On the Precision Attainable with Various Floating-Point Number Systems*

Richard P. Brent†

Abstract

For scientific computations on a digital computer the set of real numbers is usually approximated by a finite set F of "floating-point" numbers. We compare the numerical accuracy possible with different choices of F having approximately the same range and requiring the same word length. In particular, we compare different choices of base (or radix) in the usual floating-point systems. The emphasis is on the choice of F, not on the details of the number representation or the arithmetic, but both rounded and truncated arithmetic are considered. Theoretical results are given, and some simulations of typical floating point-computations (forming sums, solving systems of linear equations, finding eigenvalues) are described. If the leading fraction bit of a normalized base 2 number is not stored explicitly (saving a bit), and the criterion is to minimise the mean square roundoff error, then base 2 is best. If unnormalized numbers are allowed, so the first bit must be stored explicitly, then base 4 (or sometimes base 8) is the best of the usual systems.

Index Terms: Base, floating-point arithmetic, radix, representation error, rms error, rounding error, simulation.

1 Introduction

A real number x is usually approximated in a digital computer by an element $\mathrm{fl}(x)$ of a finite set F of "floating-point" numbers. We regard the elements of F as exactly representable real numbers, and take $\mathrm{fl}(x)$ as the floating-point number closest to x. The definition of "closest", rules for breaking ties, and the possibility of truncating instead of rounding are discussed later.

We restrict our attention to binary computers in which floating-point numbers are represented in a word (or multiple word) of fixed length w bits, using some convenient (possibly redundant) code. Usually F is a set of numbers of the form

$$s\sum_{i=1}^{t} d_i\beta^{e-i} \tag{1.1}$$

where $\beta = 2^k > 1$ is the base (or radix), $t > 0$ is the number of digits, $s = \pm 1$ is a sign, e is an exponent in some fixed range

$$m < e \leq M\,, \tag{1.2}$$

*Appeared as Report TR RC 3751, IBM Research (February 1972); and in *IEEE Transactions on Computers*, **C-22** (June 1973), 601–607 (manuscript received May 15, 1972; revised May 31, 1972). Retyped (with corrections) in LaTeX by Frances Page, July 2000.

†Former address: Mathematical Sciences Department, IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598. Current address: Oxford University Computing Laboratory, Oxford OX1 3QD, UK.

and each d_i is a β-ary digit $0, 1, \ldots, \beta - 1$. Other possible floating-point number systems (i.e, choices of F) are mentioned in Section 3.

Since the coding of the exponent e and the signed fraction $(s; d_1, \ldots, d_t)$ must fit into w bits, there is a tradeoff between precision and range. (A discussion of precision and range requirements for general scientific computing may be found in Cody [7, 8].) We do not consider this tradeoff; instead we suppose that the range and word length is prescribed, and we study the dependence of the precision on the base β.

With higher bases less bits are needed for the exponent, so more are available for the fraction (see Section 2 for details). However, more leading fraction bits may be zero, so the best choice of base is not immediately obvious. Our aim is to compare the attainable precision of systems with different bases. Theoretical results are given in Sections 4 and 5, and some simulations are described in Sections 6 and 7. The conclusions are summarised in Section 7.

Since we are interested in the precision attainable with different number systems, we assume that the arithmetic is the best possible. In other words, if $x, y \in F$, and $\dagger$ is an arithmetic operation, we asume that $x \dagger y$ is found to sufficient accuracy to give the correct (rounded) result $\mathrm{fl}(x \dagger y)$. Ensuring this may be too expensive in practice, but our conclusions should be valid provided several guard units are used when computing $\mathrm{fl}(x \dagger y)$. The reduction in precision caused by using only a small number of guard digits is discussed by Kuki and Cody [18].

2 The Usual Systems

A floating-point number of the form (1.1) may be written as

$$s \sum_{j=1}^{u} b_j 2^{ke-j} \tag{2.1}$$

where $b_{k(i-1)+1} \cdots b_{ki}$ is the binary form of the 2^k-ary digit d_i, and $u = kt$ is the number of bits required to code $d_i, \ldots, d_t$. We use (2.1) in preference to (1.1), and do not insist that t must be an integer. The details of the coding of the exponent e and the signed fraction $(s; b_1, \ldots, b_u)$ in a w-bit word do not concern us.

The representation (2.1) is said to be *normalised* if at least one of $b_1, \ldots, b_k$ is nonzero. From (2.1) and the bound (1.2) on e, the largest and smallest floating-point numbers having a normalised representation are

$$f_{\max} = 2^{kM}(1 - 2^{-u}) \tag{2.2}$$

and

$$f_{\min} = 2^{km}, \tag{2.3}$$

respectively. If the *range* R of the system is defined to be $\log_2(f_{\max}/f_{\min})$ then, negelecting the term 2^{-u} in (2.2)

$$k(M - m) = R. \tag{2.4}$$

Thus, for systems with the same range, $k(M - m)$ is invatiant.

