AN ALGORITHM FOR FLOATING-POINT ACCUMULATION OF SUMS WITH SMALL RELATIVE ERROR

BY

MICHAEL MALCOLM

STAN-CS-70-163
JUNE 1970

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

AN ALGORITHM FOR FLOATING-POINT ACCUMULATION OF SUMS WITH SMALL RELATIVE ERROR

by

Michael Malcolm

The preparation of this report was sponsored by the Office of Naval Research under grant number N0013-67-A-0112-0029, the National Science Foundation under grant number NSF GJ 408 and the Atomic Energy Commission under grant number AT (04-3) 326,PA 30.

I. Introduction

Many algorithms require the calculation of a sum

$$s = \sum_{i=1}^{n} x_i \quad , \quad n > 3 \quad ,$$

where $x_1, x_2, \ldots, x_n$ are numbers-represented in floating point. In practice, an approximate sum $\hat{s}$ is computed with rounding errors. Wilkinson [1] shows that if the sum is accumulated in a single-precision accumulator (using floating binary arithmetic with t bits of precision and proper rounding), then

$$\hat{s} - s = \sum_{i=1}^{n} x_i \eta_i \quad ,$$

where

$$(1 - 2^{-t})^{n+1-r} \leq 1 + \eta_r \leq (1 + 2^{-t})^{n+1-r} \qquad (r = 1, \ldots, n) \ .$$

Thus the error bound is dependent on the order of summation. This result has led to the well-known rule of thumb that it is usually best to add a list of numbers in order of increasing magnitude. If one has a priori knowledge of the x_i (e.g., $\sum |x_i| < 1$) or if the accumulation is performed with more precision (say double precision), then much smaller error bounds can be found. However, as Wilkinson points out, "It should be emphasized that we still cannot guarantee that an accumulated sum . . . has a low _relative_ error."

Large relative error in an accumulated sum is often the result of a phenomenon which Professor D. H. Lehmer calls _catastrophic cancellation._ This occurs when an intermediate partial sum is much larger in magnitude than the final sum. Then one or more additions result in a loss of significant digits. The post-normalization step of a subsequent addition thus introduces zeros in place of significant digits. However, as Professor William Kahan has observed, this large cancellation is not the cause of the error -- it merely reveals the error. That is, the real villain here is not the cancellation, but rather the large intermediate sums within a floating-point system of given precision. Perhaps "catastrophic loss of precision" would be a more appropriate name. **Catastophic** cancellation is fairly common with poorly designed algorithms; most good algorithms have built-in precautions which avoid (or usually avoid) this phenomenon.

Large relative errors can occur without catastrophic cancellation. This happens in large summations $(n >> 3)$ where the intermediate sums become much larger in magnitude than the individual addends, but not larger than the final sum. This sort of error can occur in numerical integration using a large number of intervals. Wolfe [2] proposed a technique for avoiding this type of error. It is described in the following section.

In the remainder of this report, a modification of Wolfe's algorithm is presented, followed by a detailed error analysis. This algorithm has the advantage that the final sum is guaranteed to have a very small <u>relative</u> error.

II. <u>Extended Summation With Cascading Accumulators</u>

Wolfe [2] suggests a technique which is easily programmed and requires only a small number of additional storage locations. These extra locations, called <u>cascading accumulators,</u> are denoted by sl, s2, ... The separate accumulators hold sums that are in various intervals; for example,

$$\begin{array}{ccccc}
1.000 & \leq & c(s1) & \leq & 9.999 \\
10.00 & \leq & c(s2) & \leq & 99.99 \\
100.0 & \leq & c(s3) & \leq & 999.9 \\
\vdots & & \vdots & & \vdots
\end{array}$$

where c(si) denotes the contents of si . The summing is done at the lowest level accumulator (s1) until it is about to overflow. At that point it is added to the next accumulator (s2) and reset to zero. Similarly, if s2 is about to overflow, it is added to s3 and reset to zero, and so on.

By this technique the intermediate sums never become much larger than the addends. However, catastrophic cancellation can occur just as before. Wolfe does not discuss how to go about summing the accumulators at the end; in an example he uses the order of increasing magnitude. For certain problems, this is a useful-technique; however, there is no guarantee that the final result has a small relative error.

