Normalized Floating-Point Arithmetic with an Index of Significance

HERBERT L. GRAY† AND CHARLES HARRISON, JR.†

General Remarks on Error and Significance

IT HAS BEEN frequently pointed out that the task of determining an error-bound for the results of a problem is usually a long difficult calculation, which is avoided as much as possible by the programmer. The introduction of floating-point arithmetic in modern computers and the ever-growing use of compilers makes the task of error analysis even more difficult and its computation even less probable. Clearly, a machine method is needed to automatically calculate a bound for the propagated and generated error, given the initial error in the input and the residual error due to approximating functions.

Two methods for doing this herein are called the "normalized significance" and the "unnormalized significance" methods. The "normalized significance" method always keeps the floating-point number normalized and provides an index of significance. The "unnormalized significance" method does not normalize floating-point numbers and uses the count of digits remaining after leading zeros as an indication of their significance.

Before discussing these methods, we should like to define what is meant by significance. If one says that α digits of a number are significant, one means that the error in the number is less than $B^{-(\alpha+n)}$ where B is the base and n is the number of leading zeros. In the normalized system, n would be zero. In the unnormalized system, n would be the total number of digits minus α, as there are no insignificant digits.

Thus, in the normalized method, as many digits as possible of a number are retained, and its index determines the number of digits which are significant; in the unnormalized method, only the digits considered significant are retained.

Since the error in any one step of a calculation is not usually a factor of the base, B, and α in either system only allows adjustment to within a factor of B, any set of arithmetic rules can have the desired property only "on the average". The design of such rules must rest upon general assumptions concerning the statistics of computation; these rules may, therefore, not be valid in special situations.

Normalized and Unnormalized Methods Contrasted

In the normalized method to be used with Argonne's arithmetic unit FLIP, each number is represented with base 2 by a triplet (x_f, x_p, x_i) where x_f is the fractional part of the number with $\frac{1}{2} \leq |x_f| < 1$, x_p is the associated power, and x_i is the index of significance. Thus $X = x_f \cdot 2^{x_p}$, to x_i significant figures. In addition and subtraction, the result has an index equal to the smaller of the two indices of the operands. Whenever a fraction is down-scaled, its index and power are increased; whenever a fraction is up-scaled its index and power are reduced. In multiplication and division, the index of the result is the smaller of the two indices; and even if the result needs to be scaled to bring it into the normal range, the index is not changed.

The "unnormalized method" is very similar to the one to be used by the University of Chicago in their new computer.[1] Each number is represented by a couplet (x_f, x_p), where $0 \leq |x_f| < 1$ and the number of digits of x_f minus the number of leading zeros equals x_i of the normalized scheme. Insignificant digits are shifted off the register. In addition and subtraction, no scaling of the result is carried out, unless, of course, its fraction is greater than or equal to unity. In multiplication and division, the resultant fraction is scaled so as to have the same number of leading zeros as the operand with the fewer number of significant figures.

In his error analysis of floating point procedures in the *Communications of the Association for Computing Machinery*[2], John W. Carr III points out that a normalized floating point procedure will always give a better result than an unnormalized procedure. While the normalized method will create less error, the loss of possible significant digits due to the shorter register-length remaining when an index of significance is used may outweigh this gain.

The logic for addition and subtraction, due to the normalization which a result may require and the handling of the index, are more complicated in the normalized method. However, division and multiplication are much more complicated in the unnormalized system because of the shifting required to obtain the correct number of leading zeros. Since the number of additions and subtractions in a calculation is usually greater than the number of multiplications and divisions, the unnormalized system might be a little faster.

* Work performed under the auspices of the U. S. Atomic Energy Commission.

† Argonne National Laboratory, Lemont, Ill.

[1] R. L. Ashenhurst, and N. Metropolis, "Unnormalized Floating Point Arithmetic," *Journal of the ACM*, July, 1959, pp. 415–429.

[2] Carr, John W. III, Error Analysis in Floating Point Arithmetic, *Communications of the ACM*, May, 1959, pp. 10–15.

Typical Examples

Following are some examples. Those numbered with (*a*) use the Argonne method, while those numbered with (*b*) use the second method. The length of the fractions is adjusted so that the register in each scheme contains the same number of digits.