Goldberg [10], McKeeman [21], and others have observed that with base 2 the leading fraction bit b_1 can be implicit, provided only normalized representations of nonzero numbers are allowed and a special exponent is reserved for zero. Define

$$p = \begin{cases} 2, & \text{if this "implicit-first-bit" idea is used} \\ 1, & \text{otherwise} \end{cases} \tag{2.5}$$

so $u - \log_2 p$ bits are required to code the fraction $(b_1, \ldots, b_u)$. One bit is required for the sign, and at least $\lceil \log_2(M - m) \rceil$ for the exponent. Thus

$$u - \log_2 p + 1 + \lceil \log_2(M - m) \rceil \le w. \tag{2.6}$$

For a sensible design, equality will hold in (2.6), and $M - m$ will be a power of two (or one less if the exponent is coded in one's complement or a special exponent is reserved for zero, but such minor differences are unimportant). Thus (2.4) gives

$$2^{-u}kp \;=\; 2^{1-w}R. \tag{2.7}$$

The right side depends only on the word length and the range, so (2.7) gives a useful relation between the fraction length u and the base $\beta \;=\; 2^k$.

Many different sustems of the class described here have actually been used. They include, with various word lengths, ranges and rounding (or truncating) rules:

$\beta = 2,\ p = 2$	(e.g., PDP 11-45);
$\beta = 2,\ p = 1$	(e.g., CDC 6400);
$\beta = 4$	(e.g., Illiac II);
$\beta = 8$	(e.g., Burroughs 5500); and
$\beta = 16$	(e.g., IBM 360).

In some machines, bases other than β are used in the arithmetic unit. For example, in the ILLIAC III (Atkins [2]) multiplication and division are performed with base 256, but numbers are stored with base 16.

3 Other Systems

Morris [22] suggests using "tapered" systems in which the division of bits between the exponent and the fraction depends on the exponent. The idea is to have a longer fraction for the (commonly occurring) numbers with exponents close to zero than for numbers with large exponents. We do not consider these interesting systems here.

Brown and Richman [5] assume that floating-point numbers are represented in a computer word with two sign bits and a fixed number of q-state devices, for some fixed $q \geq 2$, and they compare bases of the form q^k . Although the results of Sections 4 and 5 can be generalized easily to cover their assumptions, we restrict ourselves to $q = 2$, for this is the only case of practical importance.

Finally, we describe a "logarithmic" system that is interesting for theoretical reasons (see Section 4), although it is impractical (because of the difficulty of performing floating-point additions). Let a and b be positive integers which, together with the word length w , characterize the system. The floating-point numbers are zero and all nonzero real numbers x such that $a \cdot \log_2 |x| + b$ is one of the integers $1, 2, \ldots, 2^{w-1} - 1$. If

$$\lambda(x) \;=\; \begin{cases} 0, & \text{if } x = 0 \\ \operatorname{sign}(x)(a \cdot \log_2 |x| + b), & \text{if } x \neq 0 \end{cases} \tag{3.1}$$

then the floating-point number x may be represented in a computer word by a convenient code for the integer $\lambda(x)$. Since

$$x(\lambda) \;=\; \begin{cases} 0, & \text{if } \lambda = 0 \\ \operatorname{sign}(\lambda) \cdot 2^{(\lambda - b)/a}, & \text{if } \lambda \neq 0 \end{cases} \tag{3.2}$$

the largest and smallest positive floating-point numbers are

$$f_{\max} \;=\; 2^{(2^{w-1}-1-b)/a} \tag{3.3}$$

and

$$f_{\min} \;=\; 2^{(1-b)/a}, \tag{3.4}$$

respectively, and the range $\log_2(f_{\max}/f_{\min})$ is

$$R \;=\; \frac{2^{w-1}-2}{a}\,. \tag{3.5}$$

For example, taking $a = 2^{w-10}$ and $b = 2^{w-2}$ gives $f_{\max} \simeq 2^{256}$, $f_{\min} \simeq 2^{-256}$, and $r \simeq 512$.

If x and y are positive floating-point numbers with $f_{\min} \le xy \le f_{\max}$ then, from (3.1)

$$\lambda(xy) \;=\; \lambda(x) + \lambda(y) - b\,. \tag{3.6}$$

Thus, floating-point multiplication and division are easy to perform in a logarithmic system, and do not introduce any rounding errors. Unfortunately, there does not seem to be any easy way to perform floating-point addition.

