Cancellation and Swamping

The IEEE standard 754 requires that the FAFA holds. That is: any arithmetic operation on two floats returns the float nearest the true value. Here we discuss two principle ways information can be lost in this setting.

Consider what happens when adding a |large| number $0.d_1 d_2 \cdots d_t \times 2^m$ and a |small| number $0.e_1 e_2 \cdots e_t \times 2^p$. Since only t digits can remain in the result, the digits of the smaller number are effectively shifted to the right about $m-p$ places. What were once significant digits of the smaller number are now not as significant, and about $m-p$ of them are lost completely:

$$\begin{array}{cccccccccc} & 0 & . & d_1 & d_2 & \cdots & d_{m-p} & d_{m-p+1} & \cdots & d_{t-1} & d_t \\ + & 0 & . & 0 & 0 & \cdots & 0 & e_1 & e_2 & \ldots & e_{t-(m-p)} \\ \hline & 0 & . & d_1 & d_2 & \cdots & d_{m-p} & f_{m-p+1} & \cdots & f_{t-1} & f_t \end{array}$$

The result *is* the closest float to the true answer; it is working the way it is supposed to work. It is simply a fact of floating point arithmetic that in the addition or subtraction of small and large numbers, information from the small number is lost. This is called *swamping.*

Now consider subtracting x and y, when $y \approx x$. Suppose they share k significant digits:

$$\begin{array}{cccccccccc} & 0 & . & d_1 & d_2 & \cdots & d_k & d_{k+1} & \cdots & d_t \\ - & 0 & . & d_1 & d_2 & \ldots & d_k & e_{k+1} & \cdots & e_t \\ \hline & 0 & . & 0 & 0 & \ldots & 0 & f_{k+1} & \cdots & f_t \end{array}$$

Now, in normalized form our result is $0.f_{k+1} f_{k+2} \cdots f_t \; ?_1 \; ?_2 \cdots \; ?_k$, and there are k digits (the $?'s$) whose values, whatever they are, are meaningless. As above we have lost information, and this time we are left with a sum (or difference) with only $t-k$ accurate digits. The real danger here is apparent when we recognize that the digits that do survive (the $f's$) are precisely those which, in x and y, were most likely contaminated by previous rounding errors! The operation itself satisfies the FAFA, but it can significantly