ROUNDING ERRORS IN DIGITAL COMPUTER ARITHMETIC SUBROUTINES

by

GARY JOSEPH LASTMAN
B.A.Sc., The University of British Columbia, 1961

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ARTS

in the Department

of

MATHEMATICS

We accept this thesis as conforming to the

required standard

THE UNIVERSITY OF BRITISH COLUMBIA
March, 1963

Abstract

In this thesis we investigate arithmetic subroutines and round-off procedures. An error analysis of single operations in normalized floating point arithmetic leads us to the construction of an improved form of addition-subtraction subroutine. In addition, the properties of several types of round-off procedures are examined (adding $\frac{1}{2}$; adding random digits; dropping digits).

The experimental work (using the above mentioned subroutines) with the system $\dot{x} = y$, $\dot{y} = -x$ shows that the systematic accumulation of round-off error observed by Huskey is due to the type of rounding-off procedure used. Furthermore, Hartree's explanation of this effect is found to be inadequate because carrying extra digits throughout the calculations does not eliminate the systematic round-off.

Acknowledgement

I wish to thank Dr. T. E. Hull and Dr. Charlotte Froese for their helpful suggestions in the preparation of this manuscript.

The financial assistance of the Defence Research Board of Canada (grant number DRB-9540-03) is gratefully acknowledged.

Table of Contents

Chapter I

For the numerical solution of a mathematical problem on a digital computer we should provide some sort of error analysis to account for round-off errors, which are usually present. The propagated round-off error may be given in terms of a maximum upper bound or an expected value and variance: the statistical estimate is much more satisfactory. Henrici [4] strongly advocates the usage of statistical methods, but Huskey cautions against their application on the basis of some experimental results, [7]; it is our intention to try to explain Huskey's results by examining various types of arithmetic subroutines and rounding-off procedures; in addition, we will show that Hartree's explanation [7] of Huskey's results is not sufficient to explain the observed effect. For this purpose we first discuss computing schemes and rounding-off procedures in Chapter II. In Chapter III we give an error analysis of single operations in normalized floating point arithmetic. A special form of subroutine and a special rounding-off procedure are described in Chapter IV, while in Chapter V we present some experimental results.

Chapter II

In general, errors are introduced when any mathematical problem is solved by numerical computation on a digital computer. We wish to determine the effect of round-off errors on the numerical solution to a problem, so henceforth we shall be concerned with the arithmetic operations of addition, subtraction, multiplication and division. Since these operations are dependent upon the manner in which the computations are performed, we first discuss various computing schemes.

Numbers in a digital computer can be represented as an ordered pair (m,e) where m is the mantissa of word length ℓ digits and e is the exponent; both are expressed in the number base b of the machine. A machine number (m,e) is actually $m.b^e$. If the exponent e is a predetermined constant for all calculations we have the system known as fixed point arithmetic. If we allow e to vary between fixed limits, say $-a \leq e \leq a$, then we have floating point arithmetic.

With fixed point arithmetic we usually deal with ℓ-digit numbers of the form (m,o), with $-1 < m < 1$. This system is very inflexible since one must rescale numbers so that the results of numerical computations will lie within the open interval (-1,1). Also, the programmer must keep track of the decimal point throughout the calculations. But, by using a floating point arithmetic the programmer is not required to locate the decimal point at each step in a program, or to rescale. Because of these advantages over fixed point, floating point arithmetic is now used in almost all computers.

In constructing a floating point arithmetic we become aware of two facts: (1), that although we can perform calculations using numbers whose magnitudes vary over a considerable range, the maximum number of digits we can retain at any stage is the word length of m, the mantissa of our floating point number; (2), the representation for a number x, (m,e), is not unique(*). For uniqueness we must turn to a normalization convention such as $b^{-1} \leq |m| < 1$. Even so, the representation for zero is not unique; but we may define zero as (o, -a), where -a is the lower bound for e . By restricting $|m|$ to the range $b^{-1} \leq |m| < 1$ we retain a maximum number of significant digits at each computation step. However, now we are uncertain as to the number of truly significant digits retained in a long series of calculations. This loss of information can be partially eliminated by the use of an unnormalized floating point or significant digit arithmetic, [1], [2] . But now we must forfeit the obvious procedural conveniences of normalized floating point arithmetic. Because of this the latter system is more commonly used in computers.

In summary: floating point arithmetic is extremely useful for computations where the magnitudes of quantities may vary widely. With normalized floating point it is possible to represent numbers in a computer by storing in memory only the first ℓ significant digits and an exponent. The floating point system has two obvious disadvantages: (1), more complicated computer hardware is needed to handle both the mantissa and the exponent, and (2), fewer significant digits can be carried because a portion of a word must be used for the exponent. In spite of this, floating point is an effective and efficient method of representing numbers within a computer.

Now that we have some practical arithmetics that can be harnessed to do

---

(*) This is immediately evident since we could have $mb^e \equiv m^*b^{e^*}$ with $m \neq m^*$ and $e \neq e^*$.

our numerical work, we are in a position to give a qualitative description of rounding errors.

The process of rounding-off is that procedure by which low-order digits are eliminated in a number. Consequently round-off error is the difference between a finite precision number (after the low-order digits are eliminated) and the corresponding infinite precision number. Bearing in mind the computer and its arithmetic unit we classify as round-off errors those errors induced by special representations of numbers and also those errors resulting from arithmetic operations. Both are caused by rounding-off.

Several schemes have been proposed to implement the round-off process. In each a number x is changed to $\bar{x}$, its rounded form. Let R denote the rounding-off operator ($R: x \rightarrow \bar{x}$) : if $x > o$ we round x itself; if $x < o$ we round the absolute value of x, $|x|$; if $x = o$ we do not round at all. Let δ be the error in the least significant digit retained in $|x|$, after rounding-off; δ gives a measure of the accuracy of a rounding procedure.