III. A Modification of Wolfe's Algorithm

The following algorithm requires little if any more execution time than the algorithm of the last section, and nearly full-precision accuracy is achieved, provided exponent underflow or overflow do not occur. Such exceptional conditions are normally brought to the attention of the user by the system software and, if so, inaccurate results cannot go unnoticed. As in Wolfe's algorithm, additional intermediate accumulators are used -- typically fewer than 50.

The following discussion assumes the algorithm is implemented on a machine using a floating-point number system F of base β (usually β is 2, 8, 10 or 16) with a t-digit mantissa. The exponent e is assumed to lie in the range

$$-m \leq e < M \quad .$$

Thus each nonzero $x \epsilon F$ has the normalized representation

$$x = \pm .d_1 d_2 . . . d_t \cdot \beta^e \quad , \tag{1}$$

where $d_1,$ [illegible] are integers satisfying

$$1 \leq d_1 \leq \beta-1 \ ,$$

$$0 \leq d_i \leq \beta-1 \qquad (i = 2,\ldots,t) \ .$$

The number 0 belongs to F , and has the structure

$$0 = .00\ldots0 \cdot \beta^{-m} \ .$$

All floating-point addition is assumed to be normalized. The machine may do either proper rounding or truncation (chopping).

To facilitate discussion, the function Rev (similar to that used by Møller[5]) is defined as follows: If $x \epsilon F$ then $\ell ev(x) = e + m$. ℓev is the biased exponent having the mnemonic "level". Note that ℓev is a function of the representation of a number and not the number itself. For example, suppose x is to be added to y , where $|y| > |x|$, and that x must be unnormalized during operand alignment. Suppose also that no nonzero digits are lost from the mantissa of x while it is being unnormalized. If we denote the unnormalized representation of x by $\hat{x}$, then x and $\hat{x}$ both represent the same real number exactly, but $\ell ev(\hat{x}) > \ell ev(x)$.

The algorithm for computing $\sum_{i=1}^{n} x_i$ can now be described as follows. There are two positive parameters, ℓ and η :

Assume there are $\eta+1$ accumulators, the contents of which are denoted by $\alpha_0, \alpha_1, \ldots, \alpha_\eta$.

1. Set each of the accumulators to zero.
2. For each x_i form $a_{i1}, a_{i2}, \ldots, a_{iq}$ $(q \geq 1)$, where $a_{i1} + a_{i2} + \ldots + a_{iq} = x_i$ and each a_{ij} has the property that the last ℓ digits are 0 (i.e., $d_{t-\ell+1} = d_{t-\ell} = \ldots = d_t = 0$).
3. Each a_{ij} is added to the k-th accumulator, where k is determined by

$$\nu k \leq \ell ev(a_{ij}) < \nu k + \nu - 1 ,$$
$$\nu = \lceil (M + m + 1)/(\eta + 1) \rceil \qquad (2)$$

where $\lceil \xi \rceil$ denotes the smallest integer not less than ξ. (Thus

$$k = \ell ev(a_{ij}) \div \nu , \qquad (3)$$

in the sense of Algol 60.)

4. The accumulators are summed in <u>decreasing</u> order (i.e., $\eta, \eta-1, \ldots, 0$).

The second step appears, at first sight, to be quite complicated. However, in practice it is easily done, especially on a machine with double-precision arithmetic. An illustration of this in Fortran for the IBM System/360 is contained in Section VI.

The parameters ℓ and η are chosen so that the addition in step 3 retains all the significant digits involved. That is, until step 4, there are no rounding errors. More insight into choosing ℓ and η will be given in the following section. Also, an important restriction on the magnitude of the product qn will be revealed.

Step number 4 is certainly the most interesting step of the algorithm. If, instead, the accumulators are summed in increasing order (as one is tempted to do after reading Wilkinson [1]), catastrophic cancellation can occur. When this algorithm is incorporated in an innerproduct routine, it often happens that

$$\begin{aligned}
\alpha_\eta &= 0 \\
&\vdots \\
\alpha_{k+1} &= 0 \\
\alpha_k &= -B \\
\alpha_{k-1} &= B \\
\alpha_{k-2} &= 0 \\
\alpha_{k-3} &= 0 \\
\alpha_{k-4} &= \varepsilon_1 \\
\alpha_{k-5} &= \varepsilon_2 \\
&\vdots
\end{aligned}$$

where $\ell ev(B) - \ell ev(\varepsilon_i) > t$. Summing in increasing order will yield 0 . However, as D. Jordan pointed out in [4], summing the accumulators in decreasing order $(\eta,\ldots,0)$ precludes the chance of this type of error. The remaining question is: Does summing the accumulators in decreasing order lead to some other case where a large relative roundoff error can occur? The answer to this question is no. Proof of this assertion and a sharp bound on the roundoff error are given in the next section.

