A Comparison Of Methods For Accurate Summation †

John Michael McNamee
York University, Toronto, Ontario, Canada
mcnamee@yorku.ca

Abstract

The summation of large sets of numbers is prone to serious rounding errors. Several methods of controlling these errors are compared, with respect to both speed and accuracy. It is found that the method of "Cascading Accumulators" is the fastest of several accurate methods. The Double Compensation method (in both single and double precision versions) is also perfectly accurate in all the tests performed. Although slower than the Cascade method, it is recommended when double precision accuracy is required. C programs that implement both these methods are available in the BULLETIN online repository.

1 Introduction

It is well known that, unless precautions are taken, the summation of large sets of numbers can be very inaccurate due to the accumulation of rounding errors. In fact Wilkinson [11] shows that the sum of n numbers all approximately equal to x may contain a maximum error of order $n^2 10^{-t} x$, where t is the number of places in the mantissa of the machine. However Mikov [7] shows that the probable error is of order $n^{3/2} 10^{-t} x$.

Several authors (e.g. [4, 5, 6]) have described methods for reducing or eliminating the rounding error, and Higham [2] compared a number of such methods with respect to accuracy. In the present work we extend Higham's work by also considering timings, and by presenting C programs in some cases. Also we consider the method of "Cascading Accumulators", described by Malcolm [6]) but not considered by Higham (probably because it is machine-dependent). We repeat and confirm Higham's comparisons, including all methods compared or even briefly mentioned in [2, 3], and we use some additional test cases. As mentioned we present C programs implementing the methods which are most successful.

2 The Methods Considered

1. The given order.
2. Increasing order of magnitude. The C built-in function `qsort` is used here and in methods 3, 10, 11, 12, and 14 below.
3. Decreasing order of magnitude.
4. Kahan's method [4]: here $u = S_k$ = sum of first k numbers, $v = x_{k+1}$ = $(k+1)$th number. We form $w = u + v;\ r = (w - u) - v$; subtract r from next number before adding.
5. Double precision (note that most of the methods use single precision).
6. Brent's multiple precision package [1]. This is used to calibrate the tests.

†This article was formally reviewed following the procedures described in THIS BULLETIN, **32**(2), issue 124, 1998, pp 5–6.

7. The method of Cascading Accumulators [6]. Here a series of double precision accumulators are set up, and each x_i added to the accumulator which corresponds to the exponent of x_i (in our program each accumulator corresponds to 4 consecutive exponents). At the end the accumulators are added in descending order. Finally a correction term is added, see [6], section V).

8. The Psum method. Here the x_i are ordered to minimize in turn $|x_1|, |\hat{S}_2|, |\hat{S}_3|, ..., |\hat{S}_{n-1}|$ where $\hat{S}_k$ = computed sum of first k numbers.

9. Summation by pairs (see [5]). Here we form $S_1 = x_1 + x_2,\ S_2 = x_3 + x_4,\ ...\ S_{\frac{n}{2}} = x_{n-1} + x_n$; then we sum the S_i in pairs, and so on.

10. Double Compensation [3, 9]. Here we first sort the x_i in descending order of magnitude. Then we form in turn:
    $s_1 = x_1;\ \ c_1 = 0;$
    For $i = 2$ to n:
    $y_k = c_{k-1} + x_k$
    $u_k = x_k - (y_k - c_{k-1});$
    $t_k = y_k + s_{k-1};$
    $v_k = y_k - (t_k - s_{k-1});$
    $z_k = u_k + v_k;$
    $s_k = t_k + z_k;$
    $c_k = z_k - (s_k - t_k);$
    End Loop

11. Insertion [10]. Here the x_i are first sorted by order of increasing magnitude, then $x_1 + x_2$ is formed and inserted into the list $x_3, ..., x_n$ so as to maintain the increasing order. The process is repeated until there is only one number (S_n) remaining.

12. We separate the positive and negative numbers into two lists, sort each list in increasing order of magnitude, sum each list, and add the results.

13. "Pichat's method" [8]. This is similar to Kahan's method, except that after each addition the estimated error is stored instead of being subtracted from the next term. At the end the sum of the errors is subtracted from the sum of the numbers. The process is repeated, forming the "errors of the errors", and so on, until convergence is reached.

14. Kahan's method is applied after sorting the numbers into decreasing order of magnitude.

The objective of all the methods is to add a set of single-precision numbers, obtaining a single-precision result of full accuracy, or as accurate as possible. Note that on some machines, and in some languages, quadruple precision may be available. In such cases the data could be given to double precision and one could seek a result correct to double precision. In particular, methods (5) and (7) would use quadruple precision. This has not been attempted in the present work. On the other hand, a double precision version of method (10) gives results correct to double precision without using quadruple precision (see end of Sec. 4). Note also that on some machines double precision is actually faster than single precision.

