SIAM J. SCI. COMPUT.
Vol. 14, No. 4, pp. 783–799, July 1993

003

THE ACCURACY OF FLOATING POINT SUMMATION*

NICHOLAS J. HIGHAM†

Abstract. The usual recursive summation technique is just one of several ways of computing the sum of n floating point numbers. Five summation methods and their variations are analyzed here. The accuracy of the methods is compared using rounding error analysis and numerical experiments. Four of the methods are shown to be special cases of a general class of methods, and an error analysis is given for this class. No one method is uniformly more accurate than the others, but some guidelines are given on the choice of method in particular cases.

Key words. floating point summation, rounding error analysis, orderings

AMS subject classifications. primary 65G05, secondary 65B10

1. Introduction.

Sums of floating point numbers are ubiquitous in scientific computing. They occur when evaluating inner products, means, variances, norms, and all kinds of nonlinear functions. Although, at first sight, summation might appear to offer little scope for algorithmic ingenuity, the usual "recursive summation" (with various orderings) is just one of several possible techniques. Most of the other techniques have been derived with the aim of achieving greater accuracy of the computed sum, but pairwise summation has the advantage of being particularly well suited to parallel computation.

In this paper we examine a variety of methods for floating point summation, with the aim of answering the question, "which methods achieve the best accuracy?" Several authors have used error analysis to compare summation methods; see, for example, [1], [2], [32], and [38]. Here we give a more comprehensive treatment that highlights the relationships between different methods; in particular, we give an error analysis for a general class of methods that includes most of the specific summation methods as special cases.

This work was motivated by two applications in which the choice of summation method has been found to have an important influence on the performance of a numerical method.

(1) In [24], Lasdon et al. derive an algorithm for solving an optimization problem that arises in the design of sonar arrays. The authors state [24, p. 145] that "the objective gradient ∇f in (4.1) is a sum of M terms. In problems with $M = 284$ and $n = 42$, the GRG2 optimizer encountered difficulties which stem from inaccuracies in $\nabla f \ldots$. We hypothesized that this was due to roundoff error resulting from cancellation of terms in ∇f of approximately equal magnitudes and opposite signs. These problems were eliminated by accumulating separately positive and negative terms (for each component of ∇f) in the sum (4.1), adding them together only after all M terms had been processed."

(2) Dixon and Mills [7] applied a quasi-Newton method to the extended Rosenbrock function

$$F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n/2} \left(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2\right). \tag{1.1}$$

This function and its derivatives possess certain symmetries; for example, $F(a, b, c, d) = F(c, d, a, b)$ when $n = 4$. It is observed in [7] that expected symmetries in the search

*Received by the editors August 12, 1991; accepted for publication (in revised form) July 10, 1992.

†Department of Mathematics, University of Manchester, Manchester, M13 9PL, United Kingdom (na.nhigham@na-net.ornl.gov).

direction and Hessian approximation are lost in practice, resulting in more iterations for convergence of the quasi-Newton method than are predicted theoretically. Dixon and Mills attribute the loss of symmetry to rounding errors in the evaluation of certain inner products, which can cause identities such as the one quoted above to fail in floating point arithmetic. They restore symmetry (and thus reduce the number of iterations) by using a special summation algorithm when evaluating inner products: their algorithm evaluates $\sum_{i=1}^n x_i$ by sorting the x_i, dividing them into a list of negative numbers and a list of nonnegative numbers, and then repeatedly forming the sum of the largest nonnegative and most negative elements and placing the sum into the appropriate list, in order.

We return to these two applications in §7.

The five main summation methods that we consider are defined and analyzed in §§2 and 3. For the error analysis we will make use of the standard model of floating point arithmetic, in which u is the unit roundoff:

$$fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \qquad |\delta| \le u, \quad \text{op } = +, -, *, /. \tag{1.2}$$

This model is violated by machines that lack a guard digit, so we explain in §5 how our analysis has to be modified to accommodate such machines. We will assume that no floating point underflows occur; how to modify error analyses to allow for underflow is described by Demmel in [6]. An excellent tutorial on many aspects of floating point arithmetic is given by Goldberg [9].

In §4 we summarize some existing results on statistical estimates of accuracy of summation methods. Numerical experiments are presented in §6 and conclusions are given in §7.

2. Orderings of recursive summation. Our task is to evaluate $S_n = \sum_{i=1}^n x_i$, where $x_1, \dots, x_n$ are real numbers. In this section we consider the standard recursive summation technique in which S_n is evaluated according to

```
s = 0
for i = 1:n
    s = s + x_i
end
```

In general, each different ordering of the x_i will yield a different computed sum $\widehat{S}_n$ in floating point arithmetic, and it is of interest to determine how the ordering affects the error

$$E_n = \widehat{S}_n - S_n.$$

To begin, we make no assumption on the ordering and obtain a standard bound for E_n. By a direct application of the model (1.2) we have, with $S_k = \sum_{i=1}^k x_i$,

$$\widehat{S}_k = fl(\widehat{S}_{k-1} + x_k) = (\widehat{S}_{k-1} + x_k)(1+\delta_k), \qquad |\delta_k| \le u, \quad k = 2\colon n. \tag{2.1}$$

By repeated use of this relation it follows that

$$\widehat{S}_n = (x_1 + x_2)\prod_{k=2}^n (1+\delta_k) + \sum_{i=3}^n x_i \prod_{k=i}^n (1+\delta_k). \tag{2.2}$$

To simplify the product terms we use the result that if $|\delta_i| \le u$ for $i = 1\colon n$ then

$$\prod_{i=1}^n (1+\delta_i) = 1 + \theta_n, \quad \text{where} \quad |\theta_n| \le \frac{nu}{1-nu} \equiv \gamma_n.$$

Thus we rewrite (2.2) as

$$\widehat{S}_n = (x_1 + x_2)(1 + \theta_{n-1}) + \sum_{i=3}^{n} x_i(1 + \theta_{n-i+1}), \tag{2.3}$$

which yields

$$|E_n| = |(x_1 + x_2)\theta_{n-1} + \sum_{i=3}^{n} x_i\theta_{n-i+1}| \tag{2.4}$$

$$\leq (|x_1| + |x_2|)\gamma_{n-1} + \sum_{i=3}^{n} |x_i|\gamma_{n-i+1}. \tag{2.5}$$

Note that x_1 and x_2 have the same factor γ_{n-1}, because they play identical roles in the summation. The bound (2.5) is essentially the same as the one derived by Wilkinson in [42, p. 323] and [43, p. 17]. As Wilkinson notes in [43, p. 17], the upper bound in (2.5) depends on the order of summation, and the bound is minimized if the x_i are arranged in order of increasing absolute value. We emphasize that this ordering minimizes an error *bound* and not necessarily the actual error (this is illustrated later in this section, and by numerical examples in §6). We can weaken (2.5) to obtain the bound

$$|E_n| \leq \gamma_{n-1} \sum_{i=1}^{n} |x_i| = (n-1)u \sum_{i=1}^{n} |x_i| + O(u^2), \tag{2.6}$$

which is independent of the ordering. It is natural to regard satisfaction of (2.6) as a minimal requirement of any summation method; in fact, all the methods we will examine do satisfy this bound.