For every definition of R we obtain a different rounding scheme :

(i) Add b/2 to $|x|$ at the position which is to be dropped.(*) A carry will result if this digit is b/2 or greater. $\left\{ -\left(\frac{1}{2}\right)_b < \delta \leq \left(\frac{1}{2}\right)_b \right\}$

(ii) Make the lowest-order retained digit of $|x|$ equal to b/2 regardless of the correct value of that digit and the digits in lower orders.

$$\left\{ -\frac{b}{2} < \delta \leq \frac{b}{2} \right\}$$

(iii) Add a "0" or "1" randomly into the lowest-order retained digit regardless of the values of the digits of $|x|$. $\{ -1 < \delta \leq 1 \}$

(iv) Discard low-order digits. The error is $\{ 0 \leq \delta < 1 \}$.

(*) b is the number base of the machine.

Every one of the preceding schemes has its own merits: the final selection of a particular rounding procedure is a compromise between the desired accuracy for a result and the computer time required to achieve this accuracy. For example, procedure (iv) is the least time-consuming but gives a large biased error, while (i) has the minimum error but takes longer.

Chapter III

Regardless of the numerical scheme that we choose to perform our computations, the rounding-off process may introduce errors. For a quantitative description of these errors we now give an error analysis of single operations in normalized floating point arithmetic. The error analysis will tell us the accuracy of the individual arithmetic operations.

In the discussion, real arithmetic operations will be denoted by their usual symbols $+$, $-$, $\times$ or $\cdot$, $\div$; the corresponding machine operations will be $\oplus$, $\ominus$, $\otimes$, $\oslash$; also, x,y,z shall be machine numbers. The machine base will be b and the mantissas will have ℓ digits. Zero will be of the form (0, -a) where -a is the lower limit on the range of values of the exponents. Errors ε_i , $\bar{\varepsilon}_i$ will be errors in the least significant digit of the resulting ℓ-digit mantissa. In the following error analyses we shall assume that neither exponential underflow nor exponential overflow occur.

The area of computer memory in which a numerical result is formed is known as an accumulator. In the error analyses two different types of accumulators are used: the single-length ℓ-digit accumulators, and the "extra-digit" accumulators. In the usual ℓ-digit accumulator the "decimal" point is considered to be immediately to the left of the first (most significant) digit. An $(\ell+1)$ - digit accumulator is formed from this by providing an extra digit position to the left of the "decimal" point. This form of extra digit accumulator is used for addition, subtraction and division. A second form of the extra-digit accumulator is the double-length one used in multiplication; multiplication of two ℓ-digit numbers gives a product of 2ℓ digits. We form this double-length accumulator by using two single length accumulators.

A. Addition - Subtraction

$$x \oplus y = (m_1, e_1) \oplus (m_2, e_2) = (m_3, e_3)$$

Let $|\varepsilon_i| \leq \varepsilon_A \quad ; \quad i = 1, 2$

If $x = -y$ then $x \oplus y = (o, -a)$; suppose $e_2 \geq e_1$, if $e_2 \geq e_1 + \ell$, then

$x \oplus y - (x+y) = -x$.

For $|e_2 - e_1| \leq \ell - 1$,

1. $(\ell+1)$ - digit accumulator

(a) We must shift m_1 to align the decimal points: we form a new mantissa V, of ℓ digits.

$$V = m_1 b^{e_1 - e_2} + \varepsilon_1 b^{-\ell}$$

(V has $|e_1 - e_2|$ significant zeros)

(b) Add m_2 and V : $m_2 + V$

(c) Form the ℓ-digit mantissa of the result.

(i) No carry on the addition of m_2 and V

$$m_3 = (m_2 + V) b^{j} \qquad j = 0, 1, 2, \ldots, \ell - 1$$
$$e_3 = e_2 - j$$

where j is the number of significant zeros before normalization.

Thus

$$X \oplus Y = b^{e_3}\left[(m_2 + V)b^{j}\right] = b^{e_3}\left[m_2 + m_1 b^{e_1 - e_2} + \varepsilon_1 b^{-\ell}\right] b^{j}$$

$$= m_1 b^{e_1} + m_2 b^{e_2} + \varepsilon_1 b^{-\ell} b^{e_2}$$

$$= X + Y + \varepsilon_1 b^{-\ell} b^{e_2}$$

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{-\ell} b^{e_2}$$

Notice that we do not have an error bound in terms of the exponent of the final result. If we wish to use e_3 we have that

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{-\ell} b^{e_3 + j} .$$

But we do not know j . All we can say is that $j \le \ell - 1$, and therefore

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{e_3 - 1}$$

However, for almost all additions and subtractions, this bound grossly over-estimates the error. We conclude that the only realistic bound is

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{-\ell} b^{e_2} .$$

If we had performed the error analysis using $e_1 \ge e_2$ we would have obtained an error bound

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{-\ell} b^{e_1} .$$

Therefore, the most general error bound in this case is

$$|X \oplus Y - (X+Y)| \le \varepsilon_A b^{-\ell} b^{\max(e_1, e_2)}$$

(ii) Carry on the addition of m_2 and V .

Since (m_2 + V) has ℓ+1 digits we must shift it one digit to the right.

$$m_3 = b^{-1}(m_2 + V) + \varepsilon_2 b^{-\ell}$$
$$e_3 = e_2 + 1$$

$$X \oplus Y = X + Y + b^{-\ell} b^{e_3 - 1}\left[\varepsilon_1 + b\varepsilon_2\right]$$

$$\left|X \oplus Y - (X+Y)\right| \leq \varepsilon_A b^{-\ell} b^{e_3-1}\left[1 + b\right]$$

Here the error is given explicitly in terms of the exponent of the final result.

