Analysis of Rounding Methods in Floating-Point Arithmetic

DAVID J. KUCK, MEMBER, IEEE, DOUGLASS S. PARKER, JR., AND AHMED H. SAMEH

Abstract—**The error properties of floating-point arithmetic using various rounding methods (including ROM rounding, a new scheme) are analyzed. Guard digits are explained, and the rounding schemes' effectiveness are evaluated and compared.**

Index Terms—**Floating-point arithmetic, guard digits, ROM rounding, rounding methods.**

I. Introduction

EXISTING computer arithmetic units use various methods to reduce floating-point numbers to shorter precision approximations. The standard methods in use are 1) ordinary *Truncation* (or "chopping"); 2) ordinary *Rounding*. Other methods have been proposed for their cost-effectiveness or for their error-resistance properties, e.g., 3) *Jamming* (equivalent to truncation except that the lowest order bit of the result mantissa is forced to a 1); 4) *R*-rounding* (equivalent to rounding, except when the digits to be rounded away have bit values 10 000 $\cdots$, in which case Jamming is used).

This paper addresses the problem of analyzing the arithmetic behavior of these and other methods when used in floating-point computation. The paper is an outgrowth of a recent study [6] comparing these methods with *ROM rounding*, an attractive alternative both from a standpoint of cost and error resistance. With ROM rounding the (say) 7 low-order bits of the significant part of the mantissa and the highest bit to be rounded away, together form an 8-bit input ("address") to a ROM or PLA. The output of the ROM/PLA is the 7-bit rounded value of these bits, except in the threshold case when all 7 low bits of the significant part of the mantissa are 1—the case in which rounding overflow might occur—rounding is not performed. This is illustrated in Fig. 1.

Note that ROM rounding provides the advantages of rounding (255 times out of 256 for the 8-bit scheme just mentioned) without the timing or hardware overhead of the traditional additional Rounding adder stage. It is also an inexpensive scheme to implement in modern technologies. Thus, from an engineering standpoint, ROM rounding is an appealing way to build rounded arithmetic into floating-point units where timing and/or gate limitations already exist in the design. If $R(x)$ denotes the ROM-rounded value of the number x, then we have the relationships $R(x) \leq R(y)$ iff $x \leq y$, and $R(-x) = -R(x)$ if floating-point numbers are represented in the usual sign/magnitude format. However, although the method passes these two elementary tests, we must point out that the "glitch" caused by the threshold case is mathematically unpleasant. As a result, we cannot recommend ROM rounding to the designer of a highest quality arithmetic unit because it will fail, in a few cases, to satisfy arithmetic identities like the ones provable for pure Rounding [13, sec. 4.2.2]. Nevertheless it seems to perform well and should be useful in machines where pure Rounding or R*-rounding is not feasible. It could certainly be an improvement over some of the methods developed to avoid rounding carry-out on most existing computers; very very few machines implement rounding correctly.

It should be noted that an extra bit could be read from the ROM to distinguish the 1 $\cdots$ 1 (threshold) case from all others. In this one case, the adder could be used for standard Rounding. We could therefore achieve Rounding at high speed in most cases (255 times out of 256 for the scheme just mentioned), and at ordinary Rounding speed in one case. Similarly, by using a sticky bit (see below) we could achieve fast R*-rounding via a ROM. Of course, handling the 1 $\cdots$ 1 case would cause additional logic design complexities over ROM-rounding, as previously defined and discussed throughout the paper.

Whereas previously we were concerned with examining the effectiveness of ROM-rounding, this paper will concentrate more on the evaluation of any given rounding method's expected performance in floating-point computation. Generally speaking, a rounding method will perform well if the magnitude of its rounding errors are small and if it is unbiased, i.e., does not favor errors of any particular sign. (The latter condition is complicated, as will be shown in Section IV.) This paper examines the methods under consideration with respect to three indicators of rounding scheme effectiveness, viz., 1) average relative representation error (ARRE); 2) rounding scheme bias; 3) statistical tests in actual computation. The first two of these are "static" performance metrics while the third is "dynamic;" logically, static indicators should predict dynamic error performance to some degree.

II. Representation Error

A theoretical metric of rounding scheme effectiveness is the ARRE incurred by the scheme in representing real numbers in floating-point format. McKeeman [8] and later Cody [2] used this metric for comparison between base 2,

Manuscript received January 20, 1976; revised February 1, 1977. This work was supported in part by National Science Foundation Grant DCR73-07980 A02 and in part by National Science Foundation Grant MCS75-21758.

D. J. Kuck and D. S. Parker, Jr., are with the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

A. H. Sameh is with the Department of Computer Science and Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

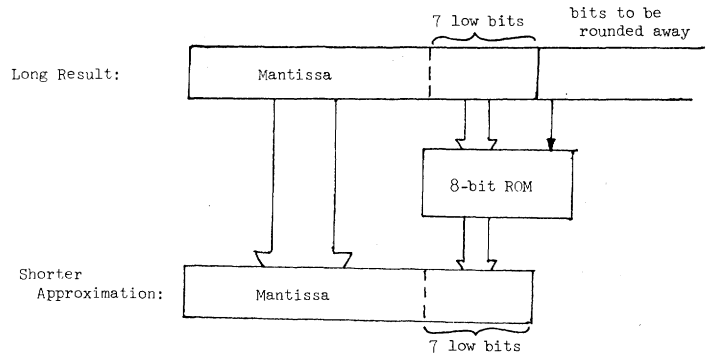


Fig. 1. Eight-bit ROM rounding.