We can rewrite (2.6) as the relative error bound

$$\frac{|\widehat{S}_n - S_n|}{|S_n|} \leq \gamma_{n-1} \frac{\sum_{i=1}^{n} |x_i|}{|S_n|} \equiv \gamma_{n-1} R_n.$$

In the special case where $x_i \geq 0$ for all u, $R_n \equiv 1$, and the relative error has a bound of order nu, but if $\sum_{i=1}^{n} |x_i| \gg |\sum_{i=1}^{n} x_i|$ we cannot guarantee that the relative error is small. The quantity R_n is easily seen to be the condition number of summation when perturbations $x_i \rightarrow x_i + \Delta x_i$ are measured by $\max_i |\Delta x_i|/|x_i|$.

Recursive summation by order of increasing absolute value can be improved upon in two possible ways. First, a method may satisfy a bound of the form (2.6) but with a constant smaller than γ_{n-1}. Second, a method may satisfy a bound that in the worst case is no better than (2.6), but the method might be expected to yield a more accurate computed sum for particular classes of $\{x_i\}$. In the rest of this section we consider two alternative orderings, which fall into the second category.

First, we derive a sharper error bound. From (2.1) we see that the error introduced on the kth step of the summation is $(\widehat{S}_{k-1} + x_k)\delta_k = \widehat{S}_k\delta_k/(1 + \delta_k)$. Summing these individual errors we find

$$E_n = \sum_{k=2}^{n} \widehat{S}_k \frac{\delta_k}{1 + \delta_k}, \tag{2.7}$$

which shows that, to first order, the overall error is the sum of the $n-1$ relative rounding errors weighted by the partial sums. We obtain the bound

$$|E_n| = |\widehat{S}_n - S_n| \leq \frac{u}{1-u} \sum_{k=2}^{n} |\widehat{S}_k|. \tag{2.8}$$

This bound involves the computed partial sums (excluding $\widehat{S}_1 = x_1$) but not the individual terms x_i. If we weaken (2.8) by bounding $|\widehat{S}_k|$ in terms of $|x_1|, |x_2|, \ldots, |x_k|$, then we recover (2.5), to within $O(u^2)$.

The bound (2.8) suggests the strategy of ordering the x_i so as to minimize $\sum_{k=2}^{n} |\widehat{S}_k|$. This is a combinatorial optimization problem that is too expensive to solve in the context of summation. A reasonable compromise is to determine the ordering sequentially by minimizing, in turn, $|x_1|, |\widehat{S}_2|, \ldots, |\widehat{S}_{n-1}|$. This ordering strategy, which we denote by Psum, can be implemented with $O(n \log n)$ comparisons. The principal difference between the Psum and increasing orderings is that the Psum ordering is influenced by the signs of the x_i, while the increasing ordering is independent of the signs. If all the x_i have the same sign then the two orderings are identical.

It is easy to show by example that the bounds (2.8), (2.5) and (2.6) are nearly attainable. Following Wilkinson [43, p. 19] we assume $u = 2^{-t}$, set $n = 2^r$ ($r \ll t$), and define

$$\begin{aligned} x(1) &= 1, \\ x(2) &= 1 - 2^{-t}, \\ x(3{:}4) &= 1 - 2^{1-t}, \\ x(5{:}8) &= 1 - 2^{2-t}, \\ &\vdots \\ x(2^{r-1}+1{:}2^r) &= 1 - 2^{r-1-t}. \end{aligned}$$

Then in the $(i-1)$st floating point addition the "2^{k-t}" portion of x_i does not propagate into the sum;[1] thus there is an error of 2^{k-t} and $\widehat{S}_i = i$. The total error is

$$2^{-t}(1 + 2^2 + 2^4 + \cdots + 2^{2(r-1)}) = 2^{-t}\frac{4^r - 1}{3},$$

while the upper bound of (2.6) is

$$\frac{(n-1)u}{1-(n-1)u}\sum_{i=1}^{n}|x_i| \le \frac{2^r 2^{-t}}{1 - 2^r 2^{-t}} 2^r \approx 2^{-t}4^r,$$

which agrees with the actual error to within a factor 3; thus the smaller upper bounds of (2.5) and (2.8) are also correct to within this factor. The example just quoted is, of course, a very special one, and as Wilkinson [43, p. 20] explains, "in order to approach the upper bound as closely as this, not only must each error take its maximum value, but all the terms must be almost equal."

Next, we consider ordering the x_i by decreasing absolute value. For the summation of positive numbers this ordering has little to recommend it. The bound (2.8) is no smaller, and potentially much larger, than for the increasing ordering (the same is true for the weaker bound (2.5)). Furthermore, in a sum of positive terms that vary widely in magnitude, the decreasing ordering may not allow the smaller terms to contribute to the sum (which is why $\sum_{i=1}^{n} 1/i$ "converges" in floating point arithmetic as $n \to \infty$). However, consider the example with $n = 4$ and

$$x = [1, \quad M, \quad 2M, \quad -3M], \tag{2.9}$$

[1] We assume in this example that the floating point arithmetic uses round to nearest with ties broken by rounding to an even last bit or rounding away from zero.

where M is a power of the machine base and is so large that $fl(1 + M) = M$ (thus $M \geq u^{-1}$). The three orderings considered so far produce the following results:

$$\begin{aligned} \text{Increasing:} \quad & \widehat{S}_n = fl(1 + M + 2M - 3M) = 0, \\ \text{Psum:} \quad & \widehat{S}_n = fl(1 + M - 3M + 2M) = 0, \\ \text{Decreasing:} \quad & \widehat{S}_n = fl(-3M + 2M + M + 1) = 1. \end{aligned}$$

Thus the decreasing ordering sustains no rounding errors and produces the exact answer, while both the increasing and Psum orderings yield computed sums with relative error 1. The reason why the decreasing ordering performs so well in this example is that it adds the "1" after the inevitable heavy cancellation has taken place, rather than before, and so retains the important information in this term.

If we evaluate the term $\mu = \sum_{k=2}^{n} |\widehat{S}_k|$ in the error bound (2.8) for the example (2.9) we find

$$\text{Increasing: } \mu = 4M, \qquad \text{Psum: } \mu = 3M, \qquad \text{Decreasing: } \mu = M + 1,$$

so (2.8) "predicts" that the decreasing ordering will produce the most accurate answer, but the bound it provides is extremely pessimistic since there are no rounding errors in this instance. This example illustrates the main weakness of bounds from a rounding error analysis: they represent the worst case and so do not account for the possibility that rounding errors may cancel or be smaller than expected.

Extrapolating from this example, we conclude that the decreasing ordering is likely to yield greater accuracy than the increasing or Psum orderings whenever there is heavy cancellation in the sum, that is, whenever $|\sum_{i=1}^{n} x_i| \ll \sum_{i=1}^{n} |x_i|$. A numerical example that illustrates this assertion is given in §6 (see Table 6.1).