(*1a*) (0.5782, 6, 3) + (0.1485, 4, 3): [Argonne scheme]

0.57820, 6, 3	Arithmetic register holds 1 more digit than storage.
+0.00148, 6, 5	Power and index adjusted.
0.5797, 6, 3	Result rounded, smaller index is used.

(*1b*) (0.00578, 8) + (0.00149, 6): [Other scheme]

0.005780, 8	Register again holds one extra digit.
+0.000014, 8	Power adjusted.
0.00579, 8	

(*2a*) (0.1397, 5, 4) − (0.9143, 4, 2):

0.13970, 5, 4	
−0.09143, 5, 3	Power and index adjusted.
0.04827, 5, 3	Smaller index used.
0.4827, 4, 2	Number normalized.

(*2b*) (0.01397, 6) − (0.00091, 7):

0.001397, 7	Power adjusted.
−0.000910, 7	
0.00049, 7	Number rounded to 2 significant digits.

(*3a*) (0.9143, 4, 2) + (0.1875, 4, 4):

0.9143, 4, 2	No power adjustment needed.
0.1875, 4, 4	
1.1018, 4, 2	Fraction exceeds one.
0.1102, 5, 3	Normalization and roundoff.

(*3b*) (0.00091, 7) + (0.01875, 5):

0.000910, 7	
0.000187, 7	
0.00110, 7	Rounded to 3 figures.

(*4a*) (0.5782, 6, 3) × (0.1485, 4, 4):
(0.5782, 6, 3) × (0.1485, 4, 4) = .08586270, 10, 3
[Double-register product]
= .8586, 9, 3
[Normalized and rounded]

Index not changed. The digit which was shifted into register was assumed to be good (optimistic approach).

(*4b*) (0.00578, 8) × (0.01485, 5):
(0.00578, 8) × (0.01485, 5) = 0000858330, 13
= .00858, 11
[Rule: result always contains same number of significant digits as the least significant operand.]

(*5a*) (0.5782, 6, 3) × (0.2485, 4, 4):
(0.5782, 6, 3) × (0.2485, 4, 4) = .14368270, 10, 3
= .1437, 10, 3
[Rounded]

(*5b*) (0.00578, 8) × (0.02485, 5):
(0.00578, 8) × (0.02485, 5) = .0001436330, 13
= .00144, 12
[Adjusted and rounded]

(*6a*) (0.5782, 6, 3) ÷ (0.2485, 4, 4)
(0.5782, 6, 3) ÷ (0.2485, 4, 4)
= 2.327605 . . . , 2, 3
= 0.2328, 3, 3
[Normalized and rounded]

Index does not change after normalization. Extra digit assumed to be no good (pessimistic approach).

(*6b*) (0.00578, 8) ÷ (0.02485, 5)
(0.00578, 8) ÷ (0.02485, 5) = 0.2325955 . . . , 3
= 0.00233, 5
[Adjusted and rounded]

Same rule holds for division as well as multiplication.

(*7a*) (0.2485, 4, 4) ÷ (0.5782, 6, 3)
(0.2485, 4, 4) ÷ (0.5782, 6, 3)
= 0.429782 . . . , −2, 3
= 0.4298, −2, 3
[Rounded]

(*7b*) (0.02485, 5) ÷ (0.00578, 8)
(0.02485, 5) ÷ (0.00578, 8) = 4.29930 . . . , −3
= 0.00430, 0
[Adjusted and rounded]

Remarks on Special Cases

In the test cases we have run so far using the Argonne scheme the number of multiplications far exceeds the number of divisions; this gives rise to optimistic indices, though none of the indices have been off by the equivalent of more than one decimal place. In both systems, a special number exists which does not obey the above rules. This number is zero and it follows the following rules in both systems.

$$X \pm 0 = X$$
$$X \times 0 = 0$$
$$0 \div X = 0$$
$X \div 0$ is not attempted.

If during the normalization process in the Argonne scheme, the index of significance is about to become

less than zero, then the shifting is stopped so that the index remains at zero. Thus, at the end of this operation, the fractional part of the number may be less than one-half. This number, known as a relative zero, obeys the same rules as any normalized number during addition, subtraction and multiplication; but division by this number, as with true zero, is not attempted. Whenever the normalization is not completed, the Argonne machine will jump to a fixed location in the memory and let the program decide what is to be done about the number.