4, and 16 arithmetic systems. They defined the ARRE of a t-digit, base β, sign-magnitude system using rounding by

$$\mathrm{ARRE}(t,\beta) = \int_{1/\beta}^{1} \left(\frac{\beta^{-t}}{4x}\right) \frac{dx}{x \ln \beta} = \left(\frac{\beta^{-t}(\beta-1)}{4 \ln \beta}\right) \quad (2.1)$$

where $(1/x \ln \beta)$, the reciprocal density, has been used because of the logarithmic law (of leading digits in floating-point computation, see e.g., [11]) and

$$\left(\frac{\beta^{-t}}{4x}\right) = \frac{\underset{1/\beta < x < 1}{\text{average}} (|\mathrm{Round}(x,t) - x|)}{x}$$

is used as an approximation to the absolute value of the relative error

$$|\delta(x)| = \left|\frac{\mathrm{Round}(x,t) - x}{x}\right|.$$

Here $\mathrm{Round}(x,t)$ is the value of x rounded to t radix-β digits. With this in mind, we can *redefine* for any rounding scheme $R(x,t)$

$$\mathrm{ARRE}(t,\beta) = \int_{1/\beta}^{1} |\delta(x)| \frac{dx}{x \ln \beta}. \quad (2.2)$$

This quantity was evaluated for each of the five rounding schemes (via a 20 000-point Monte Carlo integration) using $\beta = 16$ and various values of t. The results agreed excellently with the theoretical values for Truncation and Rounding.

Tsao [11] gives the probability densities for δ as

$$P_{\mathrm{Truncation}}(\delta = \delta_0) = \begin{cases} \beta^{t-1}(\beta-1)/\ln \beta, & -\beta^{-t} < \delta_0 \leqq 0 \\ \left(\frac{1}{\delta_0} - \beta^{t-1}\right)\Big/ \ln \beta, & -\beta^{1-t} < \delta_0 \leqq -\beta^{-t} \end{cases} \quad (2.3)$$

$$P_{\mathrm{Rounding}}(\delta = \delta_0) = \begin{cases} \beta^{t-1}(\beta-1)/\ln \beta, & |\delta_0| \leqq \beta^{-t}/2 \\ \left(\frac{1}{2|\delta_0|} - \beta^{t-1}\right)\Big/ \ln \beta, & \beta^{-t}/2 < |\delta_0| < \beta^{1-t}/2. \end{cases} \quad (2.4)$$

Changing these to reflect $|\delta|$ and integrating we expect

$$\mathrm{ARRE}(t,\beta)_{\mathrm{Truncation}} = \frac{\beta^{-t}(\beta-1)}{2 \ln \beta} \quad (2.5)$$

$$\mathrm{ARRE}(t,\beta)_{\mathrm{Rounding}} = \frac{\beta^{-t-1}(\beta^2-1)}{4 \ln \beta}. \quad (2.6)$$

Note that Rounding is optimal for ARRE since Rounding(x,t) is always as close to x as possible in a t-digit system.

The Monte Carlo results coincided well with (2.5) and (2.6), showing that ARRE[Rounding] ≡ ARRE[R*] ≈ ARRE[8-bit ROM] and ARRE[Truncation] = ARRE[Jamming], so that ARRE effectively dichotomizes the methods under consideration. The 8-bit ROM got ARRE values only marginally worse (third significant digit) than Rounding. This is reasonable not only because the two are identical 255 times out of 256, but also because the one case in which they differ (inputs to the ROM are all 1) has the lowest individual relative error. Clearly, rounding with a smaller ROM length will give a larger ARRE value.

Note that because of the glitch in ROM-rounding, however, the worst case relative error bounds dichotomize the methods differently: WCB[rounding] = WCB[R*-rounding] = $\frac{1}{2}\ \beta^{1-t}$, and WCB[ROM-rounding] = WCB[Truncation] = WCB[Jamming] = β^{1-t}. For this reason, we cannot recommend ROM-rounding to the numerical analyst interested in *a priori* error bounds. The point is, though, that the worst case bound is rarely reached, particularly when the ROM length grows large.

III. Guard Digits

Before we can approach the subject of bias, an understanding of guard digits is needed. Guard digits are temporary low-order registers in the floating-point adder hardware that hold (at least some of) the digits of the aligned operand when they are shifted right. As long as alignment shift distance does not exceed the number of guard digits, a single-precision adder will give the same results obtainable with an infinitely long accumulator rounded down to single precision. In fact, given a few guard bits it has been shown ([4]) that Wilkinson's "double-

precision accumulator bounds" on roundoff errors δ [12]

$$-\beta^{1-t} \leqq \delta \leqq 0 \quad \text{for Truncation}$$

$$-\frac{1}{2}\beta^{1-t} \leqq \delta \leqq \frac{1}{2}\beta^{1-t} \quad \text{for Rounding}$$

almost hold for a single-precision unit. Note that without guard digits the difference 1.000 − 0.9999 (or any similar computation) cannot be carried out without serious loss of accuracy and huge relative error.