IV. Error Analysis

Another convenient function is defined as follows: Let $x \in F$ be an approximation of some real number x^* . If $x=x^*$ and $x^* \neq 0$, then pad(x,x*) is defined to be the number of digits by which the mantissa of x can be shifted to the right before a significant digit is lost (i.e., before a non-zero digit is shifted out of the low-order position). If $x \neq x^*$, then pad(x,x*) is negative, and defined as follows: suppose x has the representation (1); if there exists a ζ such that ζ can be represented as

$$\zeta = \pm .\hat{d}_1\hat{d}_2\ldots\hat{d}_T \cdot \beta^{e-t} \quad \text{with } \hat{d}_T \neq 0 , \ \hat{d}_1 \neq 0$$

and $x + \zeta = x^*$ and T is finite, then pad(x,x*) is defined to be $-T$. Otherwise, pad(x,x*) is defined as $-\infty$. For completeness, $\mathrm{pad}(0,0) = \infty$. For example, if $\beta = 2$ and $t = 6$, $\mathrm{pad}(-.101000 \cdot 2^3, -5_{10}) = 3$. If $x = +.111111 \cdot 2^0$ and $y = +.111111 \cdot 2^1$, and $\oplus$ represents floating-point addition, and $\mathrm{pad}(x \oplus y, x + y) = -2$ since two digits are lost during the floating-point addition. When pad(x,x*) is positive, the mantissa of x has a "padding" of zero digits at the end.

It follows that

$$\mathrm{pad}(x,x^*) > 0 \Leftrightarrow x = x^* ,$$

$$\mathrm{pad}(x,x^*) < 0 \Leftrightarrow x \neq x^* ,$$

$$\mathrm{pad}(x,x^*) \geq t \Rightarrow \mathrm{pad}(x,x^*) = \infty \Leftrightarrow x = x^* = \square .$$

In step 2 of the algorithm, it is required that $\mathrm{pad}(a_{ij}, a_{ij}) \geq \ell > 0$, for all i,j .

It is also expedient to define

$$\rho(x,x^*) = \ell ev(x) + \mathrm{pad}(x,x^*) . \qquad (4)$$

$\rho(x,x^*)$ is invariant with respect to operand alignment (un-normalization) and post-normalization of x , provided no exponent underflow or overflow occurs.

<u>Lemma 1</u>: If x and y are two floating-point numbers and $\oplus$ represents floating-point addition, then

$$\rho(x \oplus y, x^* + y^*) \geq \min\{\rho(x,x^*), \rho(y,y^*)\} ,$$

provided no exponent underflow or overflow occurs.

<u>Proof:</u> Assume $\ell ev(x) \geq \ell ev(y)$. Let z denote an accumulator, a floating-point number with a $t+2$ digit mantissa and an overflow digit. Set $z \leftarrow y$ and, if necessary, unnormalize z so that $\ell ev(z) = \mathrm{Rev}(x)$. The accumulator z can be treated as a floating-point number if one ignores the overflow digit and

considers only the first t digits of the mantissa. In this way, pad is defined for z . Let w denote another accumulator with the same structure as z . Set $w \leftarrow z+x$. Prior to the post-normalization step in forming w ,

$$\ell ev(w) = \ell ev(z) = \ell ev(x) \text{ and}$$

$$\mathrm{pad}(w,x^*+y^*) \geq \min(\mathrm{pad}(z,y^*),\mathrm{pad}(x,x^*)) ,$$

and equality occurs whenever the low-order digits of x and y don't cancel. From Equation (4) and the fact that $\rho(w,x^*+y^*)$ remains unchanged during the post-normalization step, it follows that

$$\rho(x\oplus y,x^*+y^*) = \rho(w,x^*+y^*) \geq \min\{\rho(z,y^*),\rho(x,x^*)\} .$$

Since $\rho(z,y^*) = \rho(y,y^*)$,

$$\rho(x\oplus y,x^*+y^*) \geq \min\{\rho(x,x^*),\rho(y,y^*)\} .$$

__Lemma 2__: In the notation of Section III,

$$\rho(\alpha_k,\alpha_k^*) \geq \nu k + \ell \ .$$

__Proof:__ Any term (y) that is added to the k-th accumulator satisfies

$$\ell ev(y) \geq \nu k \quad \text{and} \quad pad(y,y) \geq \ell \ .$$

By Lemma 1, Equation (4) and the fact that $\rho(0,0) = \infty$, the lemma follows by induction.