4 The Worst Case Relative-Error Criterion

One measure of the precision of a floating-point number system is the worst relative error ϵ made in approximating a real number x (not too large or small) by $\mathrm{fl}(x)$, i.e.,

$$\epsilon \;=\; \sup_{f_{\min}\le|x|\le f_{\max}} \left|\frac{x-\mathrm{fl}(x)}{x}\right|. \tag{4.1}$$

The "worst case relative-error" criterion is simply to choose a number system (with the prescribed R and w) to minimise ϵ.

For the logarithmic systems described in Section 3, we see from (3.2) that

$$\epsilon \;=\; 2^{1/(2a)} - 1 \;=\; \frac{\log 2}{2a}\,. \tag{4.2}$$

(Here and later we neglect terms of order a^{-2} or 2^{-2u}, and logarithms are natural unless otherwise indicated.) If

$$\epsilon_0 \;=\; R2^{-w}\log 2 \tag{4.3}$$

then (3.5) and (4.2) give

$$\epsilon \;=\; \epsilon_0\,. \tag{4.4}$$

Now consider any floating-point number system with range R and word length w. If ϵ is defined by (4.1), then

$$\epsilon \;\ge\; \epsilon_0\,. \tag{4.5}$$

(In a logarithmic system, the logarithms of positive floating-point numbers are uniformly spaced and all bit patterns are used.) Thus we use logarithmic systems as a standard of comparison for other, more practical, systems.

Wilkinson [28] shows that

$$\epsilon \;=\; 2^{k-u-1} \tag{4.6}$$

for the number systems of Section 2. From (2.7), (4.3) and (4.6)

$$\frac{\epsilon}{\epsilon_0} \;=\; \frac{2^k}{kp\log 2} \;=\; f_1(k,p) \tag{4.7}$$

which shows how much ϵ exceeds the best possible value ϵ_0 for a number system with the same R and w. Table 1 gives $f_1(k,p)$ for $k = 1, 2, \ldots, 8$.

TABLE 1

THEORETICAL WORST CASE AND RMS ERRORS*

k	p	$\beta = 2^k$	$f_1(k,p)$	$f_2(k,p)$
1	2	2	1.44	1.06
1	1	2	2.89	2.12
2	1	4	2.89	1.68
3	1	8	3.85	1.87
4	1	16	5.77	2.45
5	1	32	9.23	3.51
6	1	64	15.4	5.34
7	1	128	26.4	8.47
8	1	256	46.2	13.9

* See (4.7) and (5.8) for definitions of f_1 and f_2.

The table shows that the implicit-first-bit base 2 systems are the best of those described in Section 2, and close to the best possible, on the worst case criterion. Of the explicit-first-bit systems, base 2 and base 4 are equally good. This may be explained as follows. Changing from base 2 to base 4 frees a bit from the exponent for the fraction. If the first 4-ary digit d_1 is 2 or 3, the first fraction bit b_1 is 1, and the extra fraction bit may increase the precision. However, if d_1 is 1 then b_1 is 0, and the bit gained is wasted. (According to Richman [24], Goldberg observed this independently.) If $\mathrm{fl}(x)$ is defined by truncation rather than rounding then ϵ is doubled, but the comparison between different bases is not changed.

5 The RMS Relative-Error Criterion

Consider forming the product of nonzero floating-point numbers $x_0, \ldots, x_n$ (in one of the usual systems) by n floating-point multiplications, i.e., define $p_0 = x_0$ and $p_i = \mathrm{fl}(p_{i-1}x_i)$ for $i = 1, \ldots, n$. If $\delta_i = (p_{i-1}x_i - p_i)/(p_{i-1}x_i)$ is the relative error made in forming the ith product, then the relative error in the final result is

$$\begin{aligned} \Delta &= \frac{x_0 \cdots x_n - p_n}{x_0 \cdots x_n} = 1 - \prod_{i=1}^{n} (1 - \delta_i) \\ &= \sum_{i=1}^{n} \delta_i + \text{higher order terms}. \end{aligned} \tag{5.1}$$

Thus

$$|\Delta| \le n\epsilon \tag{5.2}$$

where ϵ is defined by (4.1), and we have neglected a term of order $n^2\epsilon^2$. Many other bounds on the rounding errors in algebraic processes are also of the form $f(n)\epsilon$ (see Wilkinson [28], [29]), which is a good reason for choosing a floating-point number system according to the worst case criterion of Section 4. However, the bound (5.2) is rather pessimistic, for the individual rounding errors δ_i in (5.1) usually tend to cancel rather than to reinforce each other. (We are assuming an unbiased rounding rule as described in Section 6. With truncation or biased rounding the bound (5.2) may be realistic.)