We can state exactly when guard digits are useful:

A) *Addition* (like operand signs): One guard rounding bit is necessary and sufficient, since other guard bits could never affect the sum (except when R*-rounding is used).

B) *Subtraction* (unlike operand signs): With Truncation, arbitrarily many guard digits could be necessary because alignment shifts can be arbitrarily large, and if any shift occurs some "borrowing" must occur out of the low-order digit. With Rounding, however, it can be shown that a guard digit and a guard-rounding bit are adequate, because any borrowing caused by alignment is offset by the final round. We discuss this in depth below.

It is difficult to make general statements about the use of guard digits in multiplication since the requirements obviously vary according to the way the multiplier works. If the multiplier simply adds up partial products in a straightforward manner, the comments previously made for addition will hold. Other schemes may have more complicated guard-digit requirements. Note that it is not clear how to implement R*-rounding without forming a full double-length product; however, we will show later that R*-rounding is of questionable use in multiplication.

The problem introduced by subtraction for Truncation may be eliminated by use of a *sticky bit* (or *trap bit*). The sticky bit is a guard bit which remains at a 1 value if a 1 is shifted into or through it during alignment. It removes the borrowing problem and makes it easy to show that one guard digit, followed by a guard-rounding bit or sticky bit with Rounding or Truncation, respectively, are sufficient for accurate subtraction. Thus one guard digit and another guard bit of some kind are sufficient for the operations in the single-precision unit to obtain the same results as an infinitely long accumulator rounded to single precision. This observation permits us to bypass infinite guard-digit analyses like that in [4].

We can now clarify the use of guard digits in subtraction. Note that in subtraction, one of three possible things can occur: either 1) there is *no* alignment shift, in which case the result of the subtraction is the exact difference of the two operands and no guard digits are needed; 2) an alignment shift of one digit occurs; or 3) an alignment shift or more than one digit occurs. In the second or third cases, some "borrowing" must be done, possibly causing a cancellation of high-order digits; this would necessitate a realigning left shift of the intermediate difference (and the guard digits). Surprisingly, one guard digit and one guard-rounding bit or one sticky bit with Rounding or Truncation, respectively, are sufficient for correct computation in either of these two cases. This is obvious in case 2), where an alignment shift of only one digit occurs. In case 3), one can easily verify that because there are two or more alignment shifts, only the highest order digit can be cancelled out; i.e., at most one left realignment shift can occur. Thus one guard digit is basically sufficient. In the case of Truncation, a sticky bit is also necessary: since arbitrarily large alignment shifts can occur, we naively need arbitrarily many guard digits for correct handling of borrowing. Fortunately, the sticky bit can be used instead. In the case of rounding, the rounding bit is necessary for the case where a left realignment shift occurs. Here, however, the sticky bit is not needed because rounding will offset (cancel) any borrowing engendered by a sticky bit. Thus, as stated, one guard digit and a guard-rounding bit are sufficient to produce "infinitely long accumulator" results.

We should point out that the sticky bit can be used in both addition and subtraction for the implementation of R*-rounding. We omit the details here, but it is clear that the sticky bit is an ideal way to monitor the case in which the bits to be rounded away are exactly $1000\cdots$.

It is interesting to know how often, on the average, guard digits would be used if we were to build a machine that used several guard digits instead of a sticky bit. Statistics on alignment shifts may be found in [10, p. 39]. From this we can obtain Table I, giving probabilities that alignment shifts of $\leqq k$ digits will occur. The table shows us that even without a sticky bit, 4 guard bits will give us infinitely long accumulator results 70 percent of the time, and 8 guard bits 82 percent of the time. Unfortunately, the remaining 18 percent requires arbitrarily many guard digits, as we observed earlier.

As a final comment on guard digits, we would like to observe that seemingly higher accuracy could easily be obtained if the guard digits were *retained* in the result register for use in further computation until the result was to be stored in memory (at which time the guard bits would be rounded away). Just from the standpoint of Wilkinson's bounds, retention of g guard bits causes a factor of 2^{-g} decrease in accumulated error. The main arguments against this sort of implementation seem to be that few people will understand it and that it produces extremely compiler-dependent results (storing temporaries becomes critical), but in the hands of an intelligent programmer this feature could be useful (see [17]).

