ALGORITHM. ACCURATE FLOATING POINT SUMMATION

(Submitted to the Communications of the ACM)
ALGORITHM.
ACCURATE FLOATING POINT SUMMATION [A1]

G.A. Erskine,
Data Handling Division,
European Organization for Nuclear Research (CERN),
1211 Geneva 23, Switzerland.

Key Words and Phrases: summation, computer arithmetic, rounding error

CR Categories: 5.11
Language: Fortran

Description

Introduction. Successive calls to the subroutine TRESUM will cause the sum of a sequence of numbers $x_j$ to be evaluated according to the binary tree summation pattern proposed by P. Linz [1] as a means of reducing the accumulation of rounding errors. The subroutine may be used in circumstances in which it would be impossible or inconvenient to store the whole of the sequence before beginning the Linz summation. Possible applications include Romberg quadrature and the summation of slowly convergent series. A similar summation structure is an inherent part of the fast Fourier transform, and contributes to its good round-off properties.

From the user's point of view, the subroutine behaves as if it operated on a single internal accumulator by means of three types of subroutine call:

Type 1 CALL TRESUM(X,1) Set zero in the accumulator (X is ignored).
Type 2 CALL TRESUM(X,2) Add X to the accumulator.
Type 3 CALL TRESUM(S,3) Copy the accumulator into S.

An initial call of Type 1 would normally be followed by a sequence of calls of Type 2 and a final call of Type 3. Intermediate calls of Type 3 do not disturb the summation.
Internally, the subroutine makes use of a sequence of auxiliary accumulators \( A_1, A_2, \ldots \) in which intermediate results are built up in pairs as shown in Figure 1. It can be seen from the figure that the number of floating-point additions which are triggered by the arrival of \( x_j \) is equal to the number of zero bits at the right-hand end of the subscript \( j \) written as a binary integer. The index \( k \) which identifies the current auxiliary accumulator \( A_k \) is set initially to zero, is increased by 1 each time a new term is made available, and is decreased by 1 each time a floating-point addition is performed.

Unless the number of terms \( x_j \) is an exact power of 2, the final sum \( x_1 + x_2 + \ldots + x_n \) will not be immediately available in any of the auxiliary accumulators. To obtain this sum it is necessary to add together the contents of \( A_k, A_{k-1}, \ldots, A_1 \), where \( A_k \) is the auxiliary accumulator which was last referenced. For example, if we consider the situation immediately after \( x_{12} \) has been dealt with, we see that the sum \( x_1 + x_2 + \ldots + x_{12} \) is obtained by adding \( A_2 \) to \( A_1 \). The number of auxiliary accumulators in TRESUM has been set to 25, permitting the summation of up to \( 2^{25} = 3.3 \times 10^7 \) terms.

**Error Analysis.** We assume that the absolute error in the floating-point sum of two numbers \( \alpha \) and \( \beta \) is bounded by \( 2\eta (|\alpha| + |\beta|) \) where \( \eta \) is a machine constant. (In a binary computer with a q-bit mantissa we would have \( \eta = 2^{-q} \).) We define the **forwards sum** of the \( n \) numbers \( x_j \) to be \( f_n \), where \( f_0 = 0 \) and \( f_r = f_{r-1} + x_r \) \((r = 1, 2, \ldots, n)\), and the **backwards sum** of the same sequence to be \( b_1 \), where \( b_{n+1} = 0 \) and \( b_r = b_{r+1} + x_r \) \((r = n, n-1, \ldots, 2, 1)\). By arguments similar to those of \([1]\) it may be shown that the absolute error of the computed sum is bounded by the following expressions:

\[
\begin{align*}
\text{Forwards sum} & \quad 2\eta \sum_{r=1}^{n} (n - r + 1)|x_r| , \\
\text{Backwards sum} & \quad 2\eta \sum_{r=1}^{n} r|x_r| , \\
\text{TRESUM} & \quad 2\eta \left[1 + \log_2 n\right] \sum_{r=1}^{n} |x_r| ,
\end{align*}
\]

where the square brackets indicate that the integral part is to be taken.
These bounds are not necessarily over-pessimistic. For example, consider the sum $S = 9.787...$ of the first $10^4$ terms of the series $1 + (1/2) + (1/3) + (1/4) + ...$ as computed on a CDC 6600 computer ($\eta = 2^{-48}$) using (i) chopped arithmetic (truncation of intermediate results to 48 bits without rounding), and (ii) 48-bit rounded arithmetic. The error of the computed sum, expressed in units of the least significant bit of the result, is found to be as follows:

