SUMMATION

How should we compute a sum

$$S = a_1 + a_2 + \cdots + a_n$$

with a sequence of machine numbers $\{a_1, ..., a_n\}$. Should we add from largest to small, should we add from smallest to largest, or should we just add the numbers based on their original given order? In other words, does it matter how we calculate the sum?

Recall the relationship between a number x and its machine approximation $fl(x)$:

$$fl(x) = (1 + \varepsilon)\, x$$

For bounds on ε for a binary floating point representation with N binary digits in the mantissa, we have

$$\begin{array}{lll} -2^{-N} & \leq \varepsilon \leq 2^{-N}, & \text{rounding} \\ -2^{-N+1} & \leq \varepsilon \leq 0, & \text{chopping} \end{array}$$

We use these results as tools for analyzing the error in computing the sum S.

We create the sum S by a sequence simple additions. Define

$$\begin{array}{rcll} S_2 & = & fl\,(a_1 + a_2) & = & (1+\varepsilon_2)\,(a_1+a_2) \\ S_3 & = & fl(S_2 + a_3) & = & (1+\varepsilon_3)\,(S_2+a_3) \\ S_4 & = & fl(S_3 + a_4) & = & (1+\varepsilon_4)\,(S_3+a_4) \\ & \vdots & & & \\ S_n & = & fl(S_{n-1} + a_n) & = & (1+\varepsilon_n)\,(S_{n-1}+a_n) \end{array}$$

This says each simple addition is performed exactly, following which it is rounded or chopped back to the precision of the machine. All of the numbers ε_j satisfy the inequalities given earlier.

We now combine the above to obtain a formula that is simpler to work with. In particular, we find that

$$\begin{array}{rcl} S - S_n & \approx & -a_1\,(\varepsilon_2 + \cdots + \varepsilon_n) \\ & & -a_2\,(\varepsilon_2 + \cdots + \varepsilon_n) \\ & & -a_3\,(\varepsilon_3 + \cdots + \varepsilon_n) \\ & & -a_3\,(\varepsilon_4 + \cdots + \varepsilon_n) \\ & & \vdots \\ & & -a_n\varepsilon_n \end{array}$$

In obtaining this, we have neglected all terms containing products $\varepsilon_i \varepsilon_j$, as these are generally much smaller than the remaining terms.

For example,

$$\begin{aligned} S_2 &= (a_1 + a_2) + \varepsilon_2 (a_1 + a_2) \\ S_3 &= (1 + \varepsilon_3) [a_3 + (1 + \varepsilon_2)(a_1 + a_2)] \\ &= (a_1 + a_2 + a_3) + (a_1 + a_2)(\varepsilon_2 + \varepsilon_3) \\ &\qquad + a_3\varepsilon_3 + \varepsilon_2\varepsilon_3 (a_1 + a_2) \\ &\doteq (a_1 + a_2 + a_3) + (a_1 + a_2)(\varepsilon_2 + \varepsilon_3) + a_3\varepsilon_3 \end{aligned}$$

Continue in this manner to get

$$\begin{aligned} S_n \doteq\ & (a_1 + a_2 + \cdots + a_n) \\ & + (a_1 + a_2)(\varepsilon_2 + \varepsilon_3 + \cdots + \varepsilon_n) \\ & + a_3 (\varepsilon_3 + \cdots + \varepsilon_n) \\ & + a_4 (\varepsilon_4 + \cdots + \varepsilon_n) \\ & + \cdots + a_n\varepsilon_n \end{aligned}$$

Using this yields the formula given earlier for $S - S_n$.

Consider now the formula

$$\begin{aligned} S - S_n &= -a_1(\varepsilon_2 + \cdots + \varepsilon_n) \\ &\quad -a_2(\varepsilon_2 + \cdots + \varepsilon_n) \\ &\quad -a_3(\varepsilon_3 + \cdots + \varepsilon_n) \\ &\quad -a_4(\varepsilon_4 + \cdots + \varepsilon_n) \\ &\quad \vdots \\ &\quad -a_n\varepsilon_n \end{aligned}$$

and what it suggests as a method for adding the numbers $a_1, ..., a_n$.

Since a_1 and a_2 have the largest number of quantities ε_j multiplied times them, it makes sense to have a_1 and a_2 be the smallest numbers in magnitude. We can continue in this vein to motivate ordering the numbers to satisfy

$$|a_1| \leq |a_2| \leq \cdots \leq |a_n|$$

This is not an optimal method, but it generally is superior to using a random ordering or to adding from the largest term to the smallest.

EXAMPLE

An example is given in the text for

$$\sum_{n=1}^{n} a_j$$

with

$$a_j = fl\,(1/j)\,, \qquad j \geq 1$$

The numbers a_j are obtained by rounding to 4 significant decimal digits the numbers $1/j$. Then two different decimal arithmetics are used, along with adding from smallest to largest (SL), and from largest to smallest (LS). The results are given in Section 2.4.