2. ℓ - digit accumulator.

(i) Without carry, the results are the same as for the ℓ+1 digit accumulator.

(ii) Carry

Now we must shift both m_1 and m_2 (assume $e_2 \geq e_1$)

(a) Shift m_2 and form V_2

$$V_2 = m_2 b^{-1} + \varepsilon_1 b^{-\ell}$$

(b) Shift m_1 and align decimal points

$$V_1 = m_1 b^{-1} b^{e_1 - e_2} + \varepsilon_2 b^{-\ell}$$

(c) Add V_1 and V_2 ; both have only ℓ digits.

$$m_3 = V_1 + V_2$$

$$e_3 = e_2 + 1$$

$$X \oplus Y = X + Y + b^{-\ell} b^{e_3}\left[\varepsilon_1 + \varepsilon_2\right]$$

$$\left|X \oplus Y - (X+Y)\right| \leq 2\varepsilon_A b^{-\ell} b^{e_3}$$

If we compare this error bound to the corresponding error bound in 1. above, we see that this bound is larger by a factor of $\rho = 2b/(1+b)$.

For $b \geq 2$, $1 < \rho < 2$.

$b = 2 \quad \rho = 4/3 \simeq 1.33$

$b = 10 \quad \rho = 20/11 \simeq 1.82$

Therefore, if possible one should use an $(\ell+1)$ digit accumulator for addition and subtraction.

B. Multiplication

$$X \otimes Y = (m_1, e_1) \otimes (m_2, e_2) = (m_3, e_3)$$

Let

$$|\varepsilon_i| \le \varepsilon_M \quad , \quad |\bar{\varepsilon}_i| \le \bar{\varepsilon}_M \quad ; \quad i = 1, 2$$

If either or both x, y are zero

then $$X \otimes Y = (0, -a)$$

1. A 2ℓ-digit accumulator

(a) Form the product $m_1 \cdot m_2$

(b) If $b^{-2} \le |m_1 m_2| < b^{-1}$

$$m_3 = b(m_1 \cdot m_2) + \varepsilon_1 b^{-\ell}$$

$$e_3 = e_1 + e_2 - 1$$

$$X \otimes Y = X \cdot Y + \varepsilon_1 b^{-\ell} b^{e_3}$$

$$|X \otimes Y - X \cdot Y| \le \varepsilon_M b^{-\ell} b^{e_3}$$

(c) If $b^{-1} \le |m_1 \cdot m_2| < 1$

$$m_3 = m_1 \cdot m_2 + \varepsilon_2 b^{-\ell}$$

$$e_3 = e_1 + e_2$$

$$X \otimes Y = X \cdot Y + \varepsilon_2 b^{-\ell} b^{e_3}$$

$$|X \otimes Y - X \cdot Y| \le \varepsilon_M b^{-\ell} b^{e_3}$$

In both cases we have the error bound in terms of the exponent of the result. The most general error bound is

$$|X \otimes Y - X \cdot Y| \leq \varepsilon_M b^{-\ell} b^{e_3}$$

2. An ℓ-digit accumulator

(a) Form

$$V_1 = m_1 + \bar{\varepsilon}_1 b^{-c_1}$$

$$V_2 = m_2 + \bar{\varepsilon}_2 b^{-c_2}$$

where $c_1 + c_2 = \ell$

If ℓ is even, take $c_1 = c_2 = \ell/2$

If ℓ is odd, take $c_1 = (\ell - 1)/2$, $c_2 = (\ell + 1)/2$

(b) Multiply V_1 and V_2 to get an ℓ-digit number

(c) $b^{-2} \leq |V_1 \cdot V_2| < b^{-1}$

$$m_3 = b(V_1 \cdot V_2)$$

$$e_3 = e_1 + e_2 - 1$$

$$X \otimes Y = X \cdot Y + b^{e_3+1}\left[m_2 \bar{\varepsilon}_1 b^{-c_1} + m_1 \bar{\varepsilon}_2 b^{-c_2} + \bar{\varepsilon}_1 \bar{\varepsilon}_2 b^{-\ell}\right]$$

$$|X \otimes Y - X \cdot Y| < \bar{\varepsilon}_M b^{e_3+1}\left(b^{-c_1} + b^{-c_2} + \bar{\varepsilon}_M b^{-\ell}\right)$$

(d) $b^{-1} \leq |V_1 \cdot V_2| < 1$

$$m_3 = V_1 \cdot V_2$$

$$e_3 = e_1 + e_2$$

$$X \otimes Y = X \cdot Y + b^{e_3}\left[m_2 \bar{\varepsilon}_1 b^{-c_1} + m_1 \bar{\varepsilon}_2 b^{-c_2} + \bar{\varepsilon}_1 \bar{\varepsilon}_2 b^{-\ell}\right]$$

$$|X \otimes Y - X \cdot Y| < \bar{\varepsilon}_M b^{e_3}\left[b^{-c_1} + b^{-c_2} + b^{-\ell}\bar{\varepsilon}_M\right]$$

In each of (c) and (d) the error bound is proportional to $b^{-\ell/2} b^{e_3}$; but for a double-length accumulator the error is proportional to $b^{-\ell} b^{e_3}$.

Clearly a double-length accumulator is the better of the two.