<table>
<thead>
<tr>
<th></th>
<th>Chopped arithmetic</th>
<th>Rounded arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forwards sum</td>
<td>4418</td>
<td>26</td>
</tr>
<tr>
<td>Backwards sum</td>
<td>434</td>
<td>3</td>
</tr>
<tr>
<td>TRESUM</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

In this example the forwards, backwards, and TRESUM errors for chopped arithmetic are respectively $40\%$, $35\%$ and $35\%$ of the error bounds given above.

References

Algorithm

SUBROUTINE TRESUM(X,MODE)
REAL A(25),TEMPI,X
INTEGER J,JHALF,JSHIFT,K,L,M,MODE
DATA J,K,0,0/
C
C *****************************************************
C SUMMATION OF REAL NUMBERS USING A BINARY TREE SUMMATION
C PATTERN IN ORDER TO REDUCE ACCUMULATION OF RUNDOWN ERROR,
C AS SUGGESTED BY P. LINZ, COMMUNICATIONS OF THE ACM, VOLUME
C THIS SUBROUTINE BEHAVES AS IF IT HAD A SINGLE ACCUMULATOR
C IN WHICH THE SUM IS BUILT UP. THE FOLLOWING CALLS ARE USED
C 1. CALL TRESUM(X,1) SET ZERO IN ACCUMULATOR
   (X IS IGNORED).
C 2. CALL TRESUM(X,2) ADD X TO ACCUMULATOR.
C 3. CALL TRESUM(X,3) COPY THE ACCUMULATOR INTO X.
C AN INITIAL CALL OF TYPE 1 IS FOLLOWED BY A SEQUENCE OF
C CALLS OF TYPE 2 AND A FINAL CALL OF TYPE 3. INTERMEDIATE
C TYPE 3 CALLS DO NOT DISTURB THE SUMMATION.
C INTERNALLY, THE SUBROUTINE USES A(1),A(2), ..., A(25) AS
C AUXILIARY ACCUMULATORS, PERMITTING THE SUMMATION OF UP TO
C 2^25 (APPROXIMATELY 33 MILLION) TERMS.
C INTERNAL VARIABLES J AND K ARE USED AS FOLLOWS
C J ... COUNTER FOR NUMBER OF TYPE 2 CALLS.
C K ... A(K) IS AUXILIARY ACCUMULATOR LAST REFERENCED.
C *****************************************************
C SELECT ENTRY POINT ACCORDING AS MODE=1,2,3.
C IF(MODE=2) 10,20,30
C ENTRY MODE=1. RESET J AND K.
C 10 J=0
K=0
RETURN
C ENTRY MODE=2. PERFORM AS MANY ADDITIONS AS THERE ARE ZEROS
C AT THE TAIL OF J.
C 20 J=J+1
K=K+1
TEMP=X
JSHIFT=J
C 21 JHALF=JSHIFT/2
IF(JHALF+JHALF*NE,JSHIFT) GO TO 22
K=K-1
TEMP=TEMP+A(K)
JSHIFT=JHALF
GO TO 21
C 22 A(K)=TEMP
RETURN
C ENTRY MODE=3. SET X=A(K)+A(K-1)+ ... +A(1).
C 30 TEMP=0.
IF(K.EQ.0) GO TO 32
C DO 31 L=1,K
M=K-L+1
TEMP=TEMP+A(M)
31 CONTINUE
C 32 X=TEMP
RETURN
END
Fig. 1