Rational Arithmetic for Minicomputers

BERTHOLD K. P. HORN

Artificial Intelligence Laboratory, Massachusetts Institute of Technology,
545 Technology Square, Cambridge, Mass. 02139 U.S.A.

SUMMARY

A representation for numbers using two computer words is discussed, where the value represented is the ratio of the corresponding integers. This allows for better dynamic range and relative accuracy than single-precision fixed point, yet is less costly than floating point arithmetic. The scheme is easy to implement and particularly well suited for minicomputer applications that call for a great deal of numerical computation. The techniques described have been used to implement a mathematical function subroutine package for a mini-computer as well as a number of applications programs in the machine vision and machine manipulation area.

KEY WORDS   Rational arithmetic   Minicomputers   Number representation

INTRODUCTION

Typically, minicomputers come equipped with arithmetic instructions that operate on single words. Frequently double-word addition and subtraction is also provided for. Unfortunately, integers are inconvenient for many calculations and require the programmer (or at least the compiler) to remember scale factors relating the number represented to the integer actually stored. Even with scale factors one frequently runs out of dynamic range or has difficulty because of the need to estimate the approximate magnitude of variables ahead of time. Floating-point representation is the obvious, but costly answer. Software to perform the common floating-point operations requires substantial amounts of memory and is usually very slow, while hardware that carries out the same operations—if available— is expensive.

RATIONAL ARITHMETIC

An oft-overlooked alternative that lies between fixed-point and floating-point arithmetic in complexity, cost and capability is rational arithmetic (see Figure 1). Variables are stored in two words and the number represented is the ratio of the integers in these words. This allows a fair dynamic range and possibly higher precision than single-word integer arithmetic, without the complexity of extracting exponents and mantissas as required for floating-point arithmetic. It is also easy to provide software that performs the usual arithmetic operations at relatively high speed.

Since the number of instructions required is relatively small, one may elect to compile them in line instead of providing them as subroutine calls. Similarly, it should be easy to microprogram a machine to provide these operations directly as part of its instruction repertoire. Notice that we are not discussing *exact* rational arithmetic which requires the

BERTHOLD K. P. HORN

COMPUTER REPRESENTATION	MATHEMATICAL ABSTRACTION
FIXED POINT	INTEGERS
'NO POINT'	RATIONALS
FLOATING POINT	REAL NUMBERS

Figure 1. *Analogy between mathematical number systems and computer representations. The proposed 'no point' representation corresponds to the rational numbers*

ability to represent two integers of arbitrary size.[1,2] Our two-word representations for numbers will be approximations in the same way that floating-point numbers are approximations to real numbers.

The minimal capabilities required to implement rational arithmetic in this sense are single-word multiplications (with double-word results), double-word addition, subtraction and shifting. It is helpful to have four registers available to hold partial results. Addition, subtraction, multiplication or division of rational numbers produces a pair of double-word results which have to be 'compressed' into two single words before they can be used in further calculations.

NORMALIZATION

The main decision to be made when implementing rational arithmetic is the choice of an appropriate 'normalization' procedure, which will find a rational number that can be represented in the two-word format, and which approximates the double-size (four-word) result of the elementary arithmetic operations. The fastest method is certainly one where both double-word results are simply shifted right until each fits into one word. It must be pointed out that this does not usually result in the best approximation possible in the two-word format. In fact, it is fairly easy to see that the relative accuracy of this method is no better than that obtained in single-precision integer arithmetic. Its simplicity however makes this scheme very attractive in situations where the extended dynamic range and not the higher accuracy is of most interest.