C. Division

$$X \oplus\!\!\!\!\div\, Y = (m_1, e_1) \oplus\!\!\!\!\div\, (m_2, e_2) = (m_3, e_3)$$

We must have $y \neq 0$. If $x = 0$ then $X \oplus\!\!\!\!\div\, Y = (0, -a)$

Let $|\varepsilon_i| \leq \varepsilon_D$, $|\bar{\varepsilon}_i| \leq \bar{\varepsilon}_D$; $i = 1, 2$

1. ℓ + 1 digit accumulator (a quotient of ℓ+1 digits)

(a) $1 \leq |m_1/m_2| < b$

$$m_3 = b^{-1}(m_1/m_2) + \varepsilon_1 b^{-\ell}$$

$$e_3 = e_1 - e_2 + 1$$

$$X \oplus\!\!\!\!\div\, Y = (X \div Y) + \varepsilon_1 b^{-\ell} b^{e_3}$$

$$|X \oplus\!\!\!\!\div\, Y - (X \div Y)| \leq \varepsilon_D b^{-\ell} b^{e_3}$$

(b) $b^{-1} < |m_1/m_2| < 1$

$$m_3 = m_1/m_2 + \varepsilon_2 b^{-\ell}$$

$$e_3 = e_1 - e_2$$

$$X \oplus\!\!\!\!\div\, Y = (X \div Y) + \varepsilon_2 b^{-\ell} b^{e_3}$$

$$|X \oplus\!\!\!\!\div\, Y - (X \div Y)| \leq \varepsilon_D b^{-\ell} b^{e_3}$$

The general error bound is

$$|X \oplus\!\!\!\!\div\, Y - (X \div Y)| \leq \varepsilon_D b^{-\ell} b^{e_3}$$

2. An ℓ-digit accumulator

(a) $1 \leq |m_1/m_2| < b$

Shift m_1 and form V_1

$$V_1 = b^{-1}m_1 + \bar{\varepsilon}_1 b^{-\ell}$$

$$m_3 = (V_1 \div m_2) + \varepsilon_1 b^{-\ell}$$

$$e_3 = (1 + e_1) - e_2$$

$$X \oplus\!\!\!\!\div\, Y = (X \div Y) + b^{-\ell} b^{e_3} \left[(\bar{\varepsilon}_1 \div m_2) + \varepsilon_1\right]$$

$$|X \oplus\!\!\!\!\div\, Y - (X \div Y)| \leq b^{-\ell} b^{e_3} \left[\varepsilon_D + b\bar{\varepsilon}_D\right]$$

(b) $b^{-1} < |m_1/m_2| < 1$

This is the same as the corresponding case for $(\ell+1)$ digits.

The preceding error bounds characterize the specific machine arithmetic operation. In the next chapter we examine further characteristics of these machine operations.

Chapter IV

Automatic programming languages must use permanent machine language subroutines to perform arithmetic operations. Since the accuracy of calculations is vitally dependent upon these arithmetic subroutines, we should use routines whose maximum error is as small as possible.

Let us consider the normalized floating point arithmetic procedures discussed in Chapter III (those for extra-digit accumulators). Define t as follows:

$$t = \max \left| \frac{(X \approx Y) - (X \sim Y)}{X \sim Y} \right|$$

where $x \sim y \neq o$

$\sim$ denotes a real arithmetic operation,

$\approx$ denotes the corresponding machine operation.

Addition-Subtraction

(a) No carry

$$t_{A1} = \varepsilon_A b^{-\ell} \max \left| \frac{b^{\max(e_1, e_2) - e_3}}{m} \right| = k_1 b$$

where $e_3 = \max(e_1, e_2) - j \quad ; \quad j = 0, 1, \ldots, \ell$

(b) Carry

$$t_{A2} = \varepsilon_A (1 + b) b^{-\ell} = k_2 b^{-\ell}$$

Multiplication $t_M = k_3 b^{-\ell}$

Division $t_D = k_4 b^{-\ell}$

The maximum relative error, t, is proportional to $b^{-\ell}$ for multiplication, division and addition-subtraction with carry. "No carry" on addition-subtraction can produce a large value of t (proportional to b). From this evidence we conclude that the usual procedures for addition-subtraction are so inaccurate that we must develop a procedure which always gives a value of t proportional to $b^{-\ell}$.

Suppose that we have a $(2\ell+1)$ - digit accumulator available, with one digit position to the left of the decimal point. The analysis is similar to those given in Chapter III. Suppose $e_1 \le e_2$.

If $e_2 \ge e_1 + \ell + 1$, then $X \oplus Y - (X+Y) = -X$.

For $|e_2 - e_1| \le \ell$

1. Shift m_1

   Form $V_1 = m_1 b^{e_1 - e_2}$

   V_1 has $\ell + |e_1 - e_2|$ digits

2. Form V_2

   Add $|e_1 - e_2|$ non-significant zeros onto the end of m_2, thus $V_2 = m_2$.

   V_2 has $\ell + |e_1 - e_2|$ digits

3. Add V_1 and V_2

$$V_1 + V_2 = m_1 b^{e_1 - e_2} + m_2$$

(i) Carry

$$m_3 = b^{-1}(V_1 + V_2) + \varepsilon_1 b^{-\ell}$$

$$e_3 = e_2 + 1$$

$$X \oplus Y = X + Y + \varepsilon_1 b^{-\ell} b^{e_3}$$

(ii) No carry

$$m_3 = b^{j}(V_1 + V_2) + \varepsilon_2 b^{-\ell} \quad ; \; j = 0, 1, \dots, \ell$$

$$e_3 = e_2 - j$$

$$X \oplus Y = X + Y + \varepsilon_2 b^{-\ell} b^{e_3}$$

Error Bounds

(i) Carry $\quad |X \oplus Y - (X+Y)| \leq \varepsilon_A b^{-\ell} b^{e_3}$

(ii) No carry $\quad |X \oplus Y - (X+Y)| \leq \varepsilon_A b^{-\ell} b^{e_3}$

Advantages of this method

1. We avoid the rounding error in m_1 before the addition operation.
2. The final error bounds are explicitly in terms of the final exponent.
3. For this method we obtain

$$t = k b^{-\ell} .$$

This satisfies the requirement on t.