3 The Test Data

We considered 7 sets of data, several of which were modelled on Higham's test data. They are as follows:

1. x_i is the ith term in the Taylor series expansion for $e^{-2\pi}$, for 64 terms. This gives an alternating series of different sized terms, which is hard to sum accurately.

Type of data	Result
$e^{-2\pi}$	1.87052973e-3
Heavy Canc'n	2.e-18
Equal(1,2)	6143.5000
Normal(0,1)	-4.95493746
$\sum 1/i^2$	1.6446899
Random $10^{-35} \leq \ 10^{+35}$	-1.8820753e35
1 large, many small	1.0000001

Table 1: Results for tests using multiple precision

2. $x_1 = x_2 = ... = x_{2047} = 1.0,\ x_{2048} = x_{2049} = 1.0e-18,\ x_{2050} = x_{2051} = ... = x_{4096} = -1.0$. We refer to this data set as 'Heavy Cancellation' (always a dangerous feature).

3. The x_i are equally spaced on [1,2] for n=4096 (a relatively easy set to treat).

4. The x_i are random numbers from the Normal (0,1) distribution, for n=4096. This is harder than set 3 because the numbers vary in sign.

5. $x_i\ =\ 1/i^2$, for n=4096. This was used to be consistent with Higham's work.

6. The $x_i\ =\ \pm 10^{p_i}$ where the p_i are chosen from the Normal(0,35) distribution but with values greater than 35 in magnitude replaced by +35 or -35. As with set 2), this involves heavy cancellation in a random manner.

7. $x_1 = 1.0,\ x_2 = x_3 = ... = x_{10^9+1} = 1.0e-16$. We refer to this case as 'One large, many small'. It was used as it gives an error even in Double Precision.

In the 'One large, many small' case, only methods (1), (4), (5), and (7) of Sec. 2 were tested because the others all required the numbers to be stored in an array, but it was not possible to fit an array of 10^9 numbers on the machine used in these tests. Also the Multiple Precision test was not run as it would have taken far too much time.

The correct values of the sums, calculated by the Multiple Precision Package of Brent [1] (or by hand in case 7), are shown in Table 1.

4 The Results Of The Tests

The various methods and data were tested by running C programs on an IBM RS6000/590. The summations were usually performed 100 times each (or 10,000 times for the $e^{-2\pi}$ case with the results normalized to 100 times). Three separate runs were made to see if the times were consistent (which they were, reasonably). The results are shown in Tables 2 and 3. For each method and data set we show four numbers. They are, in order, the relative error, the mean time in seconds over the three runs, the standard deviation of the three times, and the slowdown relative to 'given order' (single precision). It is seen that the standard deviations are usually about 3% of the mean times, except where the times are very small and thus difficult to measure.

In terms of accuracy we see that the Cascading Accumulator, Pichat, Double Compensation, and Kahan Decreasing methods *always* give 0 error; the Increasing Magnitude method gives 0 error for the $\sum(1/i^2)$ case; Decreasing Magnitude for the $e^{-2\pi}$ and Heavy Cancellation cases; Kahan and Pairs for the Equal[1,2] and $\sum(1/i^2)$ cases; and Psum for several cases. Moreover, Double Precision gives 0 error for all except the Heavy Cancellation and 'One large, many small' cases. It may be thought that this last case is rather artificial, but it may be typical of the numbers found in integration of a difficult function, for example.

