 1 is of no interest at all. It is a limit case of linear convergence for which

$$x_{n+1} = x_n,$$

from Eqs. (17).

7. An estimate of the maximum number of iterations

Let ε_n denote the relative distance of the nth iterate to the Pythagorean sum for the formula of order k:

$$\varepsilon_n = 1 - \frac{x_n}{h}. \quad (27)$$

Defining

$$u_n = \frac{\frac{\varepsilon_n}{2}}{1 - \frac{\varepsilon_n}{2}}, \quad (28)$$

we obtain, from Eqs. (17) and (27),

$$u_{n+1} = u_n^k,$$

that is,

$$u_n = u_0^{k^n} \quad \text{and} \quad \frac{\varepsilon_n}{2} = \frac{u_0^{k^n}}{1 + u_0^{k^n}}. \quad (29)$$

We now assume that the algorithm is used with a machine where the floating-point number representation has t digits in base β, and for which the relative error in number representation is bounded by machine precision,

$$\varepsilon = \gamma\beta^{1-t}, \quad (30)$$

where γ is a rounding parameter taking the values 1 (chopped representation) or $\frac{1}{2}$ (rounded representation). Ignoring rounding errors, the iteration of order k ceases to be effective as soon as

$$\varepsilon_n < \varepsilon$$

or, from Eq. (29),

$$\frac{u_0^{k^n}}{1 + u_0^{k^n}} < \frac{\varepsilon}{2}.$$

This condition is satisfied *a fortiori* when

$$u_0^{k^n} < \frac{\varepsilon}{2}. \quad (31)$$

The use of this simpler criterion finds its justification in the consideration that

$$\text{fl}(1 + u_0^{k^n}) = \text{fl}(1)$$

when

$$u_0^{k^n} < \varepsilon,$$

where fl (.) denotes number floating-point representation in our machine.

We can now derive the least upper bound N of the number of effective iterations of order k.

The maximum value of ε_0, obtained for the Pythagorean sum of two equal numbers,

$$\varepsilon_0 = 1 - \frac{\sqrt{2}}{2},$$

defines the maximum of u_0 as

$$u_0 = \frac{\sqrt{2} - 1}{\sqrt{2} + 1} \tag{32}$$

from Eq. (28).

Combining Eqs. (30), (31), and (32), N is the smallest value of n for which

$$\left(\frac{\sqrt{2} - 1}{\sqrt{2} + 1}\right)^{k^n} < \frac{1}{2}\gamma\beta^{1-t},$$

that is,

$$N = \text{ceiling}\left[\frac{1}{\log k}\log\frac{\log 2 - \log\gamma + (t-1)\log\beta}{\log\dfrac{\sqrt{2}+1}{\sqrt{2}-1}}\right], \tag{33}$$

where ceiling (w) denotes the smallest integer not less than w.

We now give two examples based on IBM System/370 arithmetic, short and long precision.

In short precision,

$p = 16,\ t = 6,\ \gamma = 1,$

and we obtain, from Eq. (33),

$N = 4$ for $k = 2,$

$N = 2$ for $3 \le k \le 8,$ and

$N = 1$ for $k = 9.$

In long precision,

$\beta = 16,\ t = 14,\ \gamma = 1,$

we get

$N = 5$ for $k = 2,$

$N = 3$ for $3 \le k \le 4,$ and

$N = 2$ for $5 \le k < 9.$

These estimates were verified by numerical experiments.

8. Robustness and transportability

While the monotonic convergence of these algorithms guarantees that no unwarranted floating-point overflow can occur in the course of the computation, underflows can have an undesirable effect as illustrated below in the case of the cubic formula.

Assume that the Pythagorean sum

$$h = (u^2 + u^2)^{1/2} = u\sqrt{2}$$

is computed with the cubic iteration (6), u being the machine underflow threshold, that is, the smallest positive machine floating-point number.

We have

$r_0 = 1$ and $s_0 = 1/5.$

The computation of the first iterate reduces to

$$x_1 = u + 2s_0u.$$

Since

$2s_0 < 1,$

we have

$\text{fl}\,(2s_0u) = 0$

and the first x iterate is u. For the same reason, the first y iterate is zero, and the iteration terminates with the incorrect result u. The same phenomenon occurs for all our formulas where the iteration takes the form

$$x_{n+1} = x_n + \phi_n x_n \tag{34}$$

because ϕ_n is bounded above by $\frac{1}{2}$. This is easily proved by considering the first of Eqs. (17),

$$x_{n+1} = h\frac{(h + x_n)^k - (h - x_n)^k}{(h + x_n)^k + (h - x_n)^k}$$

and the identity

$$h = (1 + r_n)^{1/2}x_n,$$

which yield the inequalities

$$0 \le x_{n+1} - x_n \le [(1 + r_n)^{1/2} - 1]\,x_n \le \frac{1}{2}r_nx_n,$$

or

$$\phi_n \le \frac{1}{2}r_n \le \frac{1}{2}.$$

One obvious approach to remedy this defect is to perform a systematic scaling of a and b by a power of β, the base of the machine number representation, to bring x_0 into the range $(0, \beta)$, and to apply the corresponding inverse scaling to the result. This requires knowledge of β and extra operations in all cases.