__Lemma 3__: If $\alpha_k \neq 0$, and N is the number of a_{ij} added to the accumulators, then

$$\nu k \leq \ell ev(\alpha_k) \leq \nu(k+1) -1 \qquad , \quad \text{if } N = 1$$

$$\nu k - t + \ell + 1 < \ell ev(\alpha_k) \leq \nu(k+1) + \lfloor \ell og_\beta(N-1) \rfloor \quad , \quad \text{if } N > 1$$

$$(k = 0,1,\ldots,\eta)$$

where $\lfloor \xi \rfloor$ denotes the largest integer not greater-than ξ .

__Proof:__ The inequalities for N=1 are obviously the same as those satisfied by a single term added to the k-th accumulator. The upper bound for N > 1 is found by considering the largest number (ζ) which can be added to the k-th accumulator, i.e.,

$$\zeta \text{ [illegible] } \cdot \beta^{\nu(k+1)-1-m} \quad , \qquad (z = \beta-1)$$

and observing the $\ell ev(\alpha_k)$ during repeated additions of ζ to α_k . If the lower bound for $N > 1$ were not true, i.e., if

$$\ell ev(\alpha_k) < \nu k - t + \ell + 1 \quad ,$$

then, by Lemma 2,

$$pad(\alpha_k, \alpha_k^*) \geq t \Rightarrow \alpha_k = 0 \quad ,$$

which is a contradiction.

Lemma 4: If $N \leq \beta^{\ell - \nu + 1}$, then each of the α_k $(k = 0,1,\ldots,\eta)$ is exact.

Proof: By-Lemma 3,

$$\ell ev(\alpha_k) < \nu k + \ell + 1 \quad .$$

Combining this with Equation (4) and Lemma 2 gives

$$pad(\alpha_k, \alpha_k^*) \geq 0 \quad ,$$

which, by the definition of pad , implies α_k is exact.

Loss of precision in an extended summation can result from either

1. repeated truncations (roundings) of the sum, or
2. post-normalization left shift of the approximate sum.

The post-normalization error can be formalized as follows: Let the accumulation of the floating-point sum

$$\psi_n = \sum_{i=1}^{n} x_1 \quad ,$$

where the x_1 $(i = 1,2,\ldots,n)$ are exact, be defined as

$$\psi_0 = 0$$

$$\psi_i = \psi_{i-1} \oplus x_i \quad , \quad (i = 1,2,\ldots,n) \ ,$$

The function A of two floating-point variables is defined as

$$\Delta(x, y) = \max\{\ell ev(x), \ell ev(y)\} - \ell ev(x \oplus y) .$$

Thus, during post-normalization of the floating-point sum $x \oplus y$, the mantissa undergoes a left shift of $\Delta(x,y)$ digits. Clearly, if a carry occurs, $\Delta(x,y) = -1$. Also, $\Delta(x,y) > 1$ only if $|\ell ev(x) - \ell ev(y)| \leq 1$.

During the formation of $\psi_i = \psi_{i-1} \oplus x_i$, any truncation error already present in the low order digits of ψ_{i-1} is multiplied by $\beta^{\Delta(\psi_{i-1}, x_i)}$.

The accumulators are summed in decreasing order. Thus, the sum S_0 can be defined by

$$S_{\eta+1} = 0$$

$$S_k = S_{k+1} \oplus \alpha_k \quad (k = \eta, \eta-1, \ldots, 0) \quad . \tag{5}$$

<u>Lemma 5</u>: If S_{k+1} is exact and $\ell ev(S_{k+1}) \leq \ell ev(\alpha_k)$ and if S_{k+1} is un-normalized so that $\ell ev(S_{k+1}) = \ell ev(\alpha_k)$, then $pad(S_{k+1}, S^*_{k+1}) > 0$, provided $N \leq \beta^{\ell-\nu+1}$.