The machine language arithmetic subroutines, for use in conjunction with an automatic programming language, should be those with a $(2\ell+1)$ digit accumulator for addition-subtraction, a 2ℓ digit accumulator for multiplication, and an $(\ell+1)$ digit accumulator for division. We are lead to these conclusions before we have considered the choice of a rounding-off procedure. Let us now turn to the problem of choosing a round-off procedure.

In Chapter II we gave some examples of typical rounding-off methods: the most commonly used are

(1) the dropping of digits, and

(2) the addition of $(\frac{1}{2})_b$ in the last retained digit position.

We shall discuss this choice insofar as it affects the more significant problem of giving error estimates for a sequence of calculations. For example, if we are performing a repetitive calculation in which a round-off error at one step affects the results at successive steps we can give an estimate of the propagated error in terms of the rounding error at each step. The step errors are expressed as a combination (usually linear) of the individual arithmetic errors (the ε's). By using the above maximum error bounds for individual arithmetic operations we are able to obtain an error estimate at every step in a series of calculations. But this is a maximum error bound and is unsatisfactory because rarely does the propagated error become equal to its upper bound. It has been suggested that one give a statistical estimate for the propagated error: we would calculate an expected value and a variance. The validity of the statistical estimate ultimately rests on whether the ε's can be considered as independent random variables.

Huskey [7] gives an example where, for a particular arithmetic subroutine and rounding-off procedure, the errors systematically build up. Forsythe [3] has suggested that one could avoid effects similar to systematic round-off by adding random digits to the digit positions which are to be dropped. We shall examine such a procedure. Our results are somewhat different from those given by Forsythe.

Let m be the absolute value of a floating point mantissa $\bar{m}$ which we wish to round-off at the kth digit. Define a new number M such that

$$M = b^k m$$

Consider $M - [M] = V$

where $[M]$ is the greatest integer less than or equal to M. Obviously $0 \leq V < 1$; V consists of those digits of m which are to be dropped after rounding-off. We consider V to be the probability of rounding up (adding 1 into the kth digit of $\bar{m}$) and 1-V to be the probability of rounding down (dropping the k+1, k+2,... digits). Now suppose we add a random number $\eta, (0 \leq \eta < 1)$ into the k+1, k+2, ... digits of m. Let η be uniformly distributed on $[0,1)$. The error ε is now a random variable, and is given by

$$\varepsilon = [M] + 1 - M = 1 - V \qquad \text{(rounding up)}$$

$$\varepsilon = [M] - M = -V \qquad \text{(rounding down)}$$

The probability density for ε is

$$p(\varepsilon) = 1 - \varepsilon \qquad (\text{rounding up, } \varepsilon \geq 0)$$

$$p(\varepsilon) = 1 + \varepsilon \qquad (\text{rounding down, } \varepsilon \leq 0)$$

Expected value: $E(\varepsilon) = \int_{-1}^{0} \varepsilon(1+\varepsilon)\,d\varepsilon + \int_{0}^{1} \varepsilon(1-\varepsilon)\,d\varepsilon = 0$

Variance: $V(\varepsilon) = \int_{-1}^{0} \varepsilon^2(1+\varepsilon)\,d\varepsilon + \int_{0}^{1} \varepsilon^2(1-\varepsilon)\,d\varepsilon = \frac{1}{6}$

(The probability that $|\varepsilon| \leq \frac{1}{2}$ is 3/4)

A subroutine using random digits, while being theoretically ideal, would be, in practice, quite costly in computing time. The procedure by which we add $(\frac{1}{2})_b$ would be faster and may give just as good results. In the following chapter we present some experimental results that we obtained using different types of arithmetic subroutines; of these, two incorporate the special properties described in the first part of this chapter, and another two use random digits for rounding.

Chapter V

Our experimental work will be with arithmetic subroutines written in the IBM 1620 (machine number base is 10) machine language, and to be used with the automatic programming language, FORTRAN. Fortran uses an eight digit floating point mantissa. A later version of Fortran, FORTRAN 2, has the advantage of variable word length (2 to 28) for floating point mantissas.

In Table I we list the properties of the subroutines: Table II contains the execution times (in milli-seconds) of the subroutines.

Table I

Type of Fortran	Addition-Subtraction	Multiplication	Division
(1) FORTRAN 8 digits	Drops the low order digits on the smaller number before adding. If there is a carry, the lowest order digit is dropped. (9 digit accumulator)	Rounds to 8 significant digits by adding $\frac{1}{2}$ into the 8th place. (16 digit accumulator)	Forms a 9 digit quotient. Takes the most significant 8 digits. (9 digit accumulator)
(2)-ℓ where $\ell = 2,3,\ldots 28$ FORTRAN 2	As in (1) but variable ℓ	Takes the most significant ℓ digits of product. Drops the lowest order digits.	As in (1) but variable ℓ
(3) FORTRAN 8 digits	Subroutine is the special one described in Chapter IV. Rounds by adding $\frac{1}{2}$ into the 8th place.	Same as in (1) above.	Rounds by adding $\frac{1}{2}$ to the 8th significant digit of the quotient.
(4) FORTRAN 8 digits	Same as (3) above except that low order digits are dropped.	Retains only the most significant 8 digits; drops others.	As in (1) above.
(5) FORTRAN 8 digits	Rounds the smaller number by adding $\frac{1}{2}$ in the last retained digit. If carry, rounds result by adding $\frac{1}{2}$ into 8th digit.	Same as (1)	Same as (3)
(6) FORTRAN 8 digits	Same as (5) above except that we add random digits.	Same as (5) above except that we add random digits.	Same as (3)
(7) FORTRAN 8 digits	Same as (3) above except that we add random digits.	Same as (3) above except that we add random digits.	Same as (3)

The random digits used with subroutines (6) and (7) were obtained from a random number generator of the form $x_n \equiv 101\, x_{n-1} + C \pmod{10^{12}}$ (see [6]).