3. Other methods. In this section we consider in detail four more summation methods. The first three of these methods, together with recursive summation, have the following general form: with $T_k \equiv x_k$, $k = 1\colon n$, they perform $n - 1$ additions

$$T_k = T_{k_1} + T_{k_2}, \quad k_1 < k_2 < k, \quad k = n + 1\colon 2n - 1, \tag{3.1}$$

yielding $S_n = T_{2n-1}$. In recursive summation, $k_1 \leq n$ in each instance of (3.1), but for the other methods at least one addition involves two previously computed sums. A useful expression for the error in this general class of summation methods can be derived as follows. The computed quantities $\widehat{T}_k$ satisfy

$$\widehat{T}_k = (\widehat{T}_{k_1} + \widehat{T}_{k_2})(1 + \delta_k), \quad |\delta_k| \leq u, \quad k = n + 1\colon 2n - 1. \tag{3.2}$$

The local error introduced in forming $\widehat{T}_k$ is $(\widehat{T}_{k_1} + \widehat{T}_{k_2})\delta_k = \widehat{T}_k \delta_k/(1 + \delta_k)$, so overall we have

$$\widehat{S}_n - S_n = \sum_{k=n+1}^{2n-1} \widehat{T}_k \frac{\delta_k}{1 + \delta_k}. \tag{3.3}$$

The smallest possible error bound is therefore

$$|E_n| \leq \frac{u}{1 - u} \sum_{k=n+1}^{2n-1} |\widehat{T}_k|. \tag{3.4}$$

It is easy to see that $|\widehat{T}_k| \le \sum_{i=1}^n |x_i| + O(u)$ for each k, and so we also have the weaker bound

$$|E_n| \le (n-1)u \sum_{i=1}^n |x_i| + O(u^2). \tag{3.5}$$

Note that in the case of recursive summation (3.3)–(3.5) are the same as (2.6)–(2.8). Finally, we note in passing that from (3.2) there follows a backward error result which shows that $\widehat{S}_n$ is the exact sum of terms $x_i(1+\theta_i)$, where $|\theta_i| = O(u)$.

The first method we consider is pairwise summation (also known as cascade summation), which was first discussed by McCracken and Dorn [29, pp. 61–63], Babuška [1], and Linz [27]. In this method the x_i are summed in pairs,

$$y_i = x_{2i-1} + x_{2i}, \quad i = 1 : \left[\tfrac{n}{2}\right] \quad (y_{[(n+1)/2]} = x_n \text{ if } n \text{ is odd}),$$

and this pairwise summation process is repeated recursively on the y_i, $i = 1 : [(n+1)/2]$. The sum is obtained in $\lceil \log_2 n \rceil$ stages. For $n = 6$, for example, pairwise summation forms

$$S_6 = \big((x_1 + x_2) + (x_3 + x_4)\big) + (x_5 + x_6).$$

Pairwise summation is attractive in parallel settings, because each of the $\lceil \log_2 n \rceil$ stages can be done in parallel [13, §5.2.2]. Caprani [4] shows how to implement the method on a serial machine using temporary storage of size $\lceil \log_2 n \rceil$ (without overwriting the x_i).

The error expression (3.3) holds for pairwise summation, but it is easy to derive a useful error bound independently. Assume for simplicity that $n = 2^r$. Unlike in recursive summation each addend takes part in the same number of additions, $\log_2 n$. Therefore, analogously to (2.2), we have a relation of the form

$$\widehat{S}_n = \sum_{i=1}^n x_i \prod_{k=1}^{\log_2 n} (1 + \delta_k^{(i)}), \qquad |\delta_k^{(i)}| \le u,$$

which leads to the bound

$$|E_n| \le \gamma_{\log_2 n} \sum_{i=1}^n |x_i|. \tag{3.6}$$

This is a smaller bound than (2.6) for recursive summation, since it is proportional to $\log_2 n$ rather than n. However, in special cases the bound (2.5) for recursive summation can be smaller than (3.6). For example, if $x_i = 1/i^3$, the bound (3.6) is

$$|E_n| \le 1.20 \log_2 n\, u + O(u^2) \tag{3.7}$$

(using $\sum_{i=1}^n 1/i^3 \approx \sum_{i=1}^\infty 1/i^3 \approx 1.20$), while for the increasing ordering (2.5) becomes

$$|E_n| \le u \sum_{i=1}^n \frac{1}{(n-i+1)^3}(n-i+1) + O(u^2) = u \sum_{i=1}^n \frac{1}{i^2} + O(u^2) \approx 1.64u + O(u^2)$$

(using $\sum_{i=1}^n 1/i^2 \approx \sum_{i=1}^\infty 1/i^2 = \pi^2/6 \approx 1.64$), and so pairwise summation has the larger error bound, by a factor $\approx \log_2 n$. (Expression (3.3) does not enable us to improve on the factor $\log_2 n$ in (3.7).)

In [35] an "insertion adder" is proposed for the summation of positive numbers. This method can be applied equally well to arbitrary sums. First, the x_i are sorted by order of increasing magnitude. Then $x_1 + x_2$ is formed, and the sum is inserted into the list $x_2, \ldots, x_n$, maintaining the increasing order. The process is repeated recursively until the final sum is obtained. The motivation given in [35] for this strategy is that it tends to encourage the additions to be between numbers of similar magnitude. It can be argued that such additions are to be preferred, because they retain more of the information in the addends (by comparison, "large" plus "small" may lose many significant digits from "small"). A more convincing explanation of the insertion strategy is that it attempts to minimize, one at a time, the absolute values of the terms $\widehat{T}_{n+1}, \ldots, \widehat{T}_{2n-1}$ in the error expression (3.3). Indeed, if the x_i are all positive the insertion method minimizes the bound (3.4) over all methods of the form (3.1).

In particular cases the insertion method reduces to earlier methods. For example, if $x_i = 2^i$, the insertion method is equivalent to recursive summation, since the insertion is always to the bottom of the list:

$$1\,2\,4\,8 \quad \rightarrow \quad \underline{3}\,4\,8 \quad \rightarrow \quad \underline{7}\,8 \quad \rightarrow \quad 15.$$

On the other hand, if $1 \leq x_1 < x_2 < \cdots < x_n \leq 2$, every insertion is to the end of the list, and the method is equivalent to pairwise summation if n is a power of 2; for example, if $0 < \epsilon < \frac{1}{2}$,

$$1,\ 1+\epsilon,\ 1+2\epsilon,\ 1+3\epsilon \quad \rightarrow \quad 1+2\epsilon,\ 1+3\epsilon,\ \underline{2+\epsilon} \quad \rightarrow \quad 2+\epsilon,\ \underline{2+5\epsilon} \quad \rightarrow \quad 4+6\epsilon.$$

The next method we consider is the one used in [24], as quoted in the Introduction. This method can be derived by the following specious reasoning: "A major source of inaccuracy in floating point summation is cancellation when numbers of nearly equal magnitude and opposite sign are added. To minimize the amount of cancellation we can accumulate the sum of the positive numbers, S_+, and the sum of the negative numbers, S_-, separately, and then form $S_n = S_+ + S_-$." There are two flaws in this argument. First, this "$+/-$" method does not reduce the amount of cancellation—it simply concentrates all the cancellation into one step. Second, cancellation is not a bad thing per se; the problem with cancellation is that it brings into prominence any loss of significant digits suffered earlier in the calculation (and it also brings into prominence any uncertainty in the data). Indeed, nearly equal floating point numbers are always subtracted *exactly* (assuming the presence of a guard digit)—it is any (relative) uncertainty in those numbers that is magnified. For an excellent and more detailed discussion of cancellation, we refer the reader to [34, pp. 25–29].