Table 1 Formulas of even orders, iteration (25).

k	P and Q
2	$P = r$ $Q = 2 + r$
4	$P = 4r + 3r^2$ $Q = 8 + 8r + r^2$
6	$P = 16r + 20r^2 + 5r^3$ $Q = 32 + 48r + 18r^2 + r^3$
8	$P = 64r + 112r^2 + 56r^3 + 7r^4$ $Q = 128 + 256r + 160r^2 + 32r^3 + r^4$

Table 2 Formulas of odd orders, iteration (26).

k	P and S
3	$P = 2$ $S = r/(4 + r)$
5	$P = 8 + 4r$ $S = r/(16 + 12r + r^2)$
7	$P = 32 + 32r + 6r^2$ $S = r/(64 + 80r + 24r^2 + r^3)$
9	$P = 128 + 192r + 80r^2 + 8r^3$ $S = r/(256 + 448r + 240r^2 + 40r^3 + r^4)$

Table 3 Pythagorean sum of 119 and 120.

Order	*Iterates*
2	120.0000000000000 159.5549451828402 168.7209057465608 168.9997691646582 168.9999999998423 169.0000000000000
3	120.0000000000000 167.3605440280932 168.9999608618056 169.0000000000000
4	120.0000000000000 168.7209057465608 168.9999999998424 169.0000000000000
5	120.0000000000000 168.9526470501203 169.0000000000000
6	120.0000000000000 168.9919703649560 169.0000000000000
7	120.0000000000000 168.9986385471298 169.0000000000000
8	120.0000000000000 168.9997691646582 169.0000000000000
9	120.0000000000000 168.9999608618056 169.0000000000000

A preferred alternative, requiring knowledge of the underflow threshold u and machine precision ε, consists of applying some scaling to the data only when warranted.

In Eq. (34), we observe that, in the absence of underflow, x_{n+1} is computed accurately to machine precision if

$$\phi_n \geq \varepsilon.$$

Thus, the computation terminates correctly if

$$\phi_n x_n \geq \varepsilon x_n \geq u,$$

that is, when

$$x_n \geq u\varepsilon^{-1}.$$

Because x_n is an increasing function of n, it will be sufficient to satisfy the condition

$$x_0 \geq u\varepsilon^{-1}.$$

These considerations define the following scaling rule:

- If $\max(|a|, |b|) < u\varepsilon^{-1}$, multiply a and b by ε^{-1};
- Accordingly, multiply the result by ε.

For example, long-precision computation with an IBM System/370 defines

$$\varepsilon = 16^{-13} \text{ and } u = 16^{65}$$

and the scaling condition

$$\max(|a|, |b|) < 16^{-52}$$

for a factor

$$\varepsilon^{-1} = 16^{13}.$$

It must be noted that any reasonable integer power of β not less than ε^{-1} can be used as a scaling factor and that values higher than $u\varepsilon^{-1}$ can replace the threshold of applicability with little loss of efficiency in order to achieve portability across a wide range of computers.

9. Examples

The iteration formulas of orders 2 through 9 are represented in Tables 1 and 2 by the functions P, Q, and S that appear in iterations (25) and (26) (the iteration subscripts and the order superscripts are dropped for simplicity of notation).

Two examples of application of these formulas are displayed in Tables 3 and 4, obtained with an IBM System/370 in long-precision arithmetic.

10. Conclusion

While the efficiency of the Moler-Morrison algorithm may be optimum for most current computers, higher-order formulas can be useful for systems with slow division. They find an obvious use in computations involving multiple-precision software, where division can be particularly expensive. By reducing the number of iterations, they are also advantageous for implementation with interpretive high-level languages (e.g., APL).

11. Acknowledgment

The author is indebted to Cleve Moler for the communication of the cubic algorithm and its geometric derivation.

12. References

1. C. Moler and D. Morrison, "Replacing Square Roots by Pythagorean Sums," *IBM J. Res. Develop.* **27,** 577–581 (1983, this issue).
2. J. F. Traub, *Iterative Method for the Solution of Equations*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1964.

Received June 2, 1983

Table 4 Pythagorean sum of 19 and 180.

Order	*Iterates*
2	180.0000000000000
	180.9972222648517
	180.9999999786853
	181.0000000000000
3	180.0000000000000
	180.9999923053839
	181.0000000000000
4	180.0000000000000
	180.9999999786853
	181.0000000000000
5	180.0000000000000
	180.9999999999410
	181.0000000000000
6	180.0000000000000
	180.9999999999998
	181.0000000000000
7	180.0000000000000
	181.0000000000000
8	180.0000000000000
	181.0000000000000
9	180.0000000000000
	181.0000000000000

Augustin A. Dubrulle *IBM Scientific Center, 1530 Page Mill Road, Palo Alto, California 94304.* Mr. Dubrulle is currently working on numerical problems of seismic oil exploration. His education includes degrees in mathematics (MS, 1958) and physics (MS, 1959) from the University of Lille, France. Before joining the Palo Alto Scientific Center in 1977, he worked as a numerical analyst for IBM France (1962–1967), in the Data Processing Division (1968–1974), and in the General Products Division (1975–1976). His main interests are in numerical linear algebra, functional approximation, floating point arithmetic, and mathematical software. Mr. Dubrulle is a member of the Association for Computing Machinery, the Society for Industrial and Applied Mathematics, and the Special Interest Group on Numerical Mathematics.