For $n = 1000$, we have the following results:

Chopped arithmetic, smallest to largest: Error=.202
Chopped arithmetic, largest to smallest: Error=.417
Rounded arithmetic, smallest to largest: Error=0
Rounded arithmetic, largest to smallest: Error=.037

Why is it so much larger with chopped arithmetic?

Using chopped 4-digit decimal arithmetic

n	True	SL	Error	LS	Error
10	2.929	2.928	0.001	2.927	0.002
25	3.816	3.813	0.003	3.806	0.010
50	4.499	4.491	0.008	4.479	0.020
100	5.187	5.170	0.017	5.142	0.045
200	5.878	5.841	0.037	5.786	0.092
500	6.793	6.692	0.101	6.569	0.224
1000	7.486	7.284	0.202	7.069	0.417

Using rounded 4-digit decimal arithmetic

n	True	SL	Error	LS	Error
10	2.929	2.929	0	2.929	0
25	3.816	3.816	0	3.817	-0.001
50	4.499	4.500	-0.001	4.498	0.001
100	5.187	5.187	0	5.187	0
200	5.878	5.878	0	5.876	0.002
500	6.793	6.794	-0.001	6.783	0.010
1000	7.486	7.486	0	7.449	0.037

Recall the formula

$$\begin{aligned} S_n \doteq\ & (a_1 + a_2 + \cdots + a_n) \\ & + (a_1 + a_2)(\varepsilon_2 + \varepsilon_3 + \cdots + \varepsilon_n) \\ & + a_3(\varepsilon_3 + \cdots + \varepsilon_n) \\ & + a_4(\varepsilon_4 + \cdots + \varepsilon_n) \\ & + \cdots + a_n\varepsilon_n \end{aligned}$$

and examine one of the quantities on the right, as an example of the general case. The first term contains the sum

$$\varepsilon_2 + \varepsilon_3 + \cdots + \varepsilon_n$$

Write this as

$$(n-1)\left(\frac{1}{n-1}\sum_{j=2}^{n}\varepsilon_j\right) \equiv (n-1)\varepsilon^*$$

In the case of rounding, the mean ε^* is approximately zero, because the negative and positive values of ε_j are likely to cancel when large numbers of such ε_j are being summed; and thus $(n-1)\,\varepsilon^*$ is usually quite small. But for the case of chopping, ε^* is approximately -2^{-N}, and therefore,

$$(n-1)\,\varepsilon^* \approx -\,(n-1)\,2^{-N}$$

Thus $(n-1)\,\varepsilon^*$ will grow in a manner proportional to n. By more advanced arguments, it can be shown that for the case of rounded arithmetic, $(n-1)\,\varepsilon^*$ will grow in a manner proportional to sqrt(n), which grows much slower than n.

EXTENDED PRECISION ARITHMETIC

Sometimes a limited use of a higher precision arithmetic will greatly cut the amount of rounding error in a calculation. Consider as an important example the calculation of the 'dot product' or 'inner product'

$$S = \sum_{j=1}^{n} a_j b_j$$

with the numbers a_j, b_j all machine numbers.

Imagine calculating this in single precision arithmetic. Then there will be approximately n rounding errors from the multiplications and $n-1$ rounding errors from the additions. To cut this, imagine extending each of the numbers a_j, b_j to double precision by appending sufficient zeros to them. Then multiply and add them in double precision; and when completed, round the answer back to single precision. This replaces $2n-1$ single precision rounding errors with 1 such rounding error, a significant improvement.

What do you do if you are already in double precision? With the IEEE standard, you can use 'extended precision arithmetic', which gives 16 binary digits of extra accuracy. It must be accessed by using special routines, but it allows calculation of much more accurate inner products at a minimal increase in operation time.

In linear algebra problems, rounding errors in many-term summations, including inner products, are the principal source of error. We will discuss this further in the chapter on numerical linear algebra.

In *MATLAB*, all arithmetic is in double precision, and extended IEEE arithmetic is not available. Thus we illustrate these ideas with only a *Fortran* program.

```
      REAL FUNCTION SUMPRD(A,B,N)
C
C     THIS CALCULATES THE INNER PRODUCT
C
C                  I=N
C     SUMPRD = SUM A(I)*B(I)
C                  I=1
C
C     THE PRODUCTS AND SUMS ARE DONE IN DOUBLE
C     PRECISION, AND THE FINAL RESULT IS
C     CONVERTED BACK TO SINGLE PRECISION.
C
REAL A(*), B(*)
DOUBLE PRECISION DSUM
C
DSUM = 0.0DO
DO I=1,N
DSUM = DSUM + DBLE(A(I))*DBLE(B(I))
END DO
SUMPRD = DSUM
RETURN
END
```