If the δ_i were independent random variables, distributed with mean 0 and variance σ_i^2, then Δ would be distributed with mean 0 and variance $\Sigma_{i=1}^{n}\sigma_i^2$. Thus a reasonable probabilistic measure of the precision of a floating-point number system is the root-mean-square (rms) value δ_{rms} of $\delta = (x - \mathrm{fl}(x))/x$, where x is distributed like the nonzero results of arithmetic operations performed during a typical floating-point computation.

The simulations described in Sections 6 and 7 suggest that the rms rounding error in floating-point comuptations involving many arithmetic operations is often roughly proportional to $\delta_{\rm rms}$ (see also Weinstein [27]). Thus we prefer $\delta_{\rm rms}$ to other probabilistic measures of precision such as the expected value of $|\delta|$ (McKeeman [21]), the expected value of $\log_2|\delta|$ (Kuki and Codi [18]), and the expected error in "units in the last place" (Kahan [15]). We disregard errors in the conversion from internal floating-point results to decimal output, for the rms value of these errors depends on the number of decimal places rather than on the internal number system. (For the effect of repeated conversions back and forth, see Matula [20].)

What distribution should we assume for the nonzero real numbers x that are to be approximated by floating-point numbers? Hamming [11], Knuth [17], and others argue that we should assume that $\log|x|$ is uniformly distributed. There are two reasons why this assumption is only an approximation. Although $\log|x|$ may be approximately uniform locally, it is certainly not uniform on the entire interval $[\log f_{\rm min}, \log f_{\rm max}]$. Also, the fine structure of the distribution is not uniform, for the numbers arising from multiplications or (more importantly) additions of floating-point numbers are really discrete rather than continuous variables. Nevertheless, we shall make Hamming's assumption in this section. It is certainly a much better approximation than assuming that x is uniformly distributed on some interval.

For the logarithmic systems, δ is uniformly distributed on $[-\epsilon_0, \epsilon_0]$, where ϵ_0 is given by (4.3). Thus $\delta_{\rm rms} = \delta_0$, where

$$\delta_0 \;=\; \frac{\epsilon_0}{\sqrt{3}} \;=\; \frac{R\cdot\log 2}{2^w\sqrt{3}}\,. \tag{5.3}$$

Because the assumption that $\log|x|$ is uniform is only an approximation, there is no result corresponding to the inequality (4.5), but the logarithmic systems still provide a convenient standard of comparison for other, more practical, systems.

For the systems of Section 2, there is no loss of generality in assuming that x lies in $[1/\beta, 1)$ and (by our assumption) $\log_\beta x$ is uniformly distributed on $[-1, 0)$. Consider numbers y distributed uniformly on a small interval near x. The absolute error $y - {\rm fl}(y)$ is approximately uniform on $(-2^{-u-1}, 2^{-u-1})$. (It is certainly not logarithmically distributed, as is assumed to derive (18′) in Benschop and Ratz [3].) Hence $\alpha = (y - {\rm fl}(y))/y$ is uniform on $(-2^{-u-1}/x, 2^{-u-1}/x)$, and has probability density function (Feller [9])

$$g_x(\alpha) \;=\; \begin{cases} 2^u x, & \text{if } |\alpha| < 2^{-u-1}/x \\ 0, & \text{otherwise.} \end{cases} \tag{5.4}$$

Integrating over the interval $[1/\beta, 1)$, we see that δ is distributed with density

$$f(\delta) \;=\; \int_{x=1/\beta}^{1} g_x(\delta)\, d\log_\beta x \tag{5.5}$$

$$=\; \begin{cases} 2^u(1-2^{-k})/(k\cdot\log 2), & \text{if } |\delta| < 2^{-u-1} \\ \left(\dfrac{1}{2|\delta|} - 2^{u-k}\right)\Big/(k\cdot\log 2), & \text{if } 2^{-u-1} \le |\delta| < 2^{k-u-1} \\ 0, & \text{otherwise}\,. \end{cases} \tag{5.6}$$

It is easy to find the expected value of δ, δ^2, $|\delta|$, $\log_2|\delta|$, etc. from (5.6). In particular, we find that δ is distributed with standard deviation

$$\delta_{\rm rms} \;=\; 2^{-u}\sqrt{\frac{4^k-1}{24k\cdot\log 2}} \tag{5.7}$$

and mean 0. (The mean is actually of order 2^{-2u}, but terms of this order have been neglected.)

