Parabola Volume 27, Issue 1 (1991)

DO COMPUTERS AND CALCULATORS LIE?

By Bill McKee*

Introduction

In everyday life we tend to trust the numbers which come out of a computer or calculator. For example, I know that the additions and subtractions on my monthly bank statement will be done correctly. Of course, some of the numbers being added or subtracted may be wrong but that is due to human error in entering the data into the computer. However, there are some simple situations in which computers and calculators can give results which are clearly incorrect and, what's more, the results we get depend on the machine we are using and the way in which we do the calculations. In this article, I will use the word "machine" to refer to both computers and calculators.

An Example

Every drover and his dog know that if we take any number, add one to it, and then subtract one we end up with the number the dog first thought of. In symbols we have, for any x,

$$(1+x)-1 \equiv x.$$

I have two calculators. Let's see what they think of this. The table below shows the results of the calculation $(1+x)-1$ for various values of x on each of my two calculators.

x	Calculator 1	Calculator 2
$.12345 \times 10^{-4}$	$.12345 \times 10^{-4}$	$.12345 \times 10^{-4}$
$.12345 \times 10^{-5}$	$.1235 \times 10^{-5}$	$.12345 \times 10^{-5}$
$.12345 \times 10^{-6}$	$.123 \times 10^{-6}$	$.1234 \times 10^{-6}$
$.12345 \times 10^{-7}$	$.12 \times 10^{-7}$	$.123 \times 10^{-7}$
$.12345 \times 10^{-8}$	$.1 \times 10^{-8}$	$.12 \times 10^{-8}$
$.12345 \times 10^{-9}$	0	0

This is a little worrying. Let us re-arrange the calculation as $(x-1)+1$ and see what happens:

* Bill is an Applied Mathematician at the University of New South Wales

x	Calculator 1	Calculator 2
$.12345 \times 10^{-4}$	$.12345 \times 10^{-4}$	$.12345 \times 10^{-4}$
$.12345 \times 10^{-5}$	$.12345 \times 10^{-5}$	$.12345 \times 10^{-5}$
$.12345 \times 10^{-6}$	$.1234 \times 10^{-6}$	$.12345 \times 10^{-6}$
$.12345 \times 10^{-7}$	$.123 \times 10^{-7}$	$.1234 \times 10^{-7}$
$.12345 \times 10^{-8}$	$.12 \times 10^{-8}$	$.123 \times 10^{-8}$
$.12345 \times 10^{-9}$	$.1 \times 10^{-9}$	$.12 \times 10^{-9}$
$.12345 \times 10^{-10}$	0	0

This is even more worrying. Re-ordering the calculations is giving answers which are clearly not correct either, differ from the previous ones and, as before, vary from one machine to another. To try and sort this out, lets put these two calculations onto the computer at my work:

x	$(1+x)-1$	$(x-1)+1$
$.12345 \times 10^{0}$	$.1234500 \times 10^{0}$	$.1234500 \times 10^{0}$
$.12345 \times 10^{-1}$	$.1234496 \times 10^{-1}$	$.1234502 \times 10^{-1}$
$.12345 \times 10^{-2}$	$.1234531 \times 10^{-2}$	$.1234472 \times 10^{-2}$
$.12345 \times 10^{-3}$	$.1235008 \times 10^{-3}$	$.1234412 \times 10^{-3}$
$.12345 \times 10^{-4}$	$.1239777 \times 10^{-4}$	$.1233816 \times 10^{-4}$
$.12345 \times 10^{-5}$	$.1192093 \times 10^{-5}$	$.1251698 \times 10^{-5}$
$.12345 \times 10^{-6}$	$.1192093 \times 10^{-6}$	$.1192093 \times 10^{-6}$
$.12345 \times 10^{-7}$	0	0

This is even more confusing - we are getting still other results. You might like to experiment on machines available to you. It is the case that any machine will ultimately give 0 as the result of the calculation $(1+x)-1$ for sufficiently small values of x. Equally, any machine will ultimately give 0 as the result of the calculation $(x+1)-x$ when x is sufficiently large.