Relative zero may also be introduced as an input number. In this case, the fraction is unnormalized, but the index need not be zero. This number is used in special operations such as finding the integral part of another number, as follows:

0.4597, 2, 4	
+0.0000, 4, 4	
0.0045, 4, 4	Adjust power (index at max.)
0.4500, 2, 2	Normalization.

This type of relative zero is also used in operations which convert from floating to fixed point. A relative zero with an index not equal to zero can never be generated by the floating-point unit.

The only other special number which may arise in the second scheme is a number with zero significant figures and thus a zero fraction. All operations with this number except division follow the normal rules.

If a number in the Argonne scheme does not have any associated error, it is said to be "totally represented". Such numbers are given a special index and operated on in a slightly different way. As long as a calculation uses only totally represented numbers, the index of the result does not change even if normalization occurs. If, however, during a calculation any non-zero digits of a number are shifted off, or if one of the operands is not a totally represented number, then computation of the index of the result reverts to its normal form. For example:

(.4583, 2, TR)	(Totally represented)
−(.4329, 2, TR)	
.0254, 2, TR	
.2540, 1, TR	Normalization.

(.4583, 2, TR)	
+(.1497, 1, TR)	
.4583, 2, TR	
.01497, 2, TR	Adjust power
.4733, 2, 4	Results rounded to 4 figures.

.4583, 2, TR	
−.4329, 2, 4	Not a totally represented number.
.0254, 2, 4	Index reverts to its normal form.
.2540, 1, 3	

Such a scheme helps to simplify floating-point-integer arithmetic.

Tests of Argonne Method in Representative Problems

To test the accuracy of the index in the floating-point scheme, we simulated FLIP with a GEORGE program. We combined this program with some GEORGE subroutines and compared the results obtained in this manner with those computed by other means.

Characteristic Polynomial of a Matrix

First, let us consider the routine involving the method of Leverrier for determining the coefficients of the characteristic polynomial of a real-valued square matrix.

Let λ_i = roots of the characteristic polynomial of our $n \times n$ matrix $\mathbf{A}$.

Define $S_k = \text{trace } (\mathbf{A}^k) \quad k = 1, 2, \ldots, n$

$$S_k = \sum_{i=1}^{n} a_{ii}^{(k)}$$

The characteristic equation is

$$\lambda^n + C_1\lambda^{n-1} + C_2\lambda^{n-2} + \ldots + C_{n-1}\lambda + C_n = 0$$

where

$$-C_1 = S_1$$
$$-2C_2 = S_1C_1 + S_2$$
$$-3C_3 = S_1C_2 + S_2C_1 + S_3$$
$$\ldots\ldots\ldots\ldots$$
$$-nC_n = S_1C_{n-1} + S_2C_{n-2} + \ldots + S_{n-1}C_1 + S_n$$

The method requires the computation of $\mathbf{A}^2, \mathbf{A}^3, \ldots, \mathbf{A}^n$ and the corresponding $S_1, S_2, S_3, \ldots, S_n$. There are $n^3(n-1)$ multiplications and $n^3(n-1) + n(n-1)$ additions involved in these calculations. Obviously, inaccuracies can be caused if n is very large or the elements of $\mathbf{A}$ are too large.

We ran tests with our combined program using four different matrices. In the first two cases tested, the $\mathbf{A}$'s were of the tenth order and of the forms:

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ 2 & 2 & \ldots & 2 \\ . & . & \ldots & . \\ 10 & 10 & \ldots & 10 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 2 & 2 & \ldots & 2 \\ 4 & 4 & \ldots & 4 \\ . & . & \ldots & . \\ 20 & 20 & \ldots & 20 \end{pmatrix}$$

In both cases $C_1 = -S_1$ and $C_2 = C_3 = \ldots = C_{10} = 0$. We obtained exact results and the index indicated this for each coefficient.