IV. Rounding-Scheme Bias

While ARRE measures the average magnitude of roundoff errors, we are also concerned with measuring the tendency of a rounding scheme to favor errors of a particular sign. Generally speaking, we would like roundoff errors incurred in floating-point addition and multiplication to be unbiased; i.e., have an average value of zero. For mul-

TABLE I
Floating-Point Addition and Subtraction: Cumulative Probability that Alignment Shift of $\leq k$ Digits Occurs

k \ Radix	2	4	8	16	
0	.3264	.3824	.4577	.4732	
1	.4475	.5678	.6554	.7334	4 Guard bits
2	.5336	.6961	.7746	.8327	8 Guard bits
3	.6008	.7948	.8372		
4	.6725	.8252			
5	.7113				
6	.7552				
7	.8034				
8	.8163				

tiplication this usually[1] means that the rounding method used $R(x,t)$ must be unbiased since, if we are working with t digits

$$fl(a \cdot b) = R(a \cdot b,t) \tag{4.1}$$

i.e., the only error is made during rounding. For addition with g guard digits, however, we have

$$fl(a + b) = R(\text{renormalize}(a + \text{align}(b,t + g),t + g),t)$$

so we must study alignment roundoff to completely understand bias. The importance of implementing guard digits as in Section III is that it guarantees

$$fl(a + b) = R(a + b,t) \tag{4.2}$$

so that roundoff errors in addition are just the errors made in rounding, and that roundoff bias in addition is exclusively due to the rounding method used. Because of its desirable arithmetic properties and its simplification of our analysis (eliminating in one stroke all concern with the number of guard digits used and the properties of alignment roundoff) we will henceforth assume the Section III guard-digit recommendation holds in all operations.

We can now concentrate on the analysis of rounding scheme bias, by measuring the average relative error $\bar{\delta}$ incurred by a rounding scheme in floating-point computation. For an unbiased scheme $\bar{\delta}$ will be zero; for slightly biased schemes like Rounding $\bar{\delta}$ will be small; and for schemes like Truncation where errors are always of the same sign, $\bar{\delta}$ is approximately the ARRE for that scheme. We formalize this in the following pages.

If y is the random variable assigning to the mantissas in $[1/\beta,1)$ a probability of occurrence as a result of some computation with t-digit floating-point operands, then $\bar{\delta}$ is the expected value

$$\bar{\delta} = E\left[\frac{R(y,t) - y}{y}\right]. \tag{4.3}$$

[1] An exception being, for example, double-precision multiplication on the IBM 360/75, which truncates partial products and, therefore, creates error before rounding. Many double-precision units violate the assumption that all error is incurred in rounding.

We know from [4], [11], and elsewhere that y must be more or less logarithmically distributed if (4.3) is to be realistic for floating-point computation. Brent [1] points out that the logarithmic law is only an approximation, but is a considerably better approximation to the distribution of mantissas than the assumption of uniformity. Note that y will be of finite precision, since it is the result variable of finite precision calculation (this will be discussed more fully later in this section), but its general logarithmic distribution is enough to guarantee for reasonable t that the numerator and denominator of (4.3) are virtually uncorrelated. Thus we can make the simplification

$$\bar{\delta} = E[R(y,t) - y] \cdot E[1/y]. \tag{4.4}$$

We call the first factor on the right-hand side of (4.4) the *rounding bias* of the scheme R. The second factor is the average reciprocal mantissa value; using the log distribution we can obtain its theoretical estimate

$$E[1/y] = \int_{1/\beta}^{1} \frac{1}{y} \frac{dy}{y \ln \beta} = \left(\frac{\beta - 1}{\ln \beta}\right). \tag{4.5}$$

All we need now to estimate $\bar{\delta}$ is an approximation to the rounding bias. We begin with the radix 2 case: Let $M(t + s)$ be the set of all normalized binary mantissas of length $t + s$, which are to be represented in $M(t)$; i.e., rounded to t bits. For any rounding scheme $R(x,t)$ mapping $M(t + s)$ into $M(t)$ we can define

$$\text{average bias}_{\text{base }2}(R,t,s) = \frac{\sum_{x \in M(t+s)} R(x,t) - x}{||M(t + s)||} \tag{4.6}$$

where $||X||$ is the number of elements in the set X. From (4.6) we can derive the following list:

$$\text{average bias (Truncation},t,s) = \frac{1}{2} 2^{-t}(2^{-s} - 1), \qquad s \geq 0$$

$$\text{average bias (Jamming},t,s) = \frac{1}{2} 2^{-t}(2^{-s}), \qquad s \geq 0$$

$$\text{average bias (Rounding},t,s) = \frac{1}{2} 2^{-t}(2^{-s}), \qquad s > 0$$

$$\text{average bias } (R^*,t,s) = 0, \qquad s \geqq 0$$

$$\text{average bias } (\text{ROM}(l),t,s) = \frac{1}{2}2^{-t}(2^{-s} - 2^{1-l}), \qquad s > 0 \tag{4.7}$$

where l is the ROM length in bits; $l < t + 1$. The main argument to be lodged against (4.6) is that it does not take the logarithmic law into account. However, Field demonstrates [3, Appendix A] that when the radix involved is 2, the uniform and logarithmic distributions on $M(t + s)$ are almost equal. The agreement is good enough that (4.6) should be accurate in predicting bias for radix 2. The question that arises is how to extend (4.6) for arbitrary radix β, since as the radix increases the effects of the logarithmic law become more and more pronounced.

