Finding Roots and Precision

Written by Dave Pawlowski

1 Indroduction

A common problem in scientific computing is to find the solutions, or roots, of some equation

$$f(x) = 0$$

in a single variable x. Numerical techinques are useful because we aren't required to come up with a general solution in a closed form, i.e. a function. Typically, a numerical solution is found by starting with approximation to the answer, p_0, and steadily improving the solution $p_1, p_2, ...$ until we have obtained a value that is within a certain error tolerance. There are a variety of methods for doing this. The best methods are ones that ensure us of convergence, or in other words, that we will, given enough iterations, find a suitable answer. Additionally, the best methods are ones that give a sequence that converges to an answer quickly.

So, what is meant by convergence? Given a sequence of approximations to a solution for p, p_0, p_1, p_2, p_3..., if the solution converges to the answer p, then the differences, or errors $e_n = p_n - p$ must get smaller and smaller as n approaches infinity. To put it another way, if we take the ratio of successive errors, it should be less than 1. In reality, as we perform more and more iterations, then this ratio should approach a non-zero constant that is less than 1:

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = k < 1.$$

More generally, it is possible to have:

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^\alpha} = k < 1.$$

where α is some positive power called the order of convergence. When $\alpha = 1$ then we have linear convergence. When $\alpha = 2$ then we have quadratic convergence, i.e. our solution improves much faster than with linear convergence.

Using such an iterative approach to problem solving means that n really could go to infinitiy, which really isn't practical for scientific computing. Instead, we defince a tolerance, ϵ such that when $|e_n| < \epsilon$ we stop iterating and declare our solution good enough. This is called a stopping condition. If $|e_n| < \epsilon$ then it is not necessary that our solution is with in ϵ of the unknown solution. It is always possible that the error could start growing again for one reason or another. For this reason it is helpful to have a physical understanding of the problem, or even to experiment with larger than normal values of n.

2 Roots of a Quadratic

One of the most common problems is finding the solution (roots) of a quadratic equation:

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

For which the standard solution can be applied:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Certainly this equation should produce the correct solution, and in fact there is no need to perform any sort of iteration. However, in general introducing a computer into the solution process can lead to one potential source of error: round-off error.

3 Precision and Round-Off Error

When we use a computer to store a number, the computer does this physically by utilizing a limited number of binary bits. Typically, floating point numbers take up 32 bits and double precision numbers take up 64 bits. What this means for precision is that there is a limit on how many significant digits can be stored by the computer. Most computers use the IEEE 754 storage standard to represent numbers. In this manner, floating point numbers are stored as a combination of three parts: the sign, the mantissa, and the exponent. An analog is a number that is displayed in exponential format, i.e. $0.74723x10^5$. In this case, the sign is $+$, the mantissa is 0.74723, and the exponent is 5. Single precision floats have roughly 23 bits reservered for the mantissa, 8 for the exponent, and 1 for the sign, while double precision numbers have 52 bits reserved for the mantissa, 11 for the exponent, and 1 for the sign.

What this all means is that a number that requires an infinite number of digits to be represented must be rounded to a finite number of digits. The problem isn't typcially that the computer can't represent large enough numbers: single precision numbers can have exponents that range from ± 126. Rather, issues pop up when subtracting two numbers that have very similar magnitude. In this case, it is possible that significant digits may be lost.

In the example of the quadratic formula, such errors can manifest themselves if $4ac$ is much smaller than b^2. It is important to note that round-off error is an issue with nearly every computer program. However, the precision of todays computers (32 single precision) typically makes round-off error less important than other sources of error, such as truncation error, and the most important source, user error.