METHOD	TYPE OF DATA						
	$e^{-2\pi}$	Heavy Canc'n	equal [1,2]	normal (0,1)	$\sum(1/i^2)$	random $10^{-35} < 10^{35}$	One large, many small
Given	.0013	1.0	.00002	.0000013	.00002	.000001	.0000001
Order	.0004	.03	.027	.027	.033	.033	197
	.00005	.008	.005	.005	.005	.005	–
	–	–	–	–	–	–	–
Increasing	.0007	1.0	.00002	.000004	0	.0000002	–
Magnitude	.017	.35	2.19	2.23	2.22	1.18	–
	.00042	.012	.025	.03	.025	.05	–
	42	12	81	82	67	36	–
Decreasing	0	0	.00003	.000005	.00002	.0000005	–
Magnitude	.016	.36	2.21	2.24	2.20	1.19	–
	.005	.009	.021	.017	.05	.026	–
	40	12	82	83	67	36	–
Kahan	.0013	1.0	0	.0000001	0	.0000002	.0000001
	.0018	.12	.11	.11	.11	.11	270
	.0005	.008	.005	.008	.005	.005	–
	4.5	4	4	4	3.3	3.3	1.3
Double	0	1.0	0	0	0	0	.0000001
Precision	.00027	.01	.02	.02	.017	.02	198
	.00005	.008	0	0	.005	0	–
	.68	.3	.74	.74	.5	.6	1
Multiple	–	–	–	–	–	–	–
Precision	1.01	40	40	29	45	73	–
	0	0	0	0	0	0	–
	2525	1333	1481	1074	1364	2212	–
Cascading	0	0	0	0	0	0	0
Accumu-	.0020	.083	.087	.087	.10	.087	333
lators	.00008	.009	.005	.005	.008	.005	–
	5	2.8	3.2	3.2	3	2.6	1.7
Psum	.0007	1.0	.00002	0	0	0	–
	.0485	2.15	2.30	5.79	2.35	4.14	–
	.00043	.012	.033	.078	.05	.037	–
	121	72	85	214	71	125	–
Pairs	.002	1.0	0	.0000008	0	.0000002	–
	.0004	.02	.02	.02	.02	.02	–
	0	0	0	0	0	0	–
	1	.6	.74	.74	.6	.6	–
Double	0	0	0	0	0	0	–
Compensation	.019	.54	2.42	2.40	2.39	1.37	–
	.005	.021	.052	.022	.03	.005	–
	48	18	90	88	72	42	–

Table 2: Relative errors and timings for different methods and data (part I). Each box contains relative error, mean time, standard deviation, and slowdown.

METHOD	TYPE OF DATA						
	$e^{-2\pi}$	Heavy Canc'n	equal [1,2]	normal (0,1)	$\sum(1/i^2)$	random $10^{-35} < 10^{35}$	One large, many small
Insertion	.003	1	0	.000007	0	.000001	–
	.031	26.4	27.5	29.6	28.6	27.2	–
	0	.25	.082	.189	.094	.21	–
	78	880	1018	1096	837	824	–
Positive+	.005	1	.00002	.0002	0	.0000004	–
Negative	.017	.42	2.26	3.58	2.32	1.98	–
	0	.014	.012	.021	.029	.025	–
	42	14	84	132	70	60	–
Pichat	0	0	0	0	0	0	–
	.0055	.20	.19	.28	.35	.32	–
	.00008	.009	.0094	.008	.012	.005	–
	14	6.6	7	10	13	9.7	–
Kahan	0	0	0	0	0	0	–
Decreasing	.017	.45	2.28	2.3	2.3	1.24	–
	0	.012	.0082	.005	.03	.012	–
	42	15	84	85	70	38	–

Table 3: Relative errors and timings for different methods and data (part II). Each box contains relative error, mean time, standard deviation, and slowdown.

Note that Higham used Double Precision to calibrate his tests, but the present results show that this is not entirely reliable. We use multiple precision for this purpose. Many of the results agree with those of Higham[2].

As the Cascading Accumulator is the fastest of the four methods which always give 0 error, and is the only one of them which does not need to store the numbers in an array, we conclude that the Cascading Accumulator method is the best all-round one (when data given to single precision).

Also, for moderate-sized (i.e. $< 5 \times 10^8$) sets of equal- signed numbers (given to single precision) Double Precision calculation may be equally accurate and somewhat faster than this method. This is because 5×10^8 relative errors, each of 10^{-16}, contribute at most a total relative error of $.5 \times 10^{-7}$ to the sum, provided there is no cancellation.

However, if the data is given to double precision, and the result is required to that accuracy, a double precision version of the Double Compensation method would be recommended, as (unlike Cascading Accumulators) it does not require quadruple precision. It has been theoretically proven to have perfect accuracy [9], and the tests of Sec. 2 in double precision all yield perfect double precision accuracy. Pichat's method is considerably faster than Double Compensation, but gives only 12 figure accuracy in the first test of Sec. 2, for double precision data. Kahan's Decreasing method is also somewhat faster than Double Compensation, and accurate in all the tests performed, but this author does not know of any theoretical proof of its universal accuracy. For that reason we give preference to Double Compensation.

5 Conclusions

It has been found that the Cascading Accumulator method is the best overall method for summing sets of numbers of mixed sign, or large sets of numbers of equal sign. That is, it is the fastest of several accurate methods (in fact it is perfectly accurate in the absence of overflow, according to Malcolm [6]). This method is somewhat machine-dependent, but C programs are presented in the appendix for machines using IEEE Standard Floating Point format.

In cases where double precision accuracy is possible and needed, the Double Compensation method is preferable. A C program is also given for the latter method in double precision.

References

[1] R.P. BRENT (1978), *Algorithm 524: MP, a Fortran Multiple-Precision Package*, ACM Trans. Math. Software, 4, pp. 71-81.