The extension may be achieved for any radix $\beta = 2^n$ as follows. Since normalized t-digit radix-β mantissas are also (nt)-digit radix-2 mantissas having 0 to $n - 1$ leading zeroes, we can write

$$\text{average bias}_{\text{base }\beta}(R,t,s) = \sum_{k=0}^{n-1} (q_k)\text{average bias}_{\text{base }2}(R,nt,ns - k)$$

where q_k is the probability that any mantissa has exactly k leading zeroes. In effect, we break up the analysis on $[1/\beta,1)$ into analysis on $[1/\beta,2/\beta),[2/\beta,4/\beta),\cdots,[1/2,1)$. Field's result allows us to compute the bias on any of these subintervals (it is translation independent), while the q_k allow for the log weighting of the logarithmic law. In fact,

$$q_k = \Pr(2^{-k-1} < x < 2^{-k}, \quad x \text{ selected randomly from all mantissas})$$

$$= \int_{2^{-k-1}}^{2^{-k}} \frac{dx}{x \ln \beta}$$

$$= 1/n, \qquad n = \log_2 \beta$$

so

$$\text{average bias}_{\text{base }\beta}(R,t,s) = \sum_{k=0}^{n-1} \left(\frac{1}{n}\right) \text{average bias}_{\text{base }2}(R,nt,ns - k). \tag{4.8}$$

Thus we can compile Table II for any radix $\beta = 2^n$. Here s and t refer to the number of radix-β digits being used, whereas l is the ROM length in bits $(1 < l < t/n + 1)$.

As a way of testing this, we note that (2.2) and (4.4) imply

$$\text{ARRE}(t,\beta)_{\text{Truncation}} = |\text{average bias}_{\text{base }\beta}(\text{Truncation},t,\infty)| \cdot \left(\frac{\beta - 1}{\ln \beta}\right). \tag{4.9}$$

Using (2.5) and Table II, we see that equality does, in fact, hold. This is in one sense a stroke of luck, since our average bias is only an approximation to the rounding bias $E[R(y,t) - y]$. However, it is apparently a good approximation.

We can now turn to the analysis of $\bar{\delta}$ for floating-point multiplication. If $M(\beta,t)$ denotes the set of all normalized t-digit radix-β mantissas, then given any two operands in $M(\beta,t)$ their product will be in $M(\beta,t + t)$. Thus the result variable y in the bias $E[R(y,t) - y]$ for floating-point multiplication ranges "logarithmically" over $M(\beta,t + t)$, and we can use (4.4), (4.5), and Table II with $s = t$ to compute the average multiplicative roundoff error $\bar{\delta}$. This is done in Table III for each of the schemes under consideration.

TABLE II
Average Bias

Rounding Scheme R	Average Bias (R,t,s) with base β normalized arithmetic	
R*	0	$s \geq 0$
Rounding	$\beta^{-t-s}(\beta-1)/2\log_2\beta$	$s > 0$
Jamming	$\beta^{-t-s}(\beta-1)/2\log_2\beta$	$s \geq 0$
ROM(ℓ)	$\beta^{-t-s}(\beta-1)/2\log_2\beta - \beta^{-t}2^{-\ell}$	$s > 0$
Truncation	$\beta^{-t-s}(\beta-1)/2\log_2\beta - \beta^{-t}/2$	$s > 0$

We point out that although R*-rounding has an average roundoff 0, it is expensive to implement since it requires knowing the values of all of the lower t digits of the product. These almost *never* have bit values $1000\cdots$; it would seem that the effort required to check for these bit values is not worth going through, and that ordinary rounding could be used just as effectively.

We now consider the analysis of floating-point addition (and subtraction). Addition is much more complicated than multiplication because the sum of two numbers with mantissas in $M(\beta,t)$ can be in $M(\beta,t + s)$ for any $s \geqq 0$. The value of s is dependent on the difference of the exponents associated with the numbers (i.e., the alignment-shift value) and the number of renormalization shifts required by carry-out or cancellation. To evaluate $E[R(y,t) - y]$ as in (4.6), where y is the result of an addition, we must know the probability that a given value of s will occur. Specifically, if P_s is the (radix-dependent) probability that the result variable y is in $M(\beta,t + s)$, then

$$E[R(y,t) - y] \simeq \sum_{s=0}^{\infty} P_s \cdot \text{average bias}(R,t,s). \tag{4.10}$$

Noting that average bias $(R,t,0) = 0$ for all schemes here except R = jamming, if we define

$$P_\sigma = \sum_{s=1}^{\infty} P_s$$

$$P_\tau = \sum_{s=1}^{\infty} P_s\beta^{-s}$$

then we can use Table II, (4.5), and (4.10) to get the average addition roundoff errors in Table IV.