In order to gain some understanding of what is leading to the above results, we need to know how computers store and manipulate numbers. Before doing so, however, we should remark that the examples presented above are not just puzzling curiosities but things which should alert us to potential dangers in calculations. For example suppose we had, in the middle of some computer program, to calculate $r = x+y-z$ and then to divide some other number by this quantity. If x were small and y and z were both 1 or, more generally, if y and z were equal and very much larger than x, the machine evaluation of r could be subject to quite large relative errors leading to large errors in the result of the division. For very small values of x, the machine would give 0 as the value of r, leading to the program aborting when division by r was attempted.

Computer Representation of Numbers

We all normally use the decimal system to express numbers. This is because we have four fingers and one thumb on each hand. If we had evolved with only three fingers and one thumb, we would almost certainly use the octal system, i.e. use base 8 not 10.

Fortunately, most calculators use the decimal system so let's begin our discussion there. We all know that the decimal expression 12.345 represents the number $1 \times 10^1 + 2 \times 10^\circ + 3 \times 10^{-1} + 4 \times 10^{-2} + 5 \times 10^{-3}$.

Now, a calculator can only store a finite number of decimal places so that, for example, if I enter the number $x_1 = 0.112233445566778899$ into my calculator it gives me back 0.1122334455, i.e. all other digits are lost. Equally, entering $x_2 = 0.112233445599999$ also gives me back 0.1122334455. My machine cannot distinguish between x_1 and x_2 and so would give 0 as the result of the calculation $(x_1 - x_2)$ or even $10^{10}(x_1 - x_2)$.

What actually happens will vary from machine to machine so let us consider a "typical" decimal machine with a mantissa of length k digits. In such a machine, numbers are stored in the form

$$\pm 0.d_1 d_2 d_3 \ \ldots \ d_k \times 10^n$$

where the integers $d_1, \ d_2, \ \ldots \ d_k$ lie between 0 and 9 inclusive. This form represents the number

$$\pm(d_1 \times 10^{-1} + d_2 \times 10^{-2} + \ \ldots \ + d_k \times 10^{-k}) \times 10^n.$$

The normalised form is almost always used, which means that $d_1 \neq 0$ unless, of course, the whole number is zero. This means that 13.674 would be stored as 0.1367400×10^2 on a 7 decimal-digit machine ($k = 7$) rather than as 0.0136740×10^3. {Of course, the machine may print out the number as 1.367400×10^1 or in some other form, but that is an unimportant detail}.

Typically, $k = 10$ and $-99 \leq n \leq 99$. We can immediately see that, on such a machine, not all numbers can be represented exactly and that, disregarding the sign, there is a largest number ($0.1111111111 \times 10^{99}$) and a smallest non-zero number ($0.1000000000 \times 10^{-99}$) that can be represented exactly. If a particular number has too many decimal digits to be represented exactly machines generally do one of two things:

(a) chop, i.e., ignore all digits after the k-th

(b) round, i.e. add 5 to the $(k+1)$th digit then chop.

The difference between the actual number and its machine representation is usually called the round-off error (r.o.e.) irrespective of whether the machine rounds or chops.

Now lets consider what happens when we perform some arithmetic operation (addition, subtraction, multiplication or division) on two or more numbers. A standard example is 2/3. One of my calculators gives 0.6666666667 and so has rounded whereas the other calculator gives 0.6666666666 and so has chopped. Roundoff can be a particular problem if we are subtracting two numbers that are almost equal. The details will vary from machine to machine but we will consider the ideal case in which each individual operation on k-digit numbers is performed exactly and the result then rounded or chopped to k digits. We will illustrate by calculating $(1+x)-1$ on a decimal machine with a 10-digit mantissa when $x = 0.12345 \times 10^{-5}$. This x value would, of course, be held exactly by our machine. Now $1 + x = 0.10000012345 \times 10^1$. Our machine would round this to 0.1000001235×10^1 whereas a chopping machine would give 0.1000001234×10^1. Subtracting 1 would thus give $0.1235000000 \times 10^{-5}$ on a rounding machine and $0.1234000000 \times 10^{-5}$ on a chopping machine. What actually happens on any real machine may well vary from this "ideal" situation but all machines will give erroneous results if $(1 + x)$ contains more significant digits than the mantissa can accommodate.

This sort of thing can cause problems if we try to approximate derivatives numerically.