Table II

Subroutine	Addition-Subtraction Time	Multiplication Time	Division Time
(1)	8.29	15.842	50.518
(2)-8	9.512	16.572	49.514
(2)-9	9.778	19.868	59.712
(2)-10	10.044	23.530	70.950
(2)-12	10.576	31.952	96.546
(2)-16	11.640	53.188	160.218
(3)	12.80	15.842	55.906
(4)	11.85	15.042	50.518
(5)	11.44	15.842	55.906
(6)	15.322	20.462	55.906
(7)	17.121	21.392	55.906

Description of the Experiment

Choice of experimental problem was motivated by the interesting results observed by Huskey [7] in the integration of the system

$$\dot{x} = y$$

$$\dot{y} = -x$$

by the Heun method

$$\begin{aligned} x_j^* &= x_{j-1} + hy_{j-1} \\ y_j^* &= y_{j-1} + h(-x_{j-1}) \\ x_j &= x_{j-1} + \frac{h}{2}(y_{j-1} + y_j^*) \\ y_j &= y_{j-1} + \frac{h}{2}(-x_{j-1} + (-x_j^*)) \end{aligned}$$

over the range $.5200 \leq t \leq .5290$ with step size, h, of $2\text{x}10^{-5}$. Huskey found that the round-off errors systematically accumulated to such an extent as to contradict the assumption that the individual errors (the ϵ's) were independent random variables.

We performed a series of experiments which consisted of integrating the given system using the subroutines (1) to (7). At every fifth integration step the computed result for x was compared to a 25 digit result obtained with Fortran 2, using the same integration procedure. An error analysis of the method, by Rademacher [9] and quoted by Huskey [7], gives for the propagated error r_n,

$$E(r_n) = 0$$

$$\sigma(r_n) = a\sqrt{\frac{n}{3}}\,10^{-\ell} \qquad \text{uniform distribution}$$

$$\sigma(r_n) = \sqrt{\frac{n}{6}}\,10^{-\ell} \qquad \text{distribution for random digits}$$

assuming the individual rounding errors are independent random variables uniformly distributed between -a and a ;

n is the number of integration steps, and σ is the standard deviation of r_n.

Table III

Subroutine	Observed Results: Error at 450th step	Observed Results: Error Range x10^8	Observed Results: Error of Maximum Absolute Value Occurs at Step n	Theoretical Standard Deviation $\sigma(r_n)$
(1)	-223×10^{-8}	0,-223	450	12.2×10^{-8}
(2)-8	-223×10^{-8}	0,-223	450	12.2×10^{-8}
(2)-9	-25.6×10^{-8}	0,-25.6	450	1.22×10^{-8}
(2)-10	-2.85×10^{-8}	0,-2.85	450	$.122\times10^{-8}$
(2)-12	-223×10^{-12}	0,-.0223	450	12.2×10^{-12}
(2)-16	-225×10^{-16}	0,-225×10^{-8}	450	12.2×10^{-16}
(5)	-1.36×10^{-8}	2.05,-5.3	330	5.25×10^{-8}
(3)	-1.36×10^{-8}	2.05,-5.3	330	5.25×10^{-8}
(4)	-227×10^{-8}	0,-227	450	12.2×10^{-8}
				at n=450
(6)	-	3.17,-11.9	-	8.66×10^{-8}
(7)	-	10.6,-8.8	-	8.66×10^{-8}