Next, we notice that one can divide both terms of the result by any common factors they might have (provided the computer has a divide instruction of course). With luck this will lead to sufficient reduction in the size of the results so that they will now fit into the two-word format. In this case the answer is exact, but some work has to be done to find the greatest common divisor in the first place (see Figure 2). The number of iterations in the GCD algorithm is approximately equal to the logarithm of the numbers.[2] One certainly cannot rely on this method alone always to provide the required compression of the two double-word results. This *is* the appropriate method however for *exact* rational arithmetic.[1,2]

GCD(N, M): IF M = 0, THEN N
ELSE GCD(M, REMAINDER(N, M))

Figure 2. *A simple algorithm for finding the greatest common divisor of two numbers. This algorithm is a slight modification of Euclid's, and more elaborate, faster algorithms exist. The function REMAINDER determines the remainder of the indicated division. That is, $REMAINDER(N, M) = N - \lfloor N/M \rfloor * M$. The GCD algorithm is useful for normalization after arithmetic operations or for simplifying terms before they enter into arithmetic. The tail-end recursion can of course be turned into a more efficient iteration*

The Euclidian algorithm for finding the greatest common divisor suggests a related method for finding optimal approximants using number theoretic ideas. The method is

based on an algorithm for finding successive convergents of a continued fraction expansion (see Figure 3). The method is described in detail in Reference 3 and reproduced in

$$\text{FRACTS}(S, N): \quad \text{LET } P_1 \leftarrow 0; Q_1 \leftarrow 1; P_2 \leftarrow 1; Q_2 \leftarrow 0; I \leftarrow 0$$

DO UNTIL $P_2 > N$ OR $Q_2 > N$ OR $S = I$

LET $P_0 \leftarrow P_1; Q_0 \leftarrow Q_1; P_1 \leftarrow P_2; Q_1 \leftarrow Q_2$

LET I ← INTEGER-PART(S)

LET $P_2 \leftarrow P_0 + I * P_1; Q_2 \leftarrow Q_0 + I * Q_1$

IF $S \neq I$, THEN LET $S \leftarrow 1/(S-1)$,

ELSE LET $P_1 \leftarrow P_2; Q_1 \leftarrow Q_2$

END

RETURN (P_1/Q_1)

END

Figure 3. A simple algorithm for determining good rational approximations to a given number S. The successive approximants (P_2/Q_2) are alternately larger than S and smaller than S. The algorithm terminates when either numerator or denominator exceeds the given bound N, or when the ratio exactly equals the number S. The algorithm is based on a method for finding continued fraction expansions

Reference 4. This technique involves more computation, but since a result of the form p/q is accurate to within $1/q^2$, we are now dealing with a method that has similar relative accuracy as one might expect from double-precision integer arithmetic (see Figure 4).
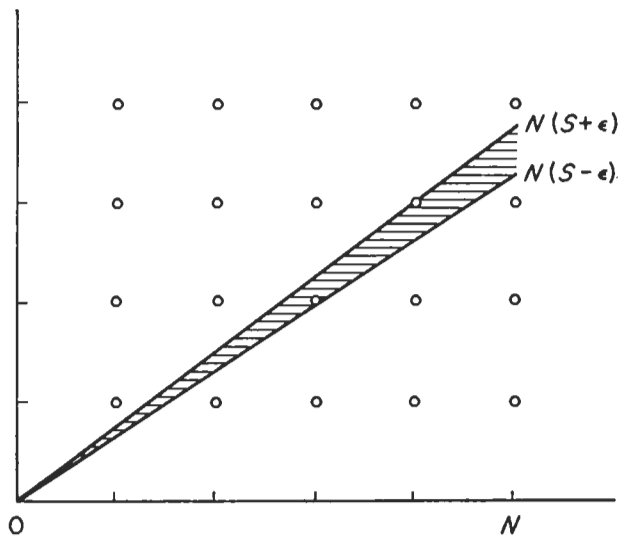

Figure 4. Illustration of the theorem about convergents. The grid points correspond to rational numbers. Those falling within the shaded triangle represent approximations to the number S. These rational numbers have a denominator less than N and lie between $S - \varepsilon$ and $S + \varepsilon$ in value. If the area of the triangle, εN^2 exceeds one, we expect to find one or more grid-points within its boundaries, since there is one grid-point per unit area. This suggests that a number S can be approximated with accuracy $1/N^2$ with rational numbers that have denominator less than or equal to N

The largest approximant that fits into the two-word format has been called the *mediant conversion* value.[4] Fortunately, the algorithm produces the approximants in order of increasing size, and so one simply keeps going until one of the two factors no longer fits into a single word. See Figure 5 for examples of small approximants for common mathematical constants.