As you know, the derivative of $f(x)$ is defined by

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{1}$$

For very complicated functions which we do not know how to differentiate analytically, we would expect that using small values of h in the r.h.s. of (1) would give us good approximations to $f'(x)$ and that the approximations would get better and better as h got smaller and smaller. This is true in principle but the finite mantissa length on any real machine ultimately defeats this strategy. After a certain point the approximation begins to get worse as the machine fails to represent $(x+h)$ exactly and we eventually get 0 when the machine cannot distinguish $(x+h)$ from x. For $x \neq 0$, point is determined by the number of digits the machine can hold in its mantissa. You might like to experiment on your calculator with $f(x) = e^x$, $x = 1$ starting with $h = 0.1$ and successively reducing h by a factor of 10 each time.

In contrast to calculators, computers generally do not use the decimal system in their internal working. They usually represent numbers in binary form (base 2), octal (base 8) or hexadecimal (base 16). If we enter a number in decimal form into such a computer, the number is converted into binary (or octal or hexadecimal) form, all calculations are then done in this form and the results re-converted back into decimal on output so we can understand them. In addition to the round-off problem discussed above there may be additional errors introduced in the conversions between decimal and binary. For example, the number $\frac{1}{10}$ clearly has a decimal form of finite length whereas its binary expansion is infinitely long and so would be rounded or chopped on a computer, introducing a small error. The computer I used in the example quoted earlier has a mantissa 24 binary digits long, which corresponds approximately, but not exactly, to 7 decimal digits. This explains the quite different results obtained for $(1+x) - 1$ between the computer and both of my calculators.

We now realise that each time a machine performs an operation there is likely to be some roundoff error introduced. If a particular program requires a lot of calculations, roundoff error may build up significantly. Consider, for example $\sum_{n=1}^{10^5} 0.1$, i.e. add $\frac{1}{10}$ to itself 10^5 times. A decimal machine will do this exactly but a binary machine will not. For example my computer at work gives 9998.557 instead of 10^4. Neither machine would give the exact value of $\sum_{n=1}^{3000}\left(\frac{1}{3}\right)$, however, since the binary and decimal representations of $\frac{1}{3}$ are both infinitely long.

In some circumstances, roundoff error may build up catastrophically and completely ruin a calculation.

Strategies to reduce roundoff error

We now recognize that computers and calculators are finite devices and cannot hold an infinite number of decimal or binary digits. We cannot eliminate roundoff error. We must learn to live with it and be on the lookout for manifestations of roundoff error corrupting the results of our calculations. There are strategies for reducing the effects of roundoff error. Two of the most obvious are listed below.

Strategy 1

Since each operation $(+, -, \times, \div)$ is likely to introduce some roundoff error it is sensible to try to reduce the total number of such operations. A typical example is in the evaluation of polynomials. Consider a cubic

$$p(x) = a + bx + cx^2 + dx^3.$$

This requires 3 additions and 6 multiplication (since $dx^3 = d \times x \times x \times x$ etc). However, the nested form

$$p(x) = a + x(b + x(c + dx))$$

requires 3 additions and 3 multiplications. The saving for high-order polynomials is obvious.

Strategy 2

Avoid subtracting numbers that are almost equal. A good example is provided by the formulae for the roots of the quadratic

$$x^2 + \alpha x + \beta.$$

The exact mathematical expressions for the roots are, of course,

$$x_1 = \frac{1}{2}\left[-\alpha - \sqrt{\alpha^2 - 4\beta}\right]$$

and

$$x_2 = \frac{1}{2}\left[-\alpha + \sqrt{\alpha^2 - 4\beta}\right].$$

Let us consider the case when α is positive and α^2 is very much larger than 4β. Then $\sqrt{\alpha^2 - 4\beta}$ is approximately α. The expression for x_2 then involves subtracting numbers which are almost equal whereas that for x_1 does not. We would therefore expect the numerical evaluation of x_2 to be subject to the effects of roundoff and hence for our calculated value of x_2 to incur potentially large relative errors. However we also know that $x_1x_2 = \beta$ so, after calculating x_1, we can calculate x_2 from $x_2 = \beta/x_1$, a form which does not suffer from this problem. You might like to experiment with this on your machine.

Conclusion

All machines are subject to the effects of roundoff error since they can only hold a finite number of binary or decimal places. Each arithmetic operation is thus likely to incur a small error. In most circumstances, these effects are of no real importance. However, we should be aware of the fact that there are certain circumstances where roundoff can be important and act accordingly.