Figure I

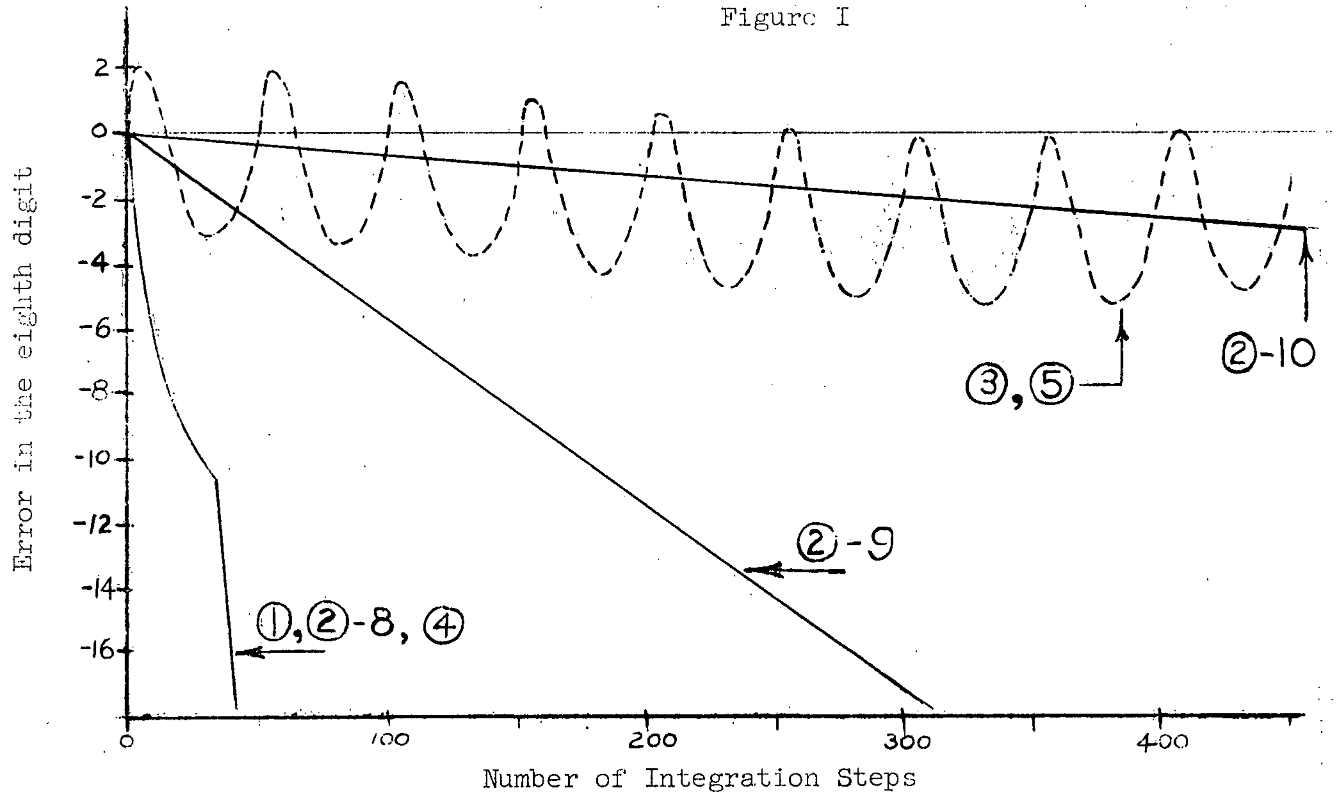
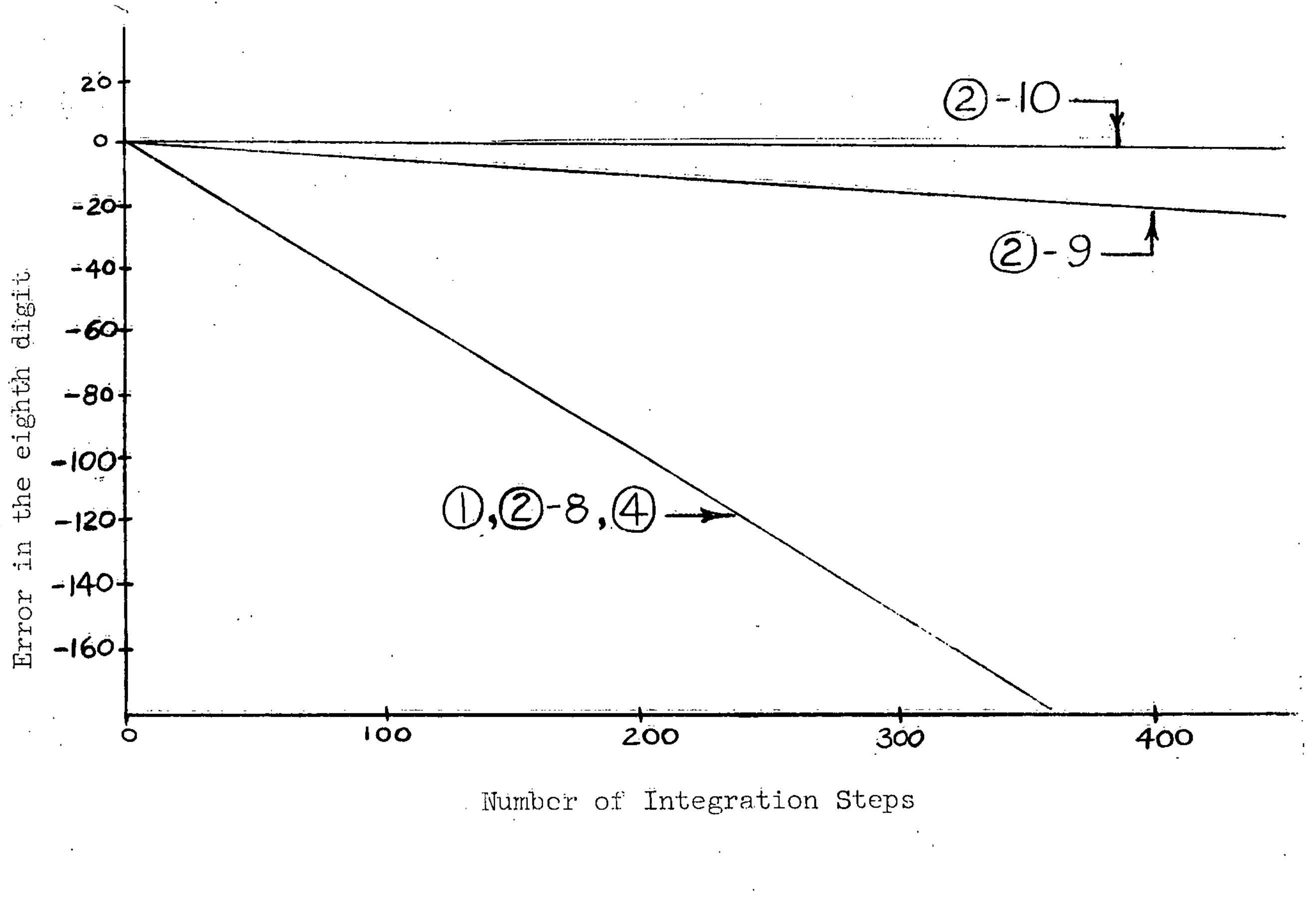


Figure II



Comments on Experimental Results

Relevant experimental results appear in Table III and in Figures I and II. Figures I and II are graphs of the observed errors,

$$r_n = (\ell\text{-digit result after n steps}) - (25 \text{ digit result after n steps})$$

1. Rounding by dropping digits (see Figures I and II).

   We obtained almost identical results with subroutines (1), (4) and (2)-8; all three exhibited the systematic accumulation of round-off error. In their construction subroutines (1) and (2)-8 differ only in the multiplication operation: a computed result in (1) is rounded by adding $\frac{1}{2}$ instead of dropping the lower-order digits. This indicates that the dominant contribution to rounding error accumulation was due to the addition operation. The use of special subroutine (4) did not give us any advantage over the usual subroutines.

