A Note on Complex Division

G. W. STEWART
University of Maryland

An algorithm (Smith, 1962) for computing the quotient of two complex numbers is modified to make it more robust in the presence of underflows.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software—*reliability and robustness*

General terms: Algorithms

Additional Key Words and Phrases: Computer arithmetic, complex division, exponent exception

This note concerns the computation of the complex quotient

$$z \equiv a + bi = \frac{c + di}{e + fi} \qquad (ef \neq 0)$$

in floating point arithmetic. An algorithm, due to R. L. Smith [2], is based on the identities

$$a + bi = \frac{c + d(fe^{-1})}{e + f(fe^{-1})} + \frac{d - c(fe^{-1})}{e + f(fe^{-1})}\, i \qquad (|e| \geq |f|) \tag{1}$$

and

$$a + bi = \frac{d + c(ef^{-1})}{f + e(ef^{-1})} - \frac{c - d(ef^{-1})}{f + e(ef^{-1})}\, i \qquad (|e| \leq |f|). \tag{2}$$

If the operations are performed in the order indicated by the parentheses, the resulting algorithm is remarkably robust in the presence of exponent exceptions, provided underflows are denormalized. Specifically, an analysis of Hough cited by Coonen [1] shows that when the algorithm works, it returns a computed value $\bar{z}$ satisfying

$$|\bar{z} - z| \leq \epsilon |z|, \tag{3}$$

where ϵ is of the same order of magnitude as the rounding unit for the arithmetic

This work was supported in part by the Air Force Office of Scientific Research under Contract AFOSR-82-0078.
Author's address: Department of Computer Science, Institute for Physical Science and Technology, University of Maryland, College Park, MD 20742.

in question. Moreover, the algorithm works for virtually all problems in which the numerator, denominator, and quotient are representable as normalized floating point numbers.[1,2] Thus the algorithm returns something that is almost a "correctly rounded" answer in the sense of (3).

However, $\bar{z}$ being accurate in the sense of (3) does not insure the accuracy of its real and imaginary components. There are two sources of inaccuracy. The first is cancellation of digits in the computation of sums like $c + d(de^{-1})$. There is not much that can be done about this source of error except to compute in higher precision.

The second source of error is the underflow of quantities like fe^{-1} to zero. How this causes inaccuracies may be illustrated by the quotient

$$\frac{10^{70} + 10^{-70}i}{10^{56} + 10^{-56}i} = 10^{14} - 10^{-98}i, \tag{4}$$

(the real and imaginary components of the right hand side are accurate to more than ten decimal digits). If the formula (1) is used to compute the imaginary part of (4) in ten digit decimal arithmetic with an exponent range of ±99, the result is

$$\begin{aligned} t_1 &= fl(fe^{-1}) = 0 \\ t_2 &= fl(ft_1) = 0 \\ t_3 &= fl(e + t_2) = 10^{56} \\ t_4 &= fl(ct_1) = 0 \\ t_5 &= fl(d - t_4) = 10^{-56} \\ b &= fl(t_5/t_3) = 0. \end{aligned} \tag{5}$$

Thus b has a relative error of one. The problem is that in computing the quantity $t_4 = c(fe^{-1}) = 10^{-42}$ the product fe^{-1} underflows to zero, which results in a spurious value of zero for t_4, even though this is the most significant part of the sum $t_5 = d - t_4$.

Note that if t_4 had been computed in the order $(cf)e^{-1}$, then a correct answer would have resulted. This suggests that we attempt to order the calculations of expressions like cfe^{-1} so that whenever the result is representable, no overflows or underflows occur. That this can be done is a consequence of the following observation.

PROPOSITION. *Let $x_1, x_2, \ldots, x_n > 0$ be representable numbers and suppose that $\pi = x_1 x_2 \cdots x_n$ is also representable. Then $max\{x_i\} \cdot min\{x_i\}$ is also representable.*

PROOF. Without loss of generality assume that $x_1 = \max\{x_i\}$ and $x_2 = \min\{x_i\}$. If x_1x_2 overflows, then $x_2 > 1$. Since $x_3, x_4, \ldots, x_n \geq x_2 > 1$ the product is not less than x_1x_2 and also overflows. On the other hand if x_1x_2 underflows, then $x_1 < 1$. Since $x_3, x_4, \ldots, x_n \leq x_1 < 1$, the product π is not greater than x_1x_2 and also underflows.

[1] We shall use the tem "representable" in this sense throughout the note.

[2] The algorithm also works well when underflows are set to zero, provided one avoids numbers whose magnitudes will underflow when multiplied by the rounding unit.

```
flip := false;
if |f| ≥ |e| then
    e :=: f;
    c :=: d;
    flip := true;
end if;
s := 1/e;
t := 1/(e + f*(f*s));
if |f| ≥ |s| then
    f :=: s;
end if;
if |d| ≥ |s| then
    a := t*(c + s*(d*f));
elseif |d| ≥ |f| then
    a := t*(c + d*(s*f));
else
    a := t*(c + f*(s*d));
end if;
if |c| ≥ |s| then
    b := t*(d + s*(c*f));
elseif |c| ≥ |f| then
    b := t*(d + c*(s*f));
else
    b := t*(d + f*(s*c));
end if
if flip then b := -b; fi;
```

Fig. 1. Computation of $a + bi = (c + di)/(e + fi)$.

If we write π in the form

$$\pi = (x_1x_2)x_3x_4 \cdots x_n, \tag{6}$$

then π is exhibited as the product of $n - 1$ representable numbers, which can then be computed by a recursive application of the proposition. Thus a general algorithm for computing the product of a collection of numbers is to replace the largest and smallest numbers in the collection by their product, repeating the process until there is only one number.

When n is large, the algorithm sketched above involves many comparisons and is probably impractical for most applications. In our application, however, where $n = 3$, the technique is workable. The resulting algorithm for complex division is exhibited in Figure 1. Some simplification in the code is obtained by observing that, except for the sign of b, the formula (2) may be obtained from (1) by making the interchanges c :=: d and e :=: f. The common denominator t^{-1} of the terms in (1) may be computed as it appears there, since fe^{-1} is the product of the greatest and the least of the numbers f, f, and e^{-1}. The rest of the algorithm is based on a straightforward implementation of the above technique for computing a product.

This algorithm is more complicated than Smith's, and it requires some additional comparisons and arithmetic operations. Is it worth it? On the one hand, the output of Smith's algorithm, which satisfies (3), is sufficient for the vast majority of applications. Moreover, our algorithm is by no means foolproof; it can fail on machines whose exponent is biased in such a way that the reciprocals of small representable numbers can overflow. On the other hand, the algorithm presented here is not that much more expensive and provides an additional degree of protection against exponent exceptions. That might be enough to convince a person of cautious temperament, especially since division is usually performed less frequently than additions and multiplications.

REFERENCES

1. COONEN, J. T. Underflow and denormalized numbers. *Comput. 13*, (1980), 68–79.
2. SMITH, R. L. Algorithm 116: Complex division. *Commun. ACM 5*, 8 (1962), 435.

Received November 1982; revised February 1984 and March 1984; accepted May 1985.