<u>Proof</u>: Lemmaland Equation (5) yield

$$\rho(S_{k+1}, S_{k+1}) \geq \min\{\rho(\alpha_{k+1}, \alpha_{k+1}), \rho(\alpha_{k+2}, \alpha_{k+2}), \ldots, \rho(\alpha_\eta, \alpha_\eta)\} ,$$

which, with Lemma 2 and the definition of ρ , gives

$$\ell ev(S_{k+1}) \;\square\; pad(S_{k+1},S_{k+1}) \geq \nu(k+1) \;\square\; \ell \quad .$$

Substituting $\ell\, ev(\alpha_k)$ for $\ell ev(S_{k+1})$ and using Lemma 3, we obtain the desired result:

$$pad(S_{k+1},S^*_{k+1}) \geq \ell - \lfloor \ell og_\beta(N-1) \rfloor \quad .$$

Suppose that, in the process of summing the accumulators as described by Equation (5), k = j is the firs-t k such that $pad(S_k,S^*_k) < 0$. As a result of Lemma‘5, the truncation (rounding) error in adding S_{j+1} and α_j must be caused in one of three ways:

i) When the operands are aligned, $pad(\alpha_j,\alpha^*_j) = 0$ and a carry occurs.

ii) When the operands are aligned, $pad(S_{j+1},S^*_{j+1}) = 0$ and a carry occurs.

iii) $\ell ev(S_{j+1}) > \ell ev(\alpha_j)$ and, when α_j is aligned so that $\ell ev(\alpha_j) = \ell ev(S_{j+1})$, $pad(\alpha_j,\alpha^*_j) < -\Delta(S_{j+1},a_j) \leq 0$.

<u>Lemma 6</u>:

$$\ell ev(S_j) > \nu j + \ell + 1 \quad .$$

<u>Proof</u>: Case i) In the aligned position, using Lemma 2, we find that

$$\ell ev(\alpha_j) = \rho(\alpha_j,\alpha^*_j) \geq \nu j + \ell \quad .$$

Thus $\ell ev(S_j) = \ell ev(\alpha_j) + 1 > \nu j + \ell + 1$.

Case ii) $\ell ev(S_{j+1}) = \rho(S_{j+1},S^*_{j+1}) \geq \min\{\rho(\alpha_{j+1},\alpha^*_{j+1}), \ldots, \rho(\alpha_\eta,\alpha^*_\eta)\}$

$$\geq \nu j + \nu + \ell \quad .$$

Therefore $\ell ev(S_j) > \nu j + \nu + \ell > \nu j + \ell + 1$.

Case iii) When α_j is aligned, $\ell ev(S_{j+1}) = \ell ev(\alpha_j) > \rho(\alpha_j,\alpha_j^*) \geq \nu j + \ell$.

Now,

$$\ell ev(S_j) = \ell ev(S_{j+1}) + \Delta(S_{j+1},\alpha_j)$$

$$\geq \ell ev(S_{j+1}) > \ell ev(\alpha_j) = \nu j + \ell \ .$$

Thus,

$$\ell ev(S_j) > \nu j + \ell + 1 \ .$$

Since $\ell ev(\alpha_{j-1}) \leq \nu j + \lfloor \ell og_\beta(N-1) \rfloor$,

$$\ell ev(S_j) - \ell ev(\alpha_{j-1}) \geq \ell + 1 - \lfloor \ell og_\beta(N-1) \rfloor \ .$$

The assumption in Lemma 4 (i.e., $N \leq \beta^{\ell-\nu+1}$) is sufficient to guarantee that $\ell ev(S_j) - \ell ev(\alpha_{j-1}) > 1$, from which it follows that $A(S_j,\alpha_{j-1}) \leq 1$. Similarly, each of the subsequent additions can undergo a **post-normalization** left **shift** of at most one digit. In fact, at most one of the additions

$$S_k = S_{k+1} \oplus \alpha_k \qquad (k = j-1, j-2, \ldots, 0)$$

will undergo a post-normalization left shift of one digit.

