Introduction to CS412 - Fundamental concepts

Why study numerical analysis and scientific computing?

Lecture of: 22 Jan 2013

In many instances we are faced with mathematical problems where the conventional solution method "on paper" does not perform as well, when implemented on the computer. Take, for example, the quadratic equation

$$ax^2 + bx + c = 0$$

In theory, we have a perfectly conclusive method for determining the roots of this equation; simply use the quadratic formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1}$$

In fact, let us consider a very specific example:

$$0.0501x^2 - 98.78x + 5.015 = 0$$

The actual roots of this equation, rounded to 10 digits of accuracy, are:

$$x_1 = 1971.605916, \quad x_2 = 0.05077069387$$

We then proceed to evaluate the formula (1) on a computer. Let us assume that the computer stores all intermediate results of this computation with 4 decimal significant digits. Thus, we obtain the following answers:

$$x_1 = 1972, \quad x_2 = 0.0998$$

Although the value x_1 is reasonably accurate (it is, in fact, the correct answer to 4 significant digits), the value x_2 is off by almost 100% from the intended value, even when rounded to 4 digits. The reason for this behavior is hiding in the inaccurate (approximate) representation of the intermediate steps of this calculation. Particularly, the computer algorithm approximates $\sqrt{b^2 - 4ac} \approx 98.77$, discarding any subsequent digits due to its limited available precision. Substituting this value into the quadratic formula yields:

$$x_{1,2} = \frac{98.78 \pm 98.77}{0.1002}$$

The numerator involves either a sum or a difference of two near-equal quantities. When adding the 2 values, we compute the reasonably accurate $x_1 = 1972$. When subtracting, all the information that was discarded by truncating $\sqrt{b^2 - 4ac} \approx 98.77$

is now dominating the computed value of the numerator. As a consequence we obtain the compromised value $x_2 = 0.0998$.

So, subtracting two near-equal quantities is risky ... Can we avoid doing so? There is, in fact, an alternative formulation:

$$\begin{aligned} x_{1,2} &= \frac{(-b \pm \sqrt{b^2-4ac})(-b \mp \sqrt{b^2-4ac})}{2a(-b \mp \sqrt{b^2-4ac})} \\ &= \frac{b^2-(b^2-4ac)}{2a(-b \mp \sqrt{b^2-4ac})} = \frac{4ac}{2a(-b \mp \sqrt{b^2-4ac})} \\ &= \frac{2c}{-b \mp \sqrt{b^2-4ac}} \qquad (2) \end{aligned}$$

Equation (2) subtracts 2 quantities in the denominator where (1) adds them and vice versa. Appying this approach we obtain:

$$x_1 = 1003, \quad x_2 = 0.05077$$

Here the roles are reversed. x_2 is quite accurate, but x_1 is off by about 50% from the intended value. One might suggest that we selectively use (1) or (2) to compute x_1 and x_2 respectively, and avoid the subtraction-induced inaccuracies. Although this may be realistic in this isolated example, in more complex problems that arise in practice, performing such a case-by-case handling may prove impractical. In many cases it may even be impossible to predict that a value lacks accuracy (not knowing the exact value beforehand).

We do, however, have another alternative: Instead of resorting to case study, we can use a fundamentally different algorithm. Consider the following methodology (which is a special case of what we will later describe as Newton's method):

- Start with an initial guess x_0 of the solution
- Iterate
$$x_{k+1} = \frac{ax_k^2 - c}{2ax_k + b}$$
- After N iterations, take x_N as the estimated solution

Let us try it ... start with a guess $x_0 = 1$

$$x_0 = 1, \quad x_1 = 0.050313\ldots, \quad x_2 = 0.0507706\ldots$$

After just 2 iterations, x_2 is correct to more than 6 significant digits! Let us aim for the other solution by setting $x_0 = 2000$

$$x_0 = 2000, \quad x_1 = 1972.003\ldots, \quad x_2 = 1971.60599\ldots$$

As is typically the case, there are trade-offs to consider: with this iterative method, we require an "initial guess", and there is little guidance offered on how to pick a good one. Also, we need a few iterations before we can obtain an excellent approximation. On the other hand, the method does not require any square roots, thus being significantly cheaper in terms of computation cost per iteration.

The traits and trade-offs of such methods will be a point of focus for CS412, and are central to the field we call numerical analysis and scientific computing.

How are numbers stored on the computer?

Corresponding textbook chapter(s): §1.2

First, we shall review the concept of "scientific notation", which will give us some helpful insights. For any decimal number x (we assume that x is a terminating decimal number, with finite nonzero digits) we can write

$$x = a \times 10^b, \quad \text{where} \quad 1 \leq |a| < 10$$

Exception: When $x = 0$, we simply set $a = b = 0$. For example:

x (decimal notation)	x (scientific notation)
2012	2.012×10^3
412	4.12×10^2
3.14	3.14×10^0
0.000789	7.89×10^{-4}
0.2091	2.091×10^{-1}

Every decimal (or Base-10) number can be written

$$\mathtt{a_k a_{k-1} \cdots a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \cdots a_{-l}} = \sum_{i=-l}^{+k} a_i 10^i$$

For example

x	$\mathtt{a_3}$	$\mathtt{a_2}$	$\mathtt{a_1}$	$\mathtt{a_0 .}$	$\mathtt{a_{-1}}$	$\mathtt{a_{-2}}$	$\mathtt{a_{-3}}$
3.14				3	1	4	
0.037						3	7
2012		2	0	1	2		

Binary (Base-2) fractional numbers are written

$$\mathtt{b_k b_{k-1} \cdots b_2 b_1 b_0 . a_{-1} b_{-2} b_{-3} \cdots b_{-l}}_{(2)} = \sum_{i=-l}^{+k} b_i 2^i$$

where every digit b_i is now only allowed to equal 0 or 1. For example