From (2.7), (5.3) and (5.6),

$$\frac{\delta_{\mathrm{rms}}}{\delta_0} = \sqrt{\frac{4^k - 1}{2p^2(k \cdot \log 2)^3}} = f_2(k,p)\,, \tag{5.8}$$

and the last column of Table 1 gives $f_2(k,p)$ for $k = 1, \ldots, 8$. The table shows that the implicit-first-bit base 2 systems are the best of the systems of Section 2 (and only 6 per cent worse than the logarithmic systems) on the rms relative-error criterion. Base 4 (closely followed by base 8) is best in the explicit-first-bit systems. The reason why base 4 is better than explicit base 2 is apparent from the discussion at the end of Section 4: $|\delta|$ is never greater for base 4 than for explicit base 2, and sometimes it is smaller. A similar argument shows that implicit base 2 is better than base 4.

Because of the different ranges possible with base 4 and base 8, there are some choices of minimal acceptable range for which base 8 is preferable to base 4, but bases higher than 8 are always inferior to base 4 on the rms relative-error criterion.

6 Simulation of Different Systems

Three classes of floating-point computations were run, using various number systems with $w = 32$ and $R \simeq 512$ (the same as for single-precision on the IBM 360 and many other computers). The systems were a logarithmic system S_0 with $a = 2^{22}$ and $b = 2^{30}$ (see Section 3), and the following examples of the systems described in Section 2.

S_1 :	$\beta = 2,\ u = 23,\ p = 2$	(base 2 with a 23-bit fraction, the first bit implicit).
S_2 :	$\beta = 4,\ u = 23$	(base 4 with 23 bits or $11\frac{1}{2}$ digits).
S_3 :	$\beta = 2,\ u = 22,\ p = 1$	(base 2 with 22 bits, all explicit).
S_4 :	$\beta = 16,\ u = 24$	(base 16 with 24 bits or 6 digits).
S_4' :	The same as S_4 with truncation (towards zero) rather than rounding.	
S_5 :	$\beta = 256,\ u = 25$	(base 256 with 25 bits or $3\frac{1}{8}$ digits).

The rounding rule for systems S_1 to S_5 is the "R*-mode" of Kuki and Cody [18]: $\mathrm{fl}(x)$ is defined to be the floating-point number closest to x, and ties are broken by choosing $\mathrm{fl}(x)$ so that its least significant fraction bit is one. Formally, if x is a nonzero real number with binary expansion

$$x = s\sum_{j=1}^{\infty} b_j 2^{ke-j} \tag{6.1}$$

(taking the terminating expansion if there is one, normalizing so that one of $b_1, \ldots, b_k$ is nonzero, and neglecting the possibility of underflow or overflow), then

$$\mathrm{fl}(x) = \begin{cases} s\sum_{j=1}^{u} b_j 2^{ke-j}, & \text{if } b_{u+1} = 0 \text{ or } \sum_{j=0}^{\infty} b_{u+j}2^{-j} = \frac{3}{2} \\ s\left(\sum_{j=1}^{u} b_j 2^{ke-j} + 2^{ke-u}\right), & \text{otherwise}. \end{cases} \tag{6.2}$$

The special case $b_u b_{u+1} \cdots = 11000\cdots$ is quite important, for it often occurs when x is the result of a floating-point addition, and neglecting it can lead to bias in the rounding.

All the floating point number systems were simulated on an IBM 360/91 computer, with arithmetic operations performed in double precision ($\beta = 16, u = 56$) before rounding or truncating approximately. Thus, the number of guard units used was effectively infinite. The data were pseudorandom double-precision numbers distributed as described in Section 7, and "exact" results were computed using double precision throughout.

Forming sums, solving systems of linear equations, and finding the eigenvalues of symmetric matrices were the chosen classes of floating-point computations. They appear to be fairly typical of computations in which the effect of rounding errors may be important. Details, and the results of the simulations, are given in Section 7. Other classes that have been considered include solving ordinary differential equations (Henrici [12], [13], Hull and Swenson [14]), fast Fourier transforms (Kaneko and Liu [16], Ramos [23], and Weinstein [27]), matrix iterative processes (Benschop and Ratz [3]), solving positive-definite linear systems (Tienari [25]), and forming products (Section 5).

7 Details and Results of the Simulations

Sums

Let m and n be positive integers. A number z was drawn from a uniform distribution on $[0,1]$, then numbers $x_1, \ldots, x_n$ were drawn independently from a uniform distribution on $[-Z, Z]$, where $Z = 256^z$ is a scale factor used to avoid a bias in favour of any of the number systems (see Kuki and Cody [18]). The approximate sums s_j of $\mathrm{fl}(x_1), \ldots, \mathrm{fl}(x_n)$ were accumulated, in the usual way, with each of the number systems S_j described in Section 6, and the errors

$$\alpha_j = \frac{\sum_{i=1}^{n} x_i - s_j}{\sum_{i=1}^{n} |x_i|} \tag{7.1}$$

were found. (The denominator is used in preference to $\sum_{i=1}^{n} x_i$ to ensure that α_j is small.) The procedure was repeated m times and the rms values β_j of the α_j were found. For purposes of comparison between the systems, it is convenient to consider the normalized rms errors $\gamma_j = \beta_j / \beta_0$. (Recall that β_0 is the rms error for the logarithmic system S_0.)