Naturally, the numerical values for the probabilities P_s will depend on the application under consideration; i.e., since the P_s values rely on alignment and renormalization shift probabilities, the values of P_s (and hence of $\bar{\delta}$) are

TABLE III
Multiplication Rounding-Error Performance

Rank	Scheme	Average Multiplicative Roundoff $\bar{\delta}$
1	R*	0
2	Rounding, Jamming	$\frac{\beta^{-2t}}{2\ln 2}\left(\frac{\beta-1}{\log_2\beta}\right)^2$
3	ROM(ℓ)	$\sim \frac{-\beta^{-t}\,2^{-\ell}(\beta-1)}{\ell n\beta}$
4	Truncation	$\sim \frac{-\beta^{-t}(\beta-1)}{2\ell n\beta}$

TABLE IV
Addition Roundoff Performance

Rank	Scheme	Average Addition Roundoff $\bar{\delta}$
1	R*	0
2	ROM(ℓ)	$\frac{\beta^{-t}(\beta-1)}{\ell n\beta}\left(\frac{(\beta-1)}{2\log_2\beta}P_\tau - 2^{-\ell}P_\sigma\right)$
3	Rounding	$\frac{\beta^{-t}}{2\ln 2}\left(\frac{\beta-1}{\log_2\beta}\right)^2 P_\tau$
4	Jamming	$\frac{\beta^{-t}}{2\ln 2}\left(\frac{\beta-1}{\log_2\beta}\right)^2 (P_0 + P_\tau)$
5	Truncation	$\frac{\beta^{-t}(\beta-1)}{\ell n\beta}\left(\frac{(\beta-1)}{2\log_2\beta}P_\tau - \frac{1}{2}P_\sigma\right)$

dependent on how often addition/subtraction operands have exponents close to, or far away from, each other. Thus error performance will vary from one type of program to another, and knowledge of shift probabilities is not sufficient to predict this variation.

We can make some general statements for radix-2 addition, however. From examining over a million floating additions drawn from twelve problems, Sweeney compiled statistics on alignment shifts [10]. These results have already been presented in Table I and show that most of the additions in the representative programs had exponent differences near zero, especially as the radix became large. Unfortunately, Sweeney did not include data on the *conditional* distribution of renormalization shifts. This has been done for radix-2 by Field [3] and allows us to compute the probability that y is in $M(2,t+s)$ via the relationship

$$P_s \overset{\text{def}}{=} \Pr\,(y \in M(2,t+s))$$

$$= \sum_{k=-1}^{\infty} \Pr\begin{pmatrix}\text{alignment right}\\ \text{shift} = s+k\end{pmatrix} \cdot \Pr\left(\begin{array}{l|l}\text{renorm left} & \text{align right}\\ \text{shift} = k & \text{shift} = s+k\end{array}\right). \tag{4.11}$$

We can tabulate the values of P_s for radix-2 defined in (4.11). Table V is computed directly from [10, table 3] and [3, table 1]; the "average" reflects Sweeney's result that 52.73 percent of his additions involved unlike signs, while 47.27 percent involved like signs.

In Table IV, ROM rounding is ranked above rounding on the assumption that l is large enough to ensure cancellation in the factor involving P_σ and P_τ. If we use the average values of P_s in Table V, we discover $P_0 \approx 0.031$, $P_\sigma \approx 0.896$, $P_\tau \approx 0.135$, so that choosing the ROM length $l = 4$ gives $\bar{\delta} \approx (0.017) \times 2^{-t}$. This is considerably better than the $(0.097) \times 2^{-t}$ result for Rounding. Unfortunately, it is difficult to estimate the optimal ROM length l for a larger radix because detailed statistics, like those for radix-2 in [3] and [10], are, to our knowledge, currently unavailable.

TABLE V
P_s for Radix-2 Addition (In Percent)

s	P_s(like signs)*100	P_s(unlike signs)*100	P_s(average)*100
0	0.000	5.898	3.110
1	30.268	6.391	17.678
2	16.414	7.031	11.466
3	8.085	6.460	7.228
4	6.860	8.549	7.750
5	9.685	6.536	8.025
6	6.335	4.265	5.244
7	4.030	3.606	3.806
8	0.971	1.248	1.117
9	1.271	1.194	1.230
10	1.989	1.705	1.840
11	1.335	1.365	1.351
12	1.756	1.346	1.540
13	0.826	0.796	0.810
14	0.254	0.683	0.480
15	0.762	0.417	0.580
16	0.212	0.360	0.290
17	0.212	0.398	0.310
18	0.592	0.417	0.500
19	0.106	0.512	0.320
20	0.127	0.379	0.260
21	0.296	0.493	0.400
22	0.254	0.341	0.300
23	0.148	0.322	0.240
24	0.127	0.360	0.250
25	0.106	0.398	0.260

It should be emphasized that Sweeney's alignment-shift statistics were averaged over twelve problems and a million addition operations, and that by selecting a rounding scheme on the basis of Table V we are not guaranteeing good error performance for a specific application, but rather are choosing a scheme that should perform well "over the years." Kent [5] points out that, although Sweeney's averages hold over a collection of programs, they will not, in general, hold for any specific program. Therefore, while we are predicting roundoff performance over many programs, we cannot say *a priori* how a given rounding method will behave on a given program. Indeed, programs can be written to force any known rounding scheme to give worst case bound results.