The $+/-$ method is of the form (3.1) (assuming that S_+ and S_- are computed using one of the methods discussed so far) and it is easy to see that it maximizes $\max_k |T_k|$ over all methods of this form. Moreover, when $\sum_{i=1}^n |x_i| \gg |\sum_{i=1}^n x_i|$ the value of $\max_k |T_k|$ tends to be much larger for the $+/-$ method than for the other methods we have considered. For example, if $n = 2m$ and the x_i are the values $\{-1, 1, -2, 2, \ldots, -m, m\}$ then

$$|S_+| = |S_-| = |T_{2n-2}| = \sum_{i=1}^{m} k = m(m+1)/2,$$

whereas for recursive summation with the increasing ordering, $|T_k| \leq m$ for all k. Despite this weakness, if S_+ and S_- are computed by recursive summation with the increasing ordering then the $+/-$ method satisfies a bound very similar to (2.5): if

$$x_p \leq x_{p-1} \leq \cdots \leq x_1 < 0 \leq x_{p+1} \leq \cdots \leq x_n,$$

then it is straightforward to derive the bound

$$|E_n| \le \sum_{i=1}^{p} \gamma_{p-i+1}|x_i| + \sum_{i=p+1}^{n} \gamma_{n-i+1}|x_i| + \frac{u}{1-u}|\widehat{S}_n|. \tag{3.8}$$

In summary, the $+/-$ method appears to have no advantages over the other methods considered here, and in cases where there is heavy cancellation in the sum it can be expected to be the least accurate method.

The final method that we examine has an interesting background. In 1951, Gill [8] noticed that the rounding error in the sum of two numbers could be estimated by subtracting one of the numbers from the sum, and he made use of this estimate in a Runge–Kutta code in a program library for the EDSAC computer. Gill's estimate is valid for fixed point arithmetic only. Kahan [16] and Møller [31] both extended the idea to floating point arithmetic. Møller shows how to estimate $a + b - fl(a + b)$ in chopped arithmetic, while Kahan uses a slightly simpler estimate to derive a "compensated summation" method for computing $\sum_{i=1}^{n} x_i$. The use of Kahan's method with a Runge–Kutta formula is described in [41] (see also the experiments in [26]).

The estimate used by Kahan is perhaps best explained with the aid of a diagram. Let a and b be floating point numbers with $|a| \ge |b|$, let $\widehat{s} = fl(a + b)$, and consider Fig. 3.1, which uses boxes to represent the mantissas of a and b. The figure suggests that if we evaluate

$$e = -\big[\big((a+b) - a\big) - b\big] = (a - \widehat{s}) + b$$

in floating point arithmetic, in the order indicated by the parentheses, then the computed $\widehat{e}$ will be a good estimate of the error $(a + b) - \widehat{s}$. In fact, for rounded floating point arithmetic in base 2, we have

$$a + b = \widehat{s} + \widehat{e}, \tag{3.9}$$

that is, the computed $\widehat{e}$ represents the error exactly. This result (which does not hold for all bases) is proved by Dekker [5, Thm. 4.7], Knuth [22, Thm. C, p. 221], and Linnainmaa [26, Thm. 3]. Note that there is no point in computing $fl(\widehat{s} + \widehat{e})$, since $\widehat{s}$ is already the best floating point representation of $a + b$!

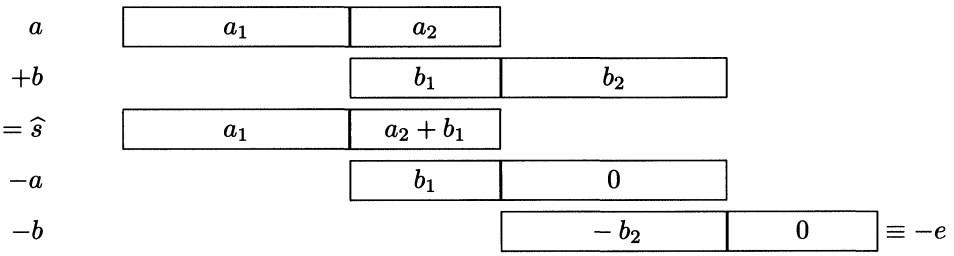

FIG. 3.1. *Recovering the rounding error.*

Kahan's compensated summation method employs the correction e on every step of a recursive summation. After each partial sum is formed, the correction is computed and immediately added to the next term x_i before that term is added to the partial sum.

Thus the idea is to capture the rounding errors and feed them back into the summation. The method may be written as follows.

```
s = 0; e = 0
for i = 1: n
    temp = s
    y = x_i + e
    s = temp + y
    e = (temp - s) + y
end
```

The compensated summation method has two weaknesses: $\widehat{e}$ is not necessarily the exact correction, since (3.9) is based on the assumption that $|a| \geq |b|$, and the addition $y = x_i + e$ is not performed exactly. Nevertheless, the use of the corrections brings a benefit in the form of an improved error bound. Knuth [22, Ex. 19, pp. 229, 572–573] shows that the computed sum $\widehat{S}_n$ satisfies

$$\widehat{S}_n = \sum_{i=1}^{n}(1+\mu_i)x_i, \qquad |\mu_i| \leq 2u + O(nu^2), \tag{3.10}$$

which is an almost ideal backward error result (a more detailed version of Knuth's proof is given in [9]).

In [17] and [18], Kahan describes a variation of compensated summation in which the final sum is also corrected (thus "$s = s+e$" is appended to the algorithm above). Kahan states in [17] and proves in [18] that (3.10) holds with the stronger bound $|\mu_i| \leq 2u + O\big((n-i+1)u^2\big)$; note that with this bound for $|\mu_i|$, (3.10) is essentially (2.3) with the n dependency transferred from the u term to the u^2 term. The proofs of (3.10) given by Knuth and Kahan are similar, and involve a subtle induction using the model (1.2).

The forward error bound corresponding to (3.10) is

$$|E_n| \leq \big(2u + O(nu^2)\big)\sum_{i=1}^{n}|x_i|. \tag{3.11}$$

As long as $nu \leq 1$, the constant in this bound is independent of n, and so the bound is a significant improvement over the bounds (2.6) for recursive summation and (3.6) for pairwise summation. Note, however, that if $\sum_{i=1}^{n}|x_i| \gg |\sum_{i=1}^{n} x_i|$, compensated summation is not guaranteed to yield a small relative error.