RATIONAL APPROXIMATION	MATHEMATICAL CONSTANT
355/113	π
1264/465	e
239/169	$\sqrt{2}$
228/395	γ
192/277	LN 2

Figure 5. Rational approximants for some common mathematical constants, accurate to at least 16 bits or about 5 decimal digits

We have presented here two extreme case algorithms for the normalization, the one exceedingly fast, but of limited accuracy, the other complicated, but with exceptional accuracy. Other points on the spectrum of possibilities can be explored. For example, a compromise acceptable for many real applications is a technique which involves simply dividing both double-word result terms by the first (high-order) word of the larger plus one. This ensures that the two resulting numbers are as large as possible. Clearly the method is not applied if the results already occupy only a single word each.

All of the above methods involve a certain amount of bookkeeping to deal properly with negative numbers and the idiosyncrasies of the particular scheme used to represent negative numbers.

ALGORITHMS

Let $n[a, b]$ stand for the normalization operation just discussed and let us represent a pair of words containing the numbers a and b as (a/b). Evidently, the four usual arithmetic operations produce the following results:

$$(a/b) + (c/d) \to n[ad + bc, bd]$$

$$(a/b) - (c/d) \to n[ad - bc, bd]$$

$$(a/b) \times (c/d) \to n[ac, bd]$$

$$(a/b) \div (c/d) \to n[ad, bc]$$

Curiously, with this representation, multiplication and division involve a little less effort than addition and subtraction. Also notice that division by zero does not lead to immediate disaster.

For reasons of efficiency one may want to retain integers represented by single words. The conversion between such single words and the double-word rational representation is straightforward of course and may be used to implement mixed arithmetic using a form of 'rational contagion'. That is, integers about to enter into operations with rationals are converted to rational form first. Alternatively, one can easily code the mixed arithmetic operations separately in order to gain speed by avoiding futile multiplications by one. Conversion between an external decimal representation and the internal rational format for input and output purposes requires division, but is also simple.

MATHEMATICAL FUNCTION SUBROUTINES

Most common methods for computing elementary transcendental functions involve argument reduction and approximation by a polynomial or a ratio of two low-order polynomials in the independent variable. Such techniques can easily be implemented for

minicomputers using rational arithmetic. This is sensible even when the rest of the program uses other representations since the resultant subroutines are comparatively fast and short. In this case one can often *simplify the normalization operation* after arithmetic steps since the magnitudes of the various coefficients and the range reduced input are known.

An additional idea worth exploring when using trigonometric functions is a representation for angles which lets 360° correspond to the largest integer that can be represented by a single word, plus one. This provides for automatic wrap-around of angles larger than 360° or less than zero on most machines. It also makes for the best possible angular resolution for a given word size. Further, it is a good idea to let arc-tangent be a function of two variables. The result is the arc-tangent of the ratio, with the quadrant picked according to the signs of the two arguments. This avoids the usual two-way ambiguity in the result of inverse trigonometric functions. Such a scheme also avoids difficulties with angles near 90° and 270° for which the argument would otherwise become excessively large.

COMPARISON WITH OTHER REPRESENTATIONS

The rational arithmetic method has already been compared with the usual single precision fixed-point and floating-point methods. A comparison with two other simple schemes, using two words to represent a number, may be called for. The first is a double-word representation with an imaginary binary point between the two words. That is, the first word is the integer part and the second the fractional part. This representation has the same dynamic range and comparable accuracy, but while addition and subtraction are simpler, the other operations, and particularly division, are much more difficult to is less implement.