Table 2 gives γ_j for various choices of m and n. If the α_j are considered as random variables drawn from a distribution with mean square B_j^2, then β_j and γ_j may be regarded as estimates of B_j and B_j / B_0, respectively. m was chosen large enough to ensure that the standard error of the estimates γ_j given in Tables 2–4 is less than five units in the last decimal place.

For $n = 1$ we are merely estimating the rms relative error in approximating x_1 by $\mathrm{fl}(x_1)$, and the results agree with the predictions of Section 5 (see the last column of Table 1). Except for S_4', the effect of varying n is small, and does not affect the ranking of the systems.

It may be shown that

$$B_j = \begin{cases} O(n^{3/2}), & \text{for } S_4' \\ O(n), & \text{for the other systems} \end{cases} \tag{7.2}$$

so it is not surprising that γ_4' appears to grow like $n^{1/2}$. (The same applies if truncation is downwards instead of towards zero.) Results for sums of positive numbers are similar, although B_j is larger by a factor of order $n^{1/2}$ for all the systems.

TABLE 2

RESULTS FOR SUMS

n	$m/1000$	γ_1	γ_2	γ_3	γ_4	γ_4'	γ_5
1	1000	1.06	1.68	2.12	2.45	4.89	13.9
2	100	1.11	1.68	2.23	2.38	5.53	13.4
4	100	1.13	1.69	2.25	2.36	6.33	13.2
8	100	1.12	1.69	2.24	2.36	7.95	13.2
10	100	1.12	1.69	2.23	2.36	8.76	13.4
16	10	1.11	1.72	2.22	2.37	10.9	13.3
32	10	1.09	1.71	2.18	2.39	15.9	13.6
64	10	1.08	1.67	2.14	2.43	22.4	13.9
100	30	1.06	1.68	2.13	2.41	28.1	13.6

Solving Systems of Linear Equations

z_1 and z_2 were drawn independently from a uniform distribution on $[0,1]$, giving scale factors $Z_1 = 256^{z_1}$ and $Z_2 = 256^{z_2}$. Numbers $a_{p,q}$ $(p,q = 1,\ldots,n)$ were drawn independently from a uniform distribution on $[-Z_1, Z_1]$; and $x_1,\ldots,x_n$ were drawn similarly from $[-Z_2, Z_2]$. For each of the number systems S_j, let $A^{(j)} = (\mathrm{fl}(a_{p,q}))$, $A = (a_{p,q})$, $x = (x_p)$, $b = (b_p) = Ax$, and $b^{(j)} = (\mathrm{fl}(b_p))$. The system of equations

$$A^{(j)}y = b^{(j)} \tag{7.3}$$

was solved by Gaussian elimination with complete pivoting, giving the approximate solution $y^{(j)}$, and the error

$$\alpha_j = \frac{\|Ay^{(j)} - b\|_2}{\|A\|_E \; \|x\|_2} \tag{7.4}$$

was computed. (Here $\|A\|_E = \left(\sum_{p=1}^{n}\sum_{q=1}^{n} a_{p,q}^2\right)^{1/2}$. From results of Wilkinson [28], [29], α_j is small even if A is rather ill conditioned.) The procedure was repeated m times, the rms values β_j of the α_j were computed, and the ratios $\gamma_j = \beta_j/\beta_0$ were found. The results for various m and n are given in Table 3.

TABLE 3

RESULTS FOR SYSTEMS OF LINEAR EQUATIONS

n	$m/1000$	γ_1	γ_2	γ_3	γ_4	γ_4'	γ_5
1	100	1.30	2.06	2.61	2.99	4.92	17.0
2	100	1.30	2.01	2.59	2.90	5.33	16.3
4	10	1.27	1.97	2.56	2.80	5.63	15.7
8	4	1.23	1.89	2.45	2.65	6.1	14.9
16	1	1.18	1.82	2.35	2.60	7.1	14.4

Multiplication and division are performed exactly in a logarithmic system, so β_0 is less than would otherwise be expected, and $\gamma_1,\ldots,\gamma_5$ are higher than for sums, especially for small values of n. However, the ratios of $\gamma_1,\ldots,\gamma_5$ are much the same as for sums, and the ranking of the systems is preserved. Results for positive $a_{p,q}$ and/or x_p are similar.

It is interesting that $\gamma_4 < 2\gamma_4'$ for $n = 1$ and 2. When $n = 1$ and S_4' is used, the errors made in forming $\mathrm{fl}(a_{1,1})$ and $\mathrm{fl}(b_1)$ tend to cancel when $\mathrm{fl}(b_1)/\mathrm{fl}(a_{1,1})$ is computed. Presumably there is a similar, though less marked, effect for $n > 1$.