Another version of compensated summation is described in [14], [15], [21], [32], and [33]. Here, instead of immediately feeding each correction back into the summation, the corrections are accumulated by recursive summation and then the global correction is added to the computed sum. For this version of compensated summation it is shown in [21] and [32] that

$$\widehat{S}_n = \sum_{i=1}^{n}(1+\mu_i)x_i, \qquad |\mu_i| \leq 2u + n^2u^2, \tag{3.12}$$

provided $nu \leq 0.1$; this is weaker than (3.10) in that the second-order term has an extra factor n. If $n^2u \leq 0.1$ then in (3.12), $|\mu_i| \leq 2.1u$. In [14], it is shown that by using a divide and conquer implementation of compensated summation the range of n for which $|\mu_i| \leq cu$ holds in (3.12) can be extended, at the cost of a slight increase in the size of the constant c.

Finally, we mention briefly two further classes of algorithms. The first builds the sum in a series of accumulators, which are themselves added to give the sum. As originally described in [44], each accumulator holds a partial sum lying in a different interval. Each term x_i is added to the lowest level accumulator; if that accumulator overflows it is added to the next higher one and then reset to zero, and this cascade continues until no overflow occurs. Modifications of Wolfe's algorithm are presented in [28] and [36]. Malcolm [28] gives a detailed error analysis to show that his method achieves a relative error of order u. A drawback of the algorithm is that it is strongly machine dependent. An interesting and crucial feature of Malcolm's algorithm is that on the final step the accumulators are summed by recursive summation in order of *decreasing* absolute value, which in this particular situation precludes severe loss of significant digits and guarantees a small relative error.

Another class of algorithms, referred to as "distillation algorithms" by Kahan [19], iteratively constructs floating point numbers $x_1^{(k)}, \ldots, x_n^{(k)}$ such that $\sum_{i=1}^n x_i^{(k)} = \sum_{i=1}^n x_i$, terminating when $x_n^{(k)}$ approximates $\sum_{i=1}^n x_i$ with relative error at most u. Kahan states that these algorithms appear to have average run times of order at least $n \log n$. See [3], [19], [25] and [23] for further details and references.

4. Statistical estimates of accuracy. As we have noted, rounding error bounds can be very pessimistic, because they account for the worst-case propagation of errors. An alternative way to compare summation methods is through statistical estimates of the error, which may be more representative of the average case. A statistical analysis of three summation methods has been given by Robertazzi and Schwartz [35] for the case of nonnegative x_i. They assume that the relative errors in floating point addition are statistically independent and have zero mean and finite variance σ^2. Two distributions of nonnegative x_i are considered: the uniform distribution on $[0, 2\mu]$, and the exponential distribution with mean μ. Making various simplifying assumptions Robertazzi and Schwartz estimate the mean square error (that is, the variance of the absolute error) of the computed sums from recursive summation with random, increasing, and decreasing orderings, and from insertion summation and pairwise summation. Their results for the summation of n numbers are given in Table 4.1.

TABLE 4.1
Mean square errors.

Distribution	Increasing	Random	Decreasing	Insertion	Pairwise
Unif$(0, 2\mu)$	$0.20\mu^2n^3\sigma^2$	$0.33\mu^2n^3\sigma^2$	$0.53\mu^2n^3\sigma^2$	$2.6\mu^2n^2\sigma^2$	$2.7\mu^2n^2\sigma^2$
Exp(μ)	$0.13\mu^2n^3\sigma^2$	$0.33\mu^2n^3\sigma^2$	$0.63\mu^2n^3\sigma^2$	$2.6\mu^2n^2\sigma^2$	$4.0\mu^2n^2\sigma^2$

The results show that for recursive summation the ordering affects only the constant in the mean square error, with the increasing ordering having the smallest constant and the decreasing ordering the largest; since the x_i are nonnegative, this is precisely the ranking given by the rounding error bound (2.8). The insertion and pairwise summation methods have mean square errors proportional to n^2 rather than n^3 for recursive summation, and the insertion method has a smaller constant than pairwise summation. This is also consistent with the rounding error analysis, in which for nonnegative x_i the insertion method satisfies an error bound no larger than pairwise summation, and the latter method has an error bound with a smaller constant than for recursive summation ($\log_2 n$ versus n).

5. No guard digit model. The model (1.2) on which our error analysis is based is not valid on machines that lack a guard digit in addition, notable examples of which are Cray computers. On Cray computers, subtracting any power of 2 from the next smaller floating point number gives an answer that is either a factor of 2 too large or is zero, so the expression $fl(x+y) = (x+y)(1+\delta)$ holds with $|\delta| = 1$ but not with $|\delta| = O(u)$ [20]. For machines without a guard digit we have to use the weaker model [43, p. 12]

$$fl(x \pm y) = x(1+\alpha) \pm y(1+\beta), \qquad |\alpha|, |\beta| \le u. \tag{5.1}$$

We now summarize the effect on the rounding error analysis of using (5.1) in place of (1.2). The equality (2.4) remains valid provided we replace $(x_1+x_2)\theta_{n-1}$ by $x_1\theta_{n-1} + x_2\theta'_{n-1}$, so (2.5) and (2.6) are unchanged. The error expression (2.7) has to be replaced by

$$E_n = \sum_{k=2}^{n} (\widehat{S}_{k-1}\alpha_k + x_i\beta_k), \qquad |\alpha_k|, |\beta_k| \le u, \tag{5.2}$$

and so the analog of (2.8) is

$$|E_n| \le u \sum_{k=2}^{n} (|\widehat{S}_{k-1}| + |x_k|), \tag{5.3}$$

which is bounded above by $3u\sum_{k=1}^{n} |\widehat{S}_k| + O(u^2)$. Notice that (5.3) contains the term $\widehat{S}_1 = x_1$, which is not present in (2.8). The error expression (3.3) has to be replaced by an expression analogous to (5.2), and in (3.5) the factor $n-1$ has to be replaced by n. The bound (3.6) for pairwise summation remains valid under the no guard digit model, while in the bound (3.8) for the $+/-$ method we have to replace $u/(1-u)|\widehat{S}_n|$ by $u(|\widehat{S}_+| + |\widehat{S}_-|)$, which is bounded by $u\sum_{i=1}^{n} |x_i| + O(u^2)$.

Neither the correction formula (3.9) nor the result (3.10) for compensated summation holds under the no guard digit model. Indeed, Kahan [20] constructs an example where compensated summation fails to achieve (3.11) on Cray machines, but he states that such failure is extremely rare. In [17] and [18], Kahan gives a modification of the compensated summation algorithm in which the assignment "$e = (temp - s) + y$" is replaced by

$f = 0$
if $\text{sign}(temp) = \text{sign}(y)$, $f = (0.46 * s - s) + s$, end
$e = ((temp - f) - (s - f)) + y$

Kahan shows in [18] that the modified algorithm achieves (3.10) "on all North American machines with floating hardware" and explains that, "the mysterious constant 0.46, which could perhaps be any number between 0.25 and 0.50, and the fact that the proof requires a consideration of known machines designs, indicate that this algorithm is not an advance in computer science."