Next we tried a twelfth-order matrix of the form

$$\begin{pmatrix}
4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4
\end{pmatrix}$$

Once again, we obtained exact results with the corresponding correct indices. It was with this case that we noticed the vast difference a change in method made in the final results. We computed the coefficients from the equation

$$C_k = (S_1C_{k-1} + S_2C_{k-2} + \ldots + S_{k-1}C_1 + S_k) \div (-k)$$

If, however, we use the slightly different equation

$$C_k = -\frac{1}{k}(S_1C_{k-1} + S_2C_{k-2} + \ldots + S_{k-1}C_1 + S_k),$$

we get different results. In fact, C_{12} is completely in error and the index associated with it is zero. This zero index means the number is worthless as a result.

The initial error was introduced in the computation of $1/k$, since non-terminating decimals like 1/3, 1/6, etc. cannot be exactly represented. The use of these inexact numbers produced inexact coefficients. The errors in both of these were propagated in the calculations of subsequent coefficients. As a result, C_{12} was completely in error.

The next matrix we used was a fifteenth order matrix having all its elements equal to 1. Even though the elements are not large, the size of n leads to some very large intermediate results. For example, $S_{15} = 15^{15} = 437{,}893{,}890{,}380{,}859{,}375$. The coefficients, however, were more reasonable. In fact $C_1 = -S_1$ and $C_2 = C_3 = \ldots = C_{15} = 0$. In this test we also found complete agreement with the indexed results and the known values for the coefficients.

Inversion of a Hilbert Matrix

Next we applied our program to Jordan's method for the inversion of Hilbert matrices of various orders. Hilbert matrices are of the form

$$\begin{pmatrix}
1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\
\frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\
\frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+2} \\
\cdot & \cdot & \cdot & \cdots & \cdots \\
\frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1}
\end{pmatrix}.$$

Hilbert matrices are almost singular and as the size of the order increases, the value of the determinant rapidly approaches zero. Several matrices were inverted including a 12×12 which had a determinant approximately equal to 10^{-75}. We compared the results we obtained with those computed by an inversion formula.[3]

The elements in our inverted matrices compared favorably with those in the table of inverses for all the orders tested. We were able to get 10- or 11-digit agreement in the elements of an inverted seventh-order Hilbert matrix and 6-or 7-digit agreement for a 10×10. The indices associated with these inverse elements were either correct or one-digit optimistic. This is about what we expected since we employ the optimistic approach in our floating-point scheme.

It may be interesting to note that Jordan's method on a floating-point machine produced inverses above the 6th order that were completely in error. Also, our GEORGE interpretive floating-point subroutine was useless beyond a 7th-order Hilbert matrix. The primary difference in these cases is the number of digits the machine can hold. However, the other two schemes give no indication of the approximate accuracy of the results.

Evaluation of a Polynomial

A third test was performed with a subroutine which evaluates a polynomial, with real or complex coefficients, having real or complex roots. We inserted known roots for certain 4th and 8th degree polynomials with real coefficients and noted the computed value of the polynomial.

When the roots were real and exact, the computed value of the polynomial was exactly zero. The same thing was true when the roots were complex but with exact real and imaginary parts. However, when the

[3] I. Savage and E. Lukacs, "Tables of Inverses of Finite Segments of the Hilbert Matrix" in *Contributions to the Solution of Systems of Linear Equations and the Determination of Eigenvalues*, NBS Applied Mathematics Series 39, p. 105–108, 1954.

roots were complex and at most one of the parts was exact we get an evaluation of the polynomial which is not exactly zero and has a zero index. Here, the zero index meant the digits in the number were meaningless but the zeros preceding the first of these digits were correct.

Numerical Integration

Finally, we applied our program to a trapezoidal integration subroutine. We evaluated some simple integrals between exact limits. If the results could be represented exactly, we obtained exact results. When the results could not be exactly represented we obtained numbers whose error was less than 5×10^{-18}. For these cases, the index on the partial sums was sometimes less than the maximum, but continued summations caused it to increase up to the maximum allowed for non-totally-represented numbers.

Conclusions

The tests we ran were considered fairly representative of the calculations involved in many problems. The fact that the index of significance was never more than one digit from its true value lends merit to the Argonne scheme. If some of the multiplications could have been replaced by divisions without increasing the errors, the index would have been even closer to the true value.