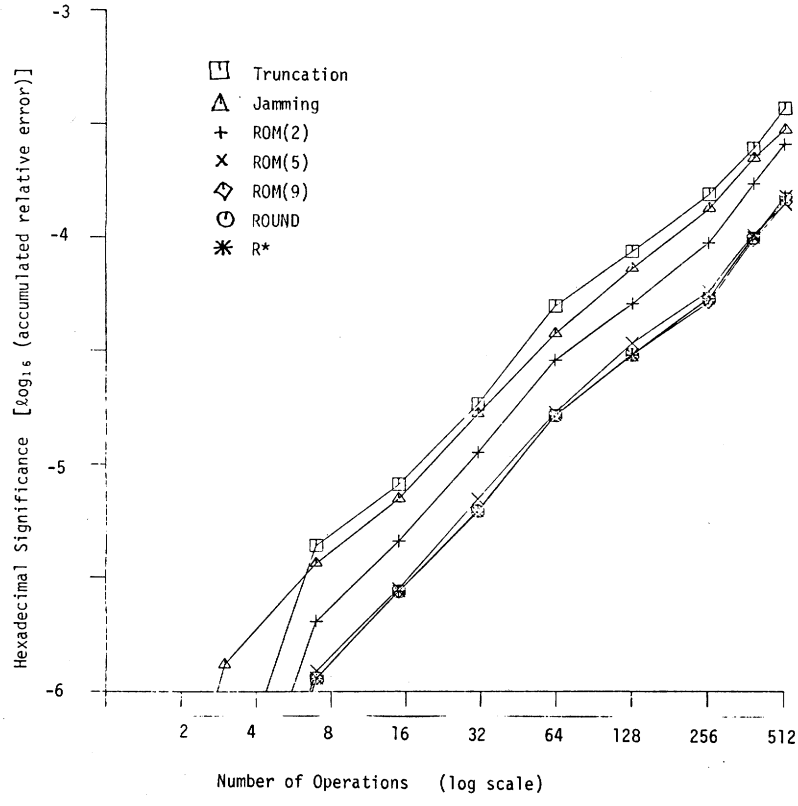


Fig. 2. Mixed-sign summation error growth.

V. Statistical Analysis of Rounding Methods

A number of papers appearing in 1973 ([1], [7], [9]) attempted to discriminate between floating-point systems, which used various radices and rounding schemes, by applying varied sets of data to a given problem, and statistically measuring the resultant roundoff errors. Except in [1], the problems typically consisted just of repeated arithmetic operations. For reasons cited in [6], we followed this line of testing and presented each of the rounding methods under consideration with several serial summation problems. The limited applicability of these tests to general computation was realized at the time; they were taken merely as indicators of the average tenacity of the rounding schemes against error, and reflectors of some of the facets of accumulated error not detectable with metrics of local roundoff error like ARRE and average bias.

What was not appreciated in [6] is that serial summation almost always produces exponent differences substantially larger than the averages obtained by Sweeney. The large exponent differences and alignment rounding tended to make the partial results satisfy the uniformity constraints of the (then logarithmic-law-oblivious) average bias definition well, so the final results gave results very close to the predicted ones.

Realizing the failings of [6], we designed a problem we felt was poorly conditioned enough to test roundoff performance, but at the same time representative of general floating-point computation: repeated serial summation of mixed-sign operands (50 percent of the summands negative). The summation was carried out over 400 sets of 512 random summands, whose mantissas were generated from a logarithmic distribution. Alignment shifts were *forced* to conform to the averages listed by Sweeney, and we felt the mixed-sign approach of the problem coincided well with Sweeney's statistics that 52.73 percent of his tabulated additions involved unlike signs, while 47.27 percent involved like signs. The actual summation was carried out in six-digit hexadecimal arithmetic because we were interested in higher radix performance, and for reasons of convenience. The resulting error growths for each rounding method tested, plotted as a function of the number of summands, are shown in Fig. 2. They give a spectrum of performance coinciding well with the results predicted by the static measures ARRE and average bias. Fig. 2 also seems to indicate that as the radix grows the optimal ROM length grows (here only l = 2, 5, and 9 were checked). However, this statement will require more verification before it can be accepted as true.

Note also the results seem to say that, on the average, rounding gives results only two bits better than truncation (i.e., 1/2 unit of hexadecimal significance better). This agrees well with [1]. It would be easy to pick a problem that discriminated against Truncation: for example, consider serial summation of positive numbers, which gives Truncation *six* bit worse results [6]. However, it is impossible to generalize from such specific problems what *average* roundoff performance will be.

This last statement points out the difficulty with constructing any single problem to test roundoff performance. The problem should be realistic but at the same time give results representative of all floating-point computation.

Sweeney's exponent difference distributions make the representativeness requirement exceedingly difficult to obtain. But the representativeness requirement is important: mixed-sign summations with Sweeney's alignment shifts as in Fig. 2 gave quite different results than mixed-sign summations without these shifts, as analyzed in [6, fig. 6]. Notice that the experiment here does not represent *any* realistic computation, in that there is probably some correlation between the average number of operations per datum and alignment shift distance in any real computation. Apparently, the only way to test roundoff performance empirically is to compare results of many standard programs run with the same data but employing different radices, word lengths, and rounding methods.