Finding Eigenvalues of Symmetric Matrices

Numbers $a_{p,q}$ $(1 \le p \le q \le n)$ were drawn independently from a uniform distribution on $[-Z, Z]$, where Z was a scale factor chosen as above. The other elements of $A = (a_{p,q})$ were defined by symmetry. For each number system S_j, the approximate eigenvalues $\lambda_1^{(j)} \le \cdots \le \lambda_n^{(j)}$ of $A^{(j)} = (\mathrm{fl}(a_{p,q}))$ were computed by reducing $A^{(j)}$ to tridiagonal form and then using the QR algorithm (Wilkinson [29]). We used translations of the Algol 60 procedures TRED1 (Martin *et al* [19]) and TQL1 (Bowler *et al* [4]), except for some trivial modifications to avoid unnecessary rounding errors when $n = 2$. The stopping criterion for the QR algorithm was the same for all number systems. (The parameters *macheps* and *tol* of the procedures were set to 10^{-8} and 10^{-60}, respectively.) The errors

$$\alpha_j = \left(\sum_{i=1}^{n} (\lambda_i - \lambda^{(j)})^2 \right)^{\frac{1}{2}} \Big/ \; \|A\|_E \tag{7.5}$$

were computed. (Here $\lambda_1 \le \cdots \le \lambda_n$ are the exact eigenvalues of A.) The procedure was repeated m times, the rms values β_j of the α_j computed, and the ratios $\gamma_j = \beta_j/\beta_0$ found, as above. The results are given in Table 4.

TABLE 4

RESULTS FOR EIGENVALUES OF SYMMETRIC MATRICES

n	$m/1000$	γ_1	γ_2	γ_3	γ_4	γ_4'	γ_5
2	100	1.07	1.61	2.14	2.38	6.06	15.2
4	10	1.33	2.24	2.65	3.60	10.5	25.8
8	3	1.14	2.01	2.34	3.73	10.8	29.6
16	1	1.00	1.82	1.99	3.49	10.7	28.8

The method used for finding eigenvalues depends heavily on multiplications by matrices of the form $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$, where $c^2 + s^2 = 1$. The numbers c and s are certainly not distributed as assumed in Section 5. This, along with other observations made above, may explain the interesting variations in the γ_j. Despite these variations, the ranking of the different systems is as predicted in Section 5.

8 Conclusions

Comparing γ_4' with γ_4 in Tables 2–4 shows that the rms error for truncation is usually considerably more than twice as much as for rounding. However, truncation is often preferred because the usual implementation of rounding requires an extra carry propagation. An interesting compromise is the "von Neumann round" (Burks *et al* [6], Urabe [26]), for which the result of an arithmetic operation is truncated, and then the least significant bit is set to one. (An exception could be made if the result is exactly representable; this would involve checking if the truncated bits were all zero.) No extra carry propagation is required, and the rms error is twice that for normal rounding, so considerably better than for truncation.

The most accurate practical systems are base 2 with the first fraction bit implicit. If the accuracy gained by having the first bit implicit is not considered sufficient compensation for the disadvantages entailed, then base 4 (or perhaps base 8) is the best choice.

The accuracy lost by using base 16 or higher is roughly as predicted in Section 5. High bases may have some implementation advantages (Anderson *et al* [1], Atkins [2]). In practice both factors should be considered. The number of guard digits used is also important. The use of high bases, only one guard digit, and truncation instead of rounding is probably acceptable on machines with a long floating-point word. However, to minimize the need for double-precision computations, it seems wise to try to squeeze out the last drop of accuracy on a computer with a short floating-point word (say 32–40 bits). The amount that can be squeezed out is often significant. For example, our simulations show that using system S_1 instead of S'_4 is roughly equivalent to carrying one more *decimal* place.

Acknowledgement

The author wishes to thank W J Cody, the late Prof. G E Forsythe, and Prof. W Kahan for their comments on an earlier version of this paper.

References

[1] S F Anderson, J G Earle and R E Goldschmidt, "The IBM system/360 model 91: Floating-point execution unit", *IBM J Res Develop*, vol 11, pp 34–53, January 1967.

[2] D E Atkins, "Design of arithmetic units of ILLIAC III: The use of redundancy and higher radix methods", *IEEE Trans Comput*, vol C-19, pp 720–733, August 1970.

[3] N F Benschop and H C Ratz, "A mean square estimate of the generated roundoff error in constant matrix interative processes", *J Ass Comput Mach*, vol 18, pp 48–62, January 1971.