2. Rounding by adding $\frac{1}{2}$ (see Figure I).

   Subroutine (3) (of the type described in Chapter IV) and subroutine (5) produced identical results in which systematic round-off was absent. Of the two routines, (3) is the better since it has the lower value of the "t" parameter (see Chapter IV).

   Comparing the results of (3) and (5) with those obtained with the corresponding routines (4) and (2)-8, which round by dropping digits, we conclude that the elimination of systematic round-off in the former results was due to the rounding-off process of adding $\frac{1}{2}$. The choice of round-off process is obviously critical.

3. Rounding by adding random digits.

   We performed several experiments with different initial values x_0 and constants c for the random number generator $X_{n+1} \equiv 101X_n + c \pmod{10^{12}}$. The maximum absolute value of observed error was 11.9×10^{-8} and 10.6×10^{-8}

for subroutines (6) and (7) respectively. In all cases, neither routine gave results which exhibited the systematic round-off.

Further experimental work with other types of problems is necessary to fully evaluate these subroutines.

4. Longer word lengths (see Figures I and II).

Subroutines (2)-9, (2)-10, (2)-12, and (2)-16 all exhibited systematic round-off. The results obtained with the 8-digit routines, (3) and (5), were generally better than those of (2)-9 and (2)-10. This seems to indicate that by carrying one or two extra digits throughout the calculations we may not counteract the effects of such a poor round-off process as the dropping of the lower order digits.

Hartree [7] attributes systematic round-off to the fact that the "leading digit rounded off remains the same in a number of successive contributions to the integral"; on this basis he develops a criterion to determine whether systematic round-off is likely to occur. According to his analysis, for step size of $2\text{x}10^{-5}$ and $.52 \leq t \leq .529$, we might avoid systematic round-off if we take the word length ℓ greater than 12. Our experimental results showed a systematic accumulation of round-off error for both the 12 digit and 16 digit word lengths (subroutines (2)-12 and (2)-16). We conclude that Hartree's analysis is not sufficient to account for the systematic build-up of errors. The effect is due to the type of round-off process (dropping digits).

5. Statistical estimation of errors.

Table III shows that the theoretical standard deviation, $\sigma(r_n)$, does not give an accurate error estimate for those subroutines which rounded by dropping digits (the maximum observed error was larger by a factor of approximately 18). On the other hand, the statistical estimate was sufficiently precise for the routines which rounded by adding $\frac{1}{2}$ or by adding random digits. This shows that a statistical treatment of round-off error can give reasonable error estimations provided the round-off process is accurate enough.

Chapter VI

Conclusions

Our experimental results emphasize the fact that a designer of an arithmetic subroutine should approximate a real arithmetic operation as closely as possible. The form of subroutine best fitting these requirements was the one which utilized a double precision (2ℓ-digits) product area, an (ℓ +1)-digit quotient area, and a (2ℓ +1)-digit area for sums and differences(*), in conjunction with a rounding-off process of adding $\frac{1}{2}$ into the last digit position retained.

The systematic accumulation of round-off error was found to be caused by the type of rounding-off process: the effect was observed in results obtained with those subroutines which rounded by dropping digits, but did not occur with routines that used $\frac{1}{2}$ or random digits for rounding. Statistical methods gave us accurate error estimates for data obtained with the latter subroutines. As a consequence, we conclude that statistical methods may be applied to the propagation of round-off error provided the arithmetic subroutines are sufficiently accurate.

(*) As described in Chapter IV.

Bibliography

1. Ashenhurst, R.L., and Metropolis, N.,
"Unnormalized Floating-point Arithmetic",
J. Assoc. C.M. 6 (1959), 415-428.

2. Carr, J.W., III.
"Error Analysis of Floating-point Arithmetic",
Comm. of A.C.M. 2 (1959), 10-15.

3. Forsythe, G.E.,
"Reprint of a Note on Rounding-off Errors",
S.I.A.M. Review 1 (1959), 66-67.

4. Henrici, P.,
Discrete Variable Methods in Ordinary Differential Equations,
J. Wiley & Sons, New York, 1962.

5. Householder, A.S.,
"Generation of Errors in Digital Computation",
Bull.Amer.Math.Soc. 60 (1954), 234-249.

6. Hull, T.E., and Dobell, A.R.,
"Random Number Generators",
S.I.A.M. Review 4 (1962), 230-254.

7. Huskey, H.D.,
"On the Precision of a Certain Procedure of Numerical Integration",
With an appendix by D.R. Hartree,
J. Research of Nat. Bur. of Stand. 42 , (1949), 57-62.

8. von Neumann, J., and Goldstine, H.H.,
"Numerical Inverting of Matrices of High Order",
Bull.Amer.Math.Soc. 53 (1947), 1021-1099.

9. Rademacher, H.,
"On the Accumulation of Errors in Processes of Integration on High-Speed Calculating Machines",
Annals Comput. Labor. Harvard Univ. 16, (1948), 176-187.

Bibliography - cont:

10. Richards, R.K.,
Arithmetic Operations in Digital Computers,
van Nostrand Co., Inc., 1955.

11. Wilkinson, J.H.,
"Error Analysis of Floating Point Computation"
Num.Math. 2 (1960), 319-340.