MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper No. 153 September 1977

RATIONAL ARITHMETIC FOR MINI-COMPUTERS

Berthold K. P. Horn

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 mini-computer 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.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-A-0643.

RATIONAL ARITHMETIC FOR MINI-COMPUTERS

1. INTRODUCTION:

Typically, mini-computers 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.

2. RATIONAL ARITHMETIC:

An oft overlooked alternative that lies between fixed point and floating point arithmetic in complexity, cost and capability is rational arithmetic (see Fig. 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 micro-program 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 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.

3. NORMALIZATION

The main decision to be made when implementing rational arithmetic is the choice of an appropriate "normalization" precedure, 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 a bit of work has to be done to find the greatest common divisor in the first place (see Fig. 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 to always provide the required compression of the two double-word results. This is the appropriate method however for exact rational arithmetic [1, 2].

The Eucledian 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 Fig. 3). The method is described in detail in [3] and reproduced in [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 Fig. 4). 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.

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 idiosyncracies of the particular scheme used to represent negative numbers.

4. 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:

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

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

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

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

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.

5. 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 mini-computers 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 360^0 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 360^0 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 90^0 and 270^0 for which the argument would otherwise become excessively large.

6. 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 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 sub-strings from words inherent in the usual floating point representations, it is also wasteful. The dynamic range of course is larger than the rational representation, but the accuracy is less and this technique is also more difficult to implement.

7. CASE STUDY AND SUMMARY

The techniques described here have been used to implement a mathematical function package for a PDP11 mini-computer. The approximations shown in Fig. 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 μ sec for ATAN, 185 μ sec for SIN/COS and 115 μ sec 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].

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 mini-computers. Users of mini-computers 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!

BIBLIOGRAPHY

[1] P. Henrici, "A Subroutine for Computations with Rational Numbers," J. ACM., Vol 3, pp 10-15, 1956.

[2] D. E. Knuth, "The Art of Computer Programming, Vol. 2, Semi-numerical Algorithms," Addison-Wesley, Reading MA, pp 290-292, 1969.

[3] D. W. Matula, "Fixed-Slash and Floating-Slash Rational Arithmetic," Proc. 3rd Sym. on Computer Arithmetic, IEEE, pp 90-91, Nov. 1975

[4] G. H. Hardy & E. M. Wright, "An Introduction to the Theory of Numbers," Claredon Press, Oxford, London, pp 137-138, 1954

[5] B. K. P. Horn, "A Problem in Computer Vision: Orienting Integrated Circuit Chips," Computer Graphics and Image Processing, Vol 4, pp 294-303, Sep. 1975.

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.

```
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 - (N/M)*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.

```
FRACT(S, N):  LET P1 ← 0; Q1 ← 1; P2 ← 1; Q2 ← 0; I ← 0
              DO UNTIL P2 ≥ N OR Q2 ≥ N OR S = I
                 LET P0 ← P1; Q0 ← Q1; P1 ← P2; Q1 ← Q2
                 LET I ← INTEGER-PART(S)
                 LET P2 ← P0 + I * P1; Q2 ← Q0 + I * Q1
                 IF S ≠ I, THEN LET S ← 1/(S - I)
              END
              RETURN (P1/Q1)
END
```

FIGURE 3: A simple algorithm for determining good rational approximations to a given number S. The succesive 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.

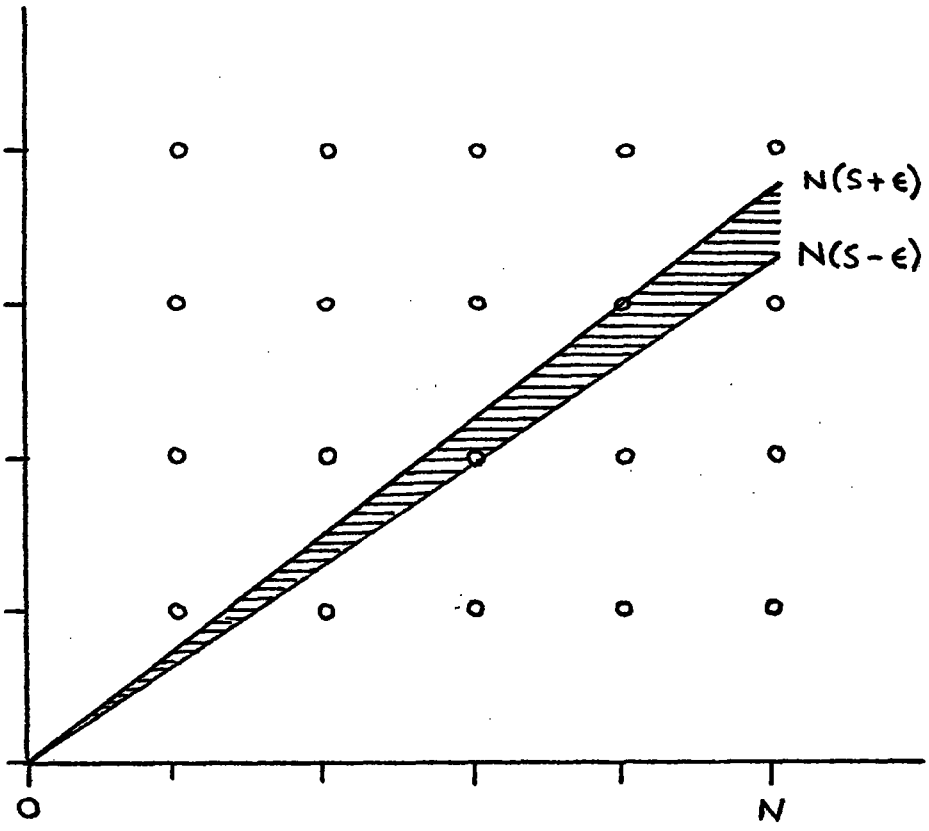

FIGURE 4: Illustration of the theorem about convergents. The grid points correspond to rational numbers. Those falling within the shaded triangle represent approximations to the number S. These rational numbers have a denominator less than N and lie between S - ϵ and S + ϵ in value. If the area of the triangle, ϵ N^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.

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.

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

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

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 mini-computers. The results are in radians, but can easily be converted into fractional revolution representation by multiplying by 355/113.

$$X\,(1 - X^2/6\,(1 - X^2/20)) \simeq \mathrm{SIN}(X) \quad \text{FOR} \quad |X| < \pi/4$$

$$1 - X^2/2\,(1 - X^2/12\,(1 - X^2/30)) \simeq \mathrm{COS}(X) \quad \text{FOR} \quad |X| < \pi/4$$

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 mini-computer 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.