[4] H Bowdler, R S Martin, C Reinsch and J H Wilkinson, "The QR and QL algorithms for symmetric matrices", *Numer Math*, vol 11, pp 293–306, 1968.

[5] W S Brown and P L Richman, "The choice of base", *Commun Ass Comput Mach*, vol 12, pp 560–561, October 1969.

[6] A W Burks, H H Goldstine and J von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument", in *Collected Works of John von Neumann*, vol 5. New York: Macmillan, 1963, pp 57–58. (Report prepared for the US Army, 1946.)

[7] W J Cody, "Desirable hardware characteristics for scientific computation", Preliminary Report to the SIGNUM Board of Directors, 1970.

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

[9] W Feller, *An Introduction to Probability Theory and its Applications*, New York: Wiley, 1950.

[10] I B Goldberg, "27 bits are not enough for 8-digit accuracy", *Commun Ass Comput Mach*, vol 10, pp 105–106, February 1967.

[11] R W Hamming, "On the distribution of numbers", *Bell Syst Tech J*, vol 49, pp 1609–1625, October 1970.

[12] P Henrici, *Discrete Variable Methods in Ordinary Differential Equations*. New York: Wiley, 1962, pp 50–54.

[13] P Henrici, "Test of probabilistic models for the propagation of roundoff errors", *Commun Ass Comput Mach*, (Letter to the Editor), vol 9, pp 409–410, June 1966.

[14] T E Hull and J R Swenson, "Tests of probabilistic models for propagation of roundoff errors", *Commun Ass Comput Mach*, vol 9, pp 108–113, February 1966.

[15] W Kahan, "What is the best base for floating-point arithmetic? Is binary best?", Dep Comput Sci, Univ California, Berkeley, Lecture Notes, December 1970.

[16] T Kaneko and B Liu, "Accumulation of roundoff error in fast Fourier transforms", *J Ass Comput Mach*, vol 17, pp 637–654, October 1970.

[17] D E Knuth, *The Art of Computer Programming*, vol 2, Reading, Mass: Addison-Wesley, 1969, pp 218–228.

[18] H Kuki and W J Cody, "A statistical study of the accuracy of floating-point number systems", *Commun Ass Comput Mach*, to be published. [Appeared in vol 16, pp 223–230, April 1973.]

[19] R S Martin, C Reinsch and J H Wilkinson, "Householder's tridiagonalization of a symmetric matrix", *Numer Math*, vol 11, pp 181–195, 1968.

[20] D W Matula, " A formalization of floating-point numeric base conversion", *IEEE Trans Comput*, vol C-19, pp 681–692, August 1970.

[21] W McKeeman, "Representation error for real numbers in binary computer arithmetic", *IEEE Trans Electron Comput*. (Short Notes), vol EC-16, pp 682–683, October 1967.

[22] R Morris, "Tapered floating-point: A new floating-point representation", *IEEE Trans Comput*. (Short Notes), vol C-20, pp 1578–1579, December 1971.

[23] G U Ramos, "Roundoff error analysis of the fast Fourier transform", *Math Comput*, vol 25, pp 757–768, October 1971.

[24] P L Richman, "Floating-point number representations: Base choice versus exponent range", Dep Comput Sci, Stanford Univ, Stanford, Calif, Tech Rep CS 64, 1967.

[25] M Tienari, "A statistical model of roundoff error for varying length floating-point arithmetic", *BIT*, vol 10, pp 355–365, 1970.

[26] M Urabe, "Roundoff error distribution in fixed-point multiplication and a remark about the rounding rule", *SIAM J Numer Anal*, vol 5, pp 202–210, 1968.

[27] C J Weinstein, "Roundoff noise in floating-point fast Fourier transform computation", *IEEE Trans Audio Electroacoust*, vol AU-17, pp 209–215, September 1969.

[28] J H Wilkinson, *Rounding Errors in Algbraic Processes*. London: HMSO, 1963.

[29] J H Wilkinson, *The Algebraic Eigenvalue Problem*. Oxford: Oxford, 1965.

Richard P. Brent (M'72) was born in Melbourne, Australia, on April 20, 1946. He received the BSc (Hons) degree in mathematics from Monash University, Clayton, Victoria, Australia, in 1967, and the MS and PhD degrees in computer science from Stanford University, Stanford, Calif, in 1970 and 1971, respectively.

He is currently[1] a Research Fellow in the Computer Centre, Australian National University, Canberra, Australia, after spending a year at the IBM T. J. Watson Research Center, Yorktown Heights, N.Y. His current research interests include computer arithmetic, numerical analysis, and computational complexity. He is the author of *Algorithms for Minimization without Derivatives.*

[1]This was written in 1972. Since 1998 he has been Professor of Computing Science at Oxford University, Oxford, England.