[2] N.J. HIGHAM (1993), *The accuracy of floating point summation*, SIAM J. Sci. Comput., 14, pp. 783-799.

[3] N.J. HIGHAM (1996), *Accuracy and Stablility of Numerical Algorithms*, Soc. Ind. Appl. Math., pp. 96-97

[4] W. KAHAN (1965), *Further remarks on reducing truncation errors*, Comm. ACM, 8, p. 40.

[5] P. LINZ (1970), *Accurate floating-point summation*, Comm. Ass. Comp. Mach., 13, pp. 361-362.

[6] M.A. MALCOLM (1971), *On Accurate floating-point summation*, Comm. Ass. Comp. Mach., 14, pp. 731-736.

[7] A.I. MIKOV (1996), *Largescale addition of machine real numbers: accuracy estimates*, Theor. Comput. Sci., 162, pp. 151-170.

[8] M. PICHAT (1972) *Correction d'une somme en arithmetique a virgule flottante*, Numer. Math., 19, pp. 400-406

[9] D.M. PRIEST (1992), *On Properties of Floating-Point Arithmetics: Numerical Stability and the Cost of Accurate Computations*, Ph. D. Thesis, Mathematics Dept., University of California, Berkeley, CA. URL: `ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z`

[10] T.G. ROBERTAZZI and S.C. SCHWARZ (1988), *Best "ordering" for floating- point addition*, ACM Trans. Math. Software, 14, pp. 101-110

[11] J.H. WILKINSON (1963), *Rounding Errors in Algebraic Processes*, H.M.S.O., London

Appendix: The CASCADING ACCUMULATORS method and programs.

We here briefly describe the method of Cascading Accumulators and its implementation in the accompanying C programs. This description is based on [6]. See that paper for more details and proofs.

We assume a normalized floating-point number system with base β and a t-digit mantissa (in single precision). The exponent e satisfies

$$-m \ \leq e \ \leq M$$

If x is a floating-point number, we define

$$lev(x) \ = \ e + m$$

i.e. it is the biased exponent.

Let us set up η+1 double-precision accumulators $\alpha_0, \alpha_1, ..., \alpha_\eta$. let l be the number of bits by which the double-precision mantissa exceeds the single-precision mantissa.

Let

$$\nu = \left\lceil \frac{(M + m + 1)}{(\eta + 1)} \right\rceil \ .$$

The algorithm to add $\sum_{i=1}^{N} x_i$ proceeds as follows:

Step 1. Set each accumulator α_j to zero.

Step 2. Convert each x_i to double precision in, say, a_i.

Step 3. Add a_i to the kth accumulator, where k is determined by $\nu k \ \leq \ lev(a_i) \ \leq \ \nu k + \nu - 1$

Step 4. The accumulators are summed in *decreasing* order (i.e. $\eta, \eta - 1, ..., 0$) using a double-precision variable S_0.

Step 5. Subtract S_0 from the appropriate α_k, determined by lev(S_0).

Step 6. Sum the accumulators (including the modified one) in decreasing order, to give Δ.
Step 7. $S_0 + \Delta$ is the result.

Malcolm shows that this is exact to single precision, provided $N \leq \beta^{l-\nu+1}$. In the case where N exceeds the above value, Malcolm describes an additional procedure which will still ensure exact accuracy: Increment an integer each time a number is added to one of the accumulators, and when the integer equals $\beta^{l-\nu+1} - (\eta + 1) \equiv$ LIMIT do the following:
Step 1. Reset the integer to 0.
Step 2. Perform:

For i=0 step 1 until η do
begin
$a = \alpha_i;\ \alpha_i = 0;$
add a to the appropriate accumulator
end.

Step 3. Resume the original algorithm.

For the IEEE Standard Floating-Point format β = 2, t = 24, m = M = 127, l = 29. In our program we used η = 63 so that ν = 4, and LIMIT = 67,108,800.

We provide two versions of the program. The first is intended for use where N is relatively small, i.e. small enough so that
(1) all the numbers fit into an array on the machine being used, **and**
(2) N $\leq$ LIMIT.

The second version should be used if either of these conditions is not satisfied. It expects the numbers to be calculated by a function genx(i). It may also be used even for smaller N if it is more convenient to generate the numbers than to store them in an array (e.g. in numerical integration).

In both cases the summation is performed by a function sumcasc(). In the short version this has two parameters, float x[ARR_SIZE] and int n. In the long version it has only one parameter, int n. Here n is the actual number of numbers to be summed, while where applicable x is the array of numbers and ARR_SIZE is the maximum size of the array. This may be altered by the user.

The Cascading Accumulators programs and the Double Compensation program are available at the BULLETIN file repository `http://www.acm.org/sigs/sigsam/bulletin/repository/issue147/`