6. Numerical experiments. In this section we describe some numerical experiments that give further insight into the accuracy of summation methods. All the experiments were done using MATLAB [30], which uses IEEE standard double precision arithmetic with unit roundoff $u \approx 1.1 \times 10^{-16}$.

First, we illustrate the behavior of the methods on four classes of data $\{x_i\}$ chosen a priori. In these tests we simulated single precision arithmetic of unit roundoff

$u_{SP} = 2^{-23} \approx 1.2 \times 10^{-7}$ by rounding the result of every arithmetic operation to 23 significant bits. We formed an approximation to the exact answer S_n by summing the single precision numbers x_i in double precision by recursive summation; in each case $nu\sum_{i=1}^{n}|x_i| < u_{SP}|S_n|$, so (2.6) guarantees that this approximation is correct to single precision. We give results for recursive summation with the original (Orig.), increasing (Inc.), decreasing (Dec.), and Psum orderings, and for the insertion (Ins.) method, the $+/-$ method (with S_+ and S_- computed by recursive summation with the increasing ordering), pairwise summation with the increasing ordering (Pair.), and compensated summation (Comp.).

The numbers reported are the relative error $|\widehat{S}_n - S_n|/|S_n|$, together with information that indicates the sharpness of the bounds. In square brackets is the value $T = \sum_{k=n+1}^{2n-1}|\widehat{T}_k|$ in (3.4), for all methods except compensated summation. In parentheses is the ratio $R = |\widehat{S}_n - S_n|/(u_{SP}\sum_{i=1}^{n}|x_i|)$, which, according to the error analyses, is certainly bounded (to first order) by n for recursive summation and the insertion and $+/-$ methods, $\log_2 n$ for pairwise summation, and 2 for compensated summation. The quantities T and R reveal how close the strongest and weakest of the error bounds are to being equalities.

(1) In the first example, x_i is the ith term in the Taylor series expansion of e^{-x} about the origin, with $x = 2\pi$ (this series provides the classic example of "catastrophic cancellation" [37]). Results for $n = 64$ are given in Table 6.1. In this example, recursive summation with the decreasing ordering yields by far the best accuracy. There is severe cancellation in the sum and the decreasing ordering allows the terms of smallest modulus to contribute fully to the computed sum; in the other methods the small terms are "swamped" by the large terms. The error bounds do not reflect the merit of the decreasing ordering, because the T terms (in square brackets in the table) are of similar magnitude for the first four methods. Note also that compensated summation produces no improvement over recursive summation with the original ordering, and the $+/-$ method yields one less correct significant figure than all the other methods (as predicted by the T values).

TABLE 6.1
x_i *from* $e^{-2\pi}$ *expansion.* $|\sum_{i=1}^{n}x_i|/\sum_{i=1}^{n}|x_i| = 3.48e-6.$

n	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	$+/-$	Comp.
64	5.11e−4	2.27e−3	1.85e−7	2.27e−3	1.41e−4	2.27e−3	1.86e−2	5.11e−4
	[2.68e2	2.97e2	2.97e2	2.85e2	8.68e1	2.97e2	1.34e3]	
	(1.49e−2	6.64e−2	5.40e−6	6.64e−2	4.13e−3	6.64e−2	5.44e−1	1.49e−2)

(2) In this example the x_i are random numbers from the Normal(0,1) distribution and we report the results for $n = 2048$ and 4096 in Table 6.2. There is cancellation in both sums, although not as much as in the first example. Here the Psum ordering is clearly the best and the $+/-$ ordering the worst, and this is reflected in the T values.

The next two tests involve positive x_i, for which all the methods are guaranteed to produce a relative error no larger than $f(n)u$, where $f(n) \leq n$ depends on the method. (Note that for positive x_i, "Psum $\equiv$ $+/-$ $\equiv$ Inc.")

(3) We take $x_i = 1/i^2$ and examine how the errors vary with n for recursive summation with the decreasing and increasing orderings. Results for $n = 500, 1000, \ldots, 5000$ are given in Table 6.3. As would be expected in view of the error bounds of §2, the decreasing ordering provides much lower accuracy than the increasing ordering when n is large.

TABLE 6.2
x_i *from Normal* (0,1) *distribution.* $|\sum_{i=1}^{n} x_i| / \sum_{i=1}^{n} |x_i| = 8.08\text{e}-3 (n = 2048), 3.48\text{e}-3 (n = 4096)$.

n	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	+/−	Comp.
2048	7.47e−6	3.32e−6	7.17e−6	6.82e−8	6.60e−7	5.12e−7	1.20e−4	2.28e−7
	[3.06e4	1.53e4	2.65e4	8.26e2	2.87e3	2.32e3	4.80e5]	
	(5.06e−1	2.25e−1	4.86e−1	4.62e−3	4.47e−2	3.47e−2	8.15e0	1.54e−2)
4096	8.06e−6	1.04e−5	1.84e−6	2.66e−8	1.38e−7	6.87e−7	3.68e−4	1.92e−7
	[6.69e4	4.74e4	4.28e4	1.68e3	5.70e3	5.38e3	2.02e6]	
	(2.35e−1	3.04e−1	5.38e−2	7.76e−4	4.04e−3	2.00e−2	1.07e1	5.59e−3)

TABLE 6.3
$x_i = 1/i^2$.

n	500	1000	2000	3000	4000	5000
Inc.	1.04e−7	1.01e−7	1.74e−8	5.22e−8	1.36e−7	3.90e−8
Dec.	3.31e−7	6.24e−7	5.64e−6	2.30e−5	2.77e−5	5.81e−5

(4) In this example the numbers x_i are equally spaced on $[1, 2]$. We tried various $n \leq 4096$ and did not observe a great difference between the increasing and decreasing orderings; this is to be expected since the x_i vary little in magnitude. For the fairly large n in Table 6.4 pairwise summation out-performs recursive summation (the insertion method is equivalent to pairwise summation in this example). The errors for compensated summation are zero for all the n we tried!

TABLE 6.4
x_i *equispaced on* $[1, 2]$.

n	Inc.	Dec.	Pair.	Comp.
2048	2.86e−6	3.86e−5	1.59e−7	0.00
	[2.80e6	3.50e6	3.38e4]	
	(2.40e1	3.24e2	1.33e0	0.00)
4096	3.35e−5	2.18e−5	1.59e−7	0.00
	[1.12e7	1.40e7	7.37e4]	
	(2.81e2	1.83e2	1.33e0	0.00)

In the next set of tests we used a MATLAB implementation [12] of the multidirectional search (MDS) method [39], [40] which attempts to locate a maximizer of $f : \mathbb{R}^n \to \mathbb{R}$ using function values only. We applied the maximizer to f defined as the relative error of the sum computed in single precision by recursive summation with the increasing ordering. With $n = 3$, starting with $x_0 = [\frac{1}{3}, \frac{2}{3}, 1]$, the maximizer located the following set of data after 280 function evaluations:

$$x = [4.975987 \quad -2.495094 \quad -2.480894], \qquad f(x) = 1.0,$$
$$\widehat{S}_3 = -9.5367 \times 10^{-7}, \qquad S_3 = -4.7684 \times 10^{-7},$$

where x and the S_3 values are quoted to seven and five significant figures, respectively. With f defined as the relative error for compensated summation, the MDS maximizer made little progress with the same starting value. But starting with $x_0 = [-\frac{1}{3}, 0, \frac{2}{3}]$, the maximizer found after 166 function evaluations the data

$$x = [-0.8308306 \quad -0.7626623 \quad 1.593493], \qquad f(x) = 1.0,$$
$$\widehat{S}_3 = 2.3842 \times 10^{-7}, \qquad S_3 = 1.1921 \times 10^{-7}.$$

(If x is reordered with the increasing ordering then $f(x) = 1.0$, but $f(x) = 0$ for the decreasing ordering.)