Secondly, we could consider a simple floating point form, with one word used for the exponent, the other for the mantissa. While this avoids the difficulty of extracting substrings from words inherent in the usual floating-point representations, it is also wasteful. The dynamic range of course is larger than it is for the rational representation, but the accuracy is less and this technique is also more difficult to implement.

CASE STUDY AND SUMMARY

The techniques described here have been used to implement a mathematical function package for a PDP11 minicomputer. The approximations shown in Figures 6, 7 and 8 were used, the results are accurate up to the limitations of the word-length of the machine and the execution times are short (160 μs for ATAN, 185 μs for SIN/COS and 115 μs for SQRT on a PDP11/40). In this case, the normalization problem was simple since the expected ranges of the numbers involved were known in advance. Similar techniques have also been employed in a number of programs involving applications of machine vision and machine manipulation techniques.[5]

$$X\frac{1+4/15\ X^2}{1+3/5\ X^2} \qquad \text{FOR } |X|<1/2$$

$$X\frac{1+10/9\ X^2+5/21\ X^4}{1+7/9\ X^2+64/945\ X^4} \qquad \text{FOR } |X|<1$$

Figure 6. Rational function approximations for ATAN(X) for range-reduced X. The first approximation is obtained from an application of Aitken series acceleration, the second from continued fraction expansion. All coefficients are rational numbers and the results have sufficient accuracy for 16-bit minicomputers. The results are in radians, but can easily be converted into fractional revolution representation by multiplying by 355/113

$$X(1 - X^2/6\,(1 - X^2/20)) \qquad \simeq \text{SIN}\,(X) \quad \text{FOR} \mid X \mid < \pi/4$$
$$1 - X^2/2\,(1 - X^2/12\,(1 - X^2/30)) \simeq \text{COS}\,(X) \quad \text{FOR} \mid X \mid < \pi/4$$

Figure 7. Truncated Taylor series approximations for sine and cosine for range-reduced X (in radians). The results are of sufficient accuracy for 16-bit computers. A single function producing both values acts as a convenient inverse to the proposed arc-tangent function which takes two arguments

```
SQRT(Y):   LET X₁ ← 0; X₀ ← 1
           DO UNTIL X₁ = X₀ OR X₁ = Y/X₀
               LET X₀ ← X₁; X₁ ← (X₀ + Y/X₀)/2
           END
           RETURN X₁
END
```

Figure 8. Simple Newton–Raphson double-precision square-root algorithm suitable for minicomputer use. Divisions are intended to produce integer quotients. Division by two can obviously be implemented using a right shift. Other initial guesses, such as Y, Y/2 or (Y + 1)/2 could be used instead of 1

The 'no-point' or rational representation for numbers has been shown to make for easy-to-implement arithmetic operations, yet is effective in providing capabilities somewhere between those available with the more traditional fixed- and floating-point schemes. The low complexity and cost in terms of memory and execution time recommend this method for applications involving minicomputers. Users of minicomputers need no longer fear matrix arithmetic, coordinate transformations, evaluations of polynomials or calculation of transcendental functions since rational arithmetic will allow them to do a fair job of tackling these tasks, while requiring only a modest effort. It is the rational choice!

REFERENCES

1. P. Henrici, 'A subroutine for computations with rational numbers', *J. ACM*, **3**, 10–15 (1956).
2. D. E. Knuth, *The Art of Computer Programming*, Volume 2, Semi-numerical Algorithms, Addison-Wesley, Reading, Mass., 1969, pp, 290–292.
3. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford and London, 1954, pp. 137–138.
4. D. W. Matula, 'Fixed-slash and floating-slash rational arithmetic', *Proc. 3rd Symp. on Computer Arithmetic*, IEEE, 90–91 (1975).
5. B. K. P. Horn, 'A problem in computer vision: orienting integrated circuit chips', *Computer Graphics and Image Processing*, **4**, 294–303 (1975).