<u>Lemma 7</u>: If $N \leq \beta^{\ell-\nu+1}$, the mantissa of each of the accumulators $\alpha_{j-\lambda}, \alpha_{j-\lambda-1}, \ldots, \alpha_0$ is shifted at least t digits during operand alignment, where

$$\lambda = \lceil (t+1)/\nu \rceil \ . \qquad (6)$$

Proof: By Lemma 6,

$$\ell ev(S_{j-i}) \geq \nu j + \ell \; , \quad (i = 0,1,\ldots,j) \; .$$

By Lemma 3,

$$\ell ev(\alpha_{j-i}) \leq \nu(j-i) + \ell + 1 \; , \quad (i = 0,1,\ldots,j) \; ,$$

and $\ell ev(S_{j-i+1}) - \ell ev(\alpha_{j-i}) \geq \nu i - 1 \geq t$, $(i = \lambda,\lambda+1,\ldots,j)$.

Thus the mantissa of each of the accumulators $\alpha_{j-\lambda}, \alpha_{j-\lambda-1}, \ldots$ is shifted at least t digits during operand alignment.

Theorem 1: If $N \leq \beta^{\ell-\nu+1}$, if the accumulator used in accumulating S_0 has at least $t+1$ digits, and if no underflow or overflow occurs, then the absolute error in S_0 is bounded by

$$|s - S_0| \leq \lambda \delta \beta^{\ell ev(S_0)-m-t+1} \; ,$$

where

$$\delta = \begin{cases} 1 & \text{for chopped arithmetic} \\ \frac{1}{2} & \text{for rounded arithmetic} \end{cases}$$

and λ is given by Equation (6).

Proof: Since a post-normalization left shift of at most one digit can occur only once while the accumulators are summed, the worst case occurs when it is caused by the addition of $\alpha_{j-\lambda}$ (see Lemma 7). Subsequent additions of $\alpha_{j-\lambda-1}, \alpha_{j-\lambda-2}, \ldots$ cannot affect the computed value of S_0 (see Knuth's [6] discussion of problem 5, page 498). Prior to the addition of α_{j-h} , a maximum of λ truncations (roundings) can occur, each resulting in an error of $\beta^{\ell ev(S_0)-m-t}$ or less.

Q.E.D.

For machines which use t-digit accumulators and chopped arithmetic, the error bound is $(\lambda-1)\beta^{\ell ev(S_0)-m-t+1}$ (a stronger result!). Note that, although the above theorem gives a bound on the absolute error, it also provides a bound on the relative error. Specifically, if the true value of the sum s is zero, then $S_0 = 0$.

Theorem 2: If $N \leq \beta^{\ell-\nu+1}$ and no underflow or overflow occurs, then

$$s = S_0(1+\varepsilon) ,$$

where

$$|\varepsilon| \leq \lambda\delta\beta^{2-t} .$$

Proof: i) If $S_0 = 0$, then, since total cancellation of significant digits cannot occur in summing the accumulators, s = 0 .

ii) If $S_0 \neq 0$, then assume $s - S_0 = S_0\varepsilon$. By Theorem 1,

$$|s - S_0| = |S_0|\,|\varepsilon| \leq \lambda\delta\beta^{\ell ev(S_0)-m-t+1} .$$

Since $|S_0| \geq \beta^{\ell ev(S_0)-m-1}$,

$$|\varepsilon| \leq \lambda\delta\beta^{2-t} .$$

These theoretical results are substantiated by an experiment reported by D. Jordan [4]. Jordan used this technique for accumulating innerproducts on an IBM 360 (β = 16, t = 14). He chose $\eta = 32$, $\ell = 6$ and $q = 2$, and states:

"Empirical-tests were run to determine the small amount of roundoff that might be expected from the procedure. The tests used 1000 dot products of 15-component vectors where the components were randomly generated in the range $(-10^{30}, 10^{30})$. The results of this routine were checked against results obtained using 256 hex-digit arithmetic through the multiple precision arithmetic package written by J. R. Ehrman of SLAC. Of the thousand cases, 467 were in exact agreement, 537 had an erroneous last bit and 3 had an erroneous penultimate bit." [4, p. 3].

If the last sentence in Jordan's statement were changed to read "... 467 were in exact agreement, 537 had an erroneous last (hexadecimal) digit and 3 had an erroneous penultimate digit.", then Jordan's results are consistent with those of Michael Saunders at Stanford University. Saunders performed several experiments on an IBM 360 using $\beta = 16$, $t = 14$, $\eta = 43$, $\ell = 6$ and $q = 2$. He found examples where the 13-th hexadecimal digit of the result was in error, but none with errors in the 12-th digit. Errors of this size are consistent with Theorem 2.