These two examples are typical—using the maximizer it is straightforward to find data for which any of the summation methods yields no correct significant figures in the sum. The maximizer can also be used to compare two different methods, by defining f as the ratio of the errors from the two methods. With $n = 3$ we compared recursive summation (with the increasing ordering) with compensated summation. For both ratios of errors (E(Inc.)/E(Comp.) and its reciprocal) with certain starting values the maximizer was able to make the ratio arbitrarily large, by converging to data for which the error forming the denominator of f is zero. We observed similar behavior when comparing other methods.

Next, we describe an experiment with the forward substitution algorithm for solving a lower triangular system. We coded the inner product version of the algorithm and provided an option to choose between eight summation methods when evaluating the inner products. (The column-oriented form of forward substitution is not amenable to the use of different summation methods.) Lower triangular systems $Tx = b$ were solved in single precision and the forward error $\|\widehat{x} - x\|_\infty / \|x\|_\infty$ was evaluated for each of the eight summation options. We give results for $T = U^T$ where $PA = LU$ is the LU factorization with partial pivoting of the 10×10 Vandermonde matrix whose (i,j) element is $((j-1)/(n-1))^{i-1}$. In Table 6.5 we report results for the two systems with right-hand sides $b_i = Tx_i$, where x_i has elements equally spaced on the intervals $[1, 100]$ for $i = 1$ and $[0, 100]$ for $i = 2$. For this matrix $\kappa_\infty(T) = \|T\|_\infty \|T^{-1}\|_\infty \approx 3 \times 10^7$, and $\mathrm{cond}(T, x_1) \approx \mathrm{cond}(T, x_2) \approx 7 \times 10^5$, where $\mathrm{cond}(T, x) = \| \, |T^{-1}||T||x| \, \|_\infty / \|x\|_\infty \leq \kappa_\infty(T)$ is the condition number that appears in a forward error bound for the substitution algorithm [11]. The forward error varies over the different summation methods by a factor 98 for b_1 and a factor 39 for b_2; these are the largest variations we observed in tests with a variety of different matrices and right-hand sides. Throughout the tests there was no pattern to which summation method led to the smallest or largest forward error. This experiment shows that the choice of summation method for inner product evaluation can significantly affect the accuracy achieved by forward substitution, and this conclusion applies a fortiori to the solution of a full system via LU factorization. However, since there appears to be no straightforward way to predict which summation method will be the best for a given linear system, there is little reason to use anything other than recursive summation in the natural order when evaluating inner products within a general linear equation solver.

TABLE 6.5
Forward substitution with a 10×10 *matrix.*

	Orig.	Inc.	Dec.	Psum	Pair.	Ins.	+/−	Comp.
b_1	3.01e-4	1.18e-2	7.70e-3	1.18e-2	2.94e-2	1.18e-2	4.01e-3	7.70e-3
b_2	1.31e-2	2.64e-2	4.63e-3	2.64e-2	6.81e-4	1.06e-2	2.64e-2	2.04e-2

We have also experimented with compensated summation with the data arranged in order of decreasing magnitude. For all the problems we have tried, including those described above, the relative errors are $\leq u_{SP}$. Our attempts to use the MDS maximizer to find a set of x_i for which the relative error exceeds u_{SP} have been unsuccessful. It is therefore natural to ask whether a relative error bound of the form $|E_n| \leq cu|S_n|$ can be derived, where c is a constant independent of the x_i. The answer is no, because E_n can

can be nonzero when $S_n = 0$. Even when $n = 3$ and $S_n \neq 0$ it appears to be impossible to obtain such a bound. Nevertheless, our (limited) experience suggests that compensated summation with the decreasing ordering performs remarkably well in practice, and it would be interesting to determine why this is so.

Further test results can be found in the literature, although none are extensive. Linz [27] compares recursive summation with pairwise summation for uniform random numbers on $[0, 1]$ with $n = 2048$, averaging the errors over 20 trials, and Caprani [4] and Gregory [10] both conduct a similar experiment including compensated summation as well. Linnainmaa [26] applies recursive summation and compensated summation to series expansions, Simpson's rule for quadrature and Gill's Runge–Kutta method. Robertazzi and Schwartz [35] evaluate average mean square errors for recursive summation (with increasing, decreasing, and random orderings), pairwise summation and the insertion method, for the uniform $[0, 1]$ and exponential ($\mu = 0.5$) distributions with $n \leq 4096$.

7. Concluding remarks. No summation method from among those considered here can be regarded as superior to the rest from the point of view of accuracy, since for each method the error can vary greatly with the data, within the freedom afforded by the error bounds. However, some specific conclusions can be drawn.

1. For all but two of the methods the errors are, in the worst case, proportional to n. If n is very large, pairwise summation (error constant $\log_2 n$) and compensated summation (error constant of order 1) are attractive.

2. If the x_i all have the same sign, then all the methods yield a relative error of at most nu, and compensated summation guarantees perfect relative accuracy (as long as $nu \leq 1$). For recursive summation of one-signed data the increasing ordering is preferable to the decreasing ordering (and it is equivalent to the Psum ordering); however, the insertion method has the smallest bound (3.4) over all the methods considered here (excluding compensated summation).

3. For sums with heavy cancellation ($\sum_{i=1}^n |x_i| \gg |\sum_{i=1}^n x_i|$) recursive summation with the decreasing ordering is attractive (see Table 6.1), although it cannot be guaranteed to achieve the best accuracy (see Table 6.2).

Considerations of computational cost and the way in which the data are generated may rule out some of the methods. Recursive summation in the natural order, pairwise summation, and compensated summation can be implemented in $O(n)$ operations for general x_i, but the other methods are more expensive since they require searching or sorting. Furthermore, in an application such as the numerical solution of ordinary differential equations where the x_i are generated sequentially, and x_k may depend on $\sum_{i=1}^{k-1} x_i$, sorting and searching may be impossible. One way to achieve higher accuracy that we have not mentioned is simply to implement recursive summation in higher precision; if this is feasible, it may be less expensive (and more accurate) than using one of the alternative methods in working precision.