VI. Conclusions

We have presented a theoretical development of two static metrics of rounding scheme performance, ARRE and average bias, and have shown how they may be used to get dynamic estimates for average roundoff errors in any given floating-point system. The analysis here has displayed the difficulties introduced by both the logarithmic law of leading digits and the distribution of exponent differences in floating-point addition, and has shown why any further rounding method analysis must take these factors into consideration.

All of this analysis indicates that ROM(l)-rounding, with a suitably large ROM length l, is a viable rounding scheme in radix-2 and apparently (Fig. 2) for radices as big as 16. It also shows how any other new rounding scheme could be analyzed. However, the difficulties of testing this theory empirically are currently too great for us to do it properly: a study at least as extensive as those in [5] and [10] is warranted.

Late note: Work of related interest ([14]–[16]) by R. Goodman and A. Feldstein has appeared recently.

Acknowledgment

The authors wish to thank the referees for several incisive comments. They also wish to thank Mrs. Vivian Alsip for the superb job she did of typing the manuscript.

References

[1] R. P. Brent, "On the precision attainable with various floating-point number systems," *IEEE Trans. Comput.*, vol. C-22, pp. 601–607, June 1973.

[2] W. J. Cody, "Static and dynamic numerical characteristics of floating-point arithmetic," *IEEE Trans. Comput.*, vol. C-22, pp. 598–601, June 1973.

[3] J. A. Field, "Optimizing floating-point arithmetic via post addition shift probabilities," in *Proc. 1969 Spring Joint Computer Conf.*, 1969, pp. 597–603.

[4] T. Kaneko and B. Liu, "On the local roundoff errors in floating-point arithmetic," *J. Ass. Comput. Mach.*, vol. 20, pp. 391–398, July 1973.

[5] J. G. Kent, "Comparison sets: A useful partitioning of the space of floating-point operand pairs," in *Proc. 3rd IEEE Symp. Computer Arithmetic*, Nov. 1975, pp. 36–39.

[6] D. J. Kuck, D. S. Parker and A. H. Sameh, "ROM-rounding: A new rounding scheme," in *Proc. 3rd IEEE Symp. Computer Arithmetic*, Nov. 1975, pp. 67–72.

[7] H. Kuki and W. J. Cody, "A statistical study of the accuracy of floating-point number systems," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 223–230, Apr. 1973.

[8] W. M. McKeeman, "Representation error for real numbers in binary computer arithmetic," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 682–683, Oct. 1967.

[9] J. D. Marasa and D. W. Matula, "A simulative study of correlated error propagation in various finite-precision arithmetics," *IEEE Trans. Comput.*, vol. C-22, pp. 587–597, June 1973.

[10] T. Sweeney, "An analysis of floating-point addition," *IBM Syst. J.*, vol. 4, pp. 31–42, 1965.

[11] N. Tsao, "On the distributions of significant digits and roundoff errors," *Commun. Ass. Comput. Mach.*, vol. 17, pp. 269–271, May 1974.

[12] J. H. Wilkinson, *Rounding Errors in Algebraic Processes.* Englewood Cliffs, NJ: Prentice-Hall, 1963.

[13] D. E. Knuth, *The Art of Computer Programming*, vol. 2. Reading, Mass.: Addison-Wesley, 1969.

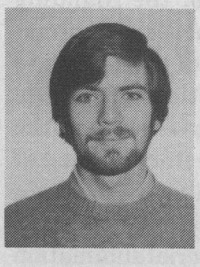
[14] R. Goodman and A. Feldstein, "Round-off error in products," *Computing*, vol. 15, pp. 263–273, 1975.

[15] A. Feldstein and R. Goodman, "Convergence estimates for the distribution of trailing digits," *J. Ass. Comput. Mach.*, vol. 23, pp. 287–297, Apr. 1976.

[16] R. Goodman, "On round-off error in fixed-point multiplication," *BIT*, vol. 16, pp. 41–51, 1976.

[17] W. M. Gentleman and S. B. Marovich, "More on algorithms that reveal properties of floating point arithmetic units," *Commun. Ass. Comput. Mach.*, vol. 17, pp. 276–277, May 1974.

David J. Kuck (S'59–M'69), for a photograph and biography please see p. 153 of the February 1977 issue of this TRANSACTIONS.



Douglass S. Parker, Jr., was born in New Haven, CT, on December 31, 1952. He received the A.B. degree in mathematics from Princeton University, Princeton, NJ, in 1974 and the M.S. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, in 1976.

Currently, he is working toward the Ph.D. degree as a Research Assistant in the Department of Computer Science at the University of Illinois. His interests include mathematical software, the design and analysis of algorithms, complexity, and systems analysis.

Mr. Parker is a member of Sigma Xi and the Association for Computing Machinery.

Ahmed H. Sameh, for a photograph and biography please see p. 152 of the February 1977 issue of this TRANSACTIONS.