V. Additional Modifications to the Algorithm

If one desires the final floating-point result to be correct in all digits, the following procedure can be used immediately after calculating S_0 :

1. Form $a_1, a_2, \ldots, a_q (q \geq 1)$, where $a_1 + a_2 + \ldots + a_q = S_0$ and each a_i has the property that the last ℓ digits of its mantissa are 0 .
2. Add $-a_1, -a_2, \ldots, -a_q$ to the accumulators.
3. Sum the accumulators in decreasing order. Call the result A .
4. $S_0 + \Delta$ is the full precision result.

In problems where N may get arbitrarily large (e.g., numerical integration), all is not lost. One merely increments an integer every time a term is added to the accumulators and when the integer is equal to $\beta^{\ell-\nu+1}-m(\eta+1)$, the following procedure is executed:

1. reset the integer to zero.

2.
```
for i := 0 step 1 until $\eta$ do
begin
    a := $\alpha_i$; $\alpha_i$ := 0;
    addtoaccumulators(a)
end
```
   where the procedure addtoaccumulators forms the $a_{i1}, \dots, a_{iq}$ variables, adds q to the integer and adds the a_{ij} $(j = 1,\dots,q)$ to the accumulators.

3. Resume the original algorithm.

VI. Conclusion

The generality of the preceding discussion tends to obscure the simplicity of the algorithm. For this reason, a simple illustrative example of this technique programmed in Fortran for the IBM 360 is included:

```
      REAL FUNCTION SUM(X,N)
      EQUIVALENCE (IEQ,W)
      DIMENSION X(1)
      REAL*8 R(43),S8,DBLE
      DO 10 I=1,43
10    R(I)=0.OD0
      DO 20 I=1,N
      W=ABS(X(I))
      IEXP=IEQ/50331648 + 1
C 50331648 IS 3*(2**24) WHICH SHIFTS RIGHT 24 BITS AND
C DIVIDES BY 3. ABS GETS RID OF THE SIGN BIT.
20    R(IEXP)=R(IEXP) + DBLE(X(I))
      S8=0 .OD0
      DO 30 I=1,43
30    S8=S8 + R(44-I)
      SUM=SNGL(S8)
      RETURN
      END
```

This subroutine finds the sum of a vector of short-precision ($t = 6$) numbers. Since the 360 used has long-precision ($t = 14$) floating-point hardware, it is convenient to use 14 digit accumulators and append 8 zero digits at the end of each x_i . Thus, $q = 1$ and $\ell = 8$. The value $\eta = 43$ was chosen to give $v = 3$. Thus, since $\beta = 16$, the short-precision result is guaranteed to have full-precision (chopped) accuracy provided $N \leq 16^6 = 16,777,216$.

This example is typical in that q is usually small (1 or 2), ℓ is usually chosen for convenience, and η is usually chosen so the accumulators can be quickly indexed and so ν is sufficiently small.

The algorithm is currently used in several innerproduct routines (see Malcolm [3] for descriptions of these routines, including running times) . Since efficiency is satisfactory, it may well be feasible to implement this technique through a microprogram so that the programmer can specify by a certain operation code that a summation is to be performed with a set of these accumulators rather than with a single accumulator.

VII. Acknowledgment

The author is indebted to Professor George E. Forsythe for his helpful comments and criticisms of the manuscript, The author would also like to thank Michael Saunders for several enlightening discussions during the evolution of this algorithm and for his wealth of numerical counterexamples.

BIBLIOGRAPHY

[1] Wilkinson, J. H., Rounding Errors in Algebraic Processes, Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.

[2] Wolfe, J. M., "Reducing Truncation Errors by Programming," CACM, Vol. 7, No. 6, June 1964, 355-356.

[3] Malcolm, M. A., "A Description and Comparison of Subroutines for Computing Euclidean Inner Products of Vectors," Technical Report (to appear), Computer Science Department, Stanford University, 1970.

[4] Jordan, D. F., "ANL F154S - DOTP, Extra-Precision Accumulating Inner Product,'? Argonne National Laboratory, Applied Mathematics Division, System/360 Library Subroutine, Argonne, Illinois, November, 1967.

[5] Møller, Ole, "Quasi Double-precision in Floating Point Addition," BIT, 5 (1965), 37-50.

[6] Knuth, D. E., "Seminumerical Algorithms," The Art of Computer Programming, Vol. 2, Reading, Mass.: Addison-Wesley Publishing CO.' 1969.