Finally, we return to the two practical applications mentioned in the introduction. In [24], the $+/-$ method was found to cure some problems with inaccurate gradients in an optimization method. This is a little surprising since we have found the $+/-$ method to be unattractive. It appears that there is some feature of this application, not apparent from [24], that encourages the $+/-$ method to perform better than recursive summation with the natural ordering. The loss of symmetry in a quasi-Newton method that was observed in [7] is easier to understand. For example, symmetries in F in (1.1) can be preserved by using any summation method whose computed answer does not depend on the given order of the data—such as recursive summation with the increasing ordering and with elements of equal magnitude ordered by sign.

Acknowledgments. I thank Jeremy Du Croz and Philip Gill for providing some of the references, and Des Higham, Nick Trefethen, and the referee for suggesting improvements to the manuscript.

REFERENCES

[1] I. BABUŠKA, *Numerical stability in mathematical analysis*, in Proc. IFIP Congress, Information Processing 68, North-Holland, Amsterdam, 1969, pp. 11–23.
[2] ———, *Numerical stability in problems of linear algebra*, SIAM J. Numer. Anal., 9 (1972), pp. 53–77.
[3] G. BOHLENDER, *Floating-point computation of functions with maximum accuracy*, IEEE Trans. Comput., C-26 (1977), pp. 621–632.
[4] O. CAPRANI, *Implementation of a low round-off summation method*, BIT, 11 (1971), pp. 271–275.
[5] T. J. DEKKER, *A floating-point technique for extending the available precision*, Numer. Math., 18 (1971), pp. 224–242.
[6] J. W. DEMMEL, *Underflow and the reliability of numerical software*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 887–919.
[7] L. C. W. DIXON AND D. J. MILLS, *The effect of rounding error on the variable metric method*, Tech. Rep. 229, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, UK, Apr. 1990.
[8] S. GILL, *A process for the step-by-step integration of differential equations in an automatic digital computing machine*, Proc. Cambridge Phil. Soc., 47 (1951), pp. 96–108.
[9] D. GOLDBERG, *What every computer scientist should know about floating-point arithmetic*, ACM Comput. Surveys, 23 (1991), pp. 5–48.
[10] J. GREGORY, *A comparison of floating point summation methods*, Comm. ACM, 15 (1972), p. 838.
[11] N. J. HIGHAM, *The accuracy of solutions to triangular systems*, SIAM J. Numer. Anal., 26 (1989), pp. 1252–1265.
[12] ———, *Optimization by direct search in matrix computations*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 317–333.
[13] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, Bristol, 1981.
[14] M. JANKOWSKI, A. SMOKTUNOWICZ, AND H. WOŹNIAKOWSKI, *A note on floating-point summation of very many terms*, J. Inform. Process. Cybern., 19 (1983), pp. 435–440.
[15] M. JANKOWSKI AND H. WOŹNIAKOWSKI, *The accurate solution of certain continuous problems using only single precision arithmetic*, BIT, 25 (1985), pp. 635–651.
[16] W. KAHAN, *Further remarks on reducing truncation errors*, Comm. ACM, 8 (1965), p. 40.
[17] ———, *A survey of error analysis*, in Proc. IFIP Congress, Ljubljana, Information Processing 71, North-Holland, Amsterdam, 1972, pp. 1214–1239.
[18] ———, *Implementation of algorithms (lecture notes by W. S. Haugeland and D. Hough)*, Tech. Rep. 20, Dept. of Computer Science, Univ. of California, Berkeley, 1973.
[19] ———, *Doubled-precision* IEEE *standard 754 floating-point arithmetic*, manuscript, Feb. 1987.
[20] ———, *How Cray's arithmetic hurts scientific computation (and what might be done about it)*, manuscript prepared for the Cray User Group meeting in Toronto, June 1990.
[21] A. KIELBASIŃSKI, *Summation algorithm with corrections and some of its applications*, Math. Stos., 1 (1973), pp. 22–41. (In Polish, cited in [14] and [15].)
[22] D. E. KNUTH, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, 2nd Ed., Addison-Wesley, Reading, MA, 1981.
[23] U. W. KULISCH AND W. L. MIRANKER, *The arithmetic of the digital computer: A new approach*, SIAM Rev., 28 (1984), pp. 1–40.
[24] L. S. LASDON, J. PLUMMER, B. BUEHLER, AND A. D. WAREN, *Optimal design of efficient acoustic antenna arrays*, Math. Prog., 39 (1987), pp. 131–155.
[25] H. LEURECHT AND W. OBERAIGNER, *Parallel algorithms for the rounding exact summation of floating point numbers*, Computing, 28 (1982), pp. 89–104.
[26] S. LINNAINMAA, *Analysis of some known methods of improving the accuracy of floating-point sums*, BIT, 14 (1974), pp. 167–202.
[27] P. LINZ, *Accurate floating-point summation*, Comm. ACM, 13 (1970), pp. 361–362.
[28] M. A. MALCOLM, *On accurate floating-point summation*, Comm. ACM, 14 (1971), pp. 731–736.
[29] D. D. MCCRACKEN AND W. S. DORN, *Numerical Methods and Fortran Programming*, John Wiley, New York, 1964.
[30] C. B. MOLER, J. N. LITTLE, AND S. BANGERT, *PC-Matlab User's Guide*, The MathWorks, Inc., Natick, MA, 1987.

[31] O. MØLLER, *Quasi double-precision in floating point addition*, BIT, 5 (1965), pp. 37–50.
[32] A. NEUMAIER, *Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen*, Z. Angew. Math. Mech., 54 (1974), pp. 39–51.
[33] K. NICKEL, *Das Kahan-Babuškasche Summierungsverfahren in Triplex-ALGOL* 60, Z. Angew. Math. Mech., 50 (1970), pp. 369–373.
[34] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
[35] T. G. ROBERTAZZI AND S. C. SCHWARTZ, *Best "ordering" for floating-point addition*, ACM Trans. Math. Software, 14 (1988), pp. 101–110.
[36] D. R. ROSS, *Reducing truncation errors using cascading accumulators*, Comm. ACM, 8 (1965), pp. 32–33.
[37] I. A. STEGUN AND M. ABRAMOWITZ, *Pitfalls in computation*, J. Soc. Indust. Appl. Math., 4 (1956), pp. 207–219.
[38] F. STUMMEL, *Rounding error analysis of elementary numerical algorithms*, Computing, Suppl., 2 (1980), pp. 169–195.
[39] V. J. TORCZON, *Multi-directional Search: A direct search algorithm for parallel machines*, Ph.D. thesis, Rice University, Houston, TX, May 1989.
[40] ———, *On the convergence of the multi-directional search algorithm*, SIAM J. Optimization, 1 (1991), pp. 123–145.
[41] E. VITASEK, *The numerical stability in solution of differential equations*, in Conference on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics 109, Springer-Verlag, Berlin, 1969, pp. 87–111.
[42] J. H. WILKINSON, *Error analysis of floating-point computation*, Numer. Math., 2 (1960), pp. 319–340.
[43] ———, *Rounding Errors in Algebraic Processes*, Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963.
[44] J. M. WOLFE, *Reducing truncation errors by programming*, Comm. ACM, 7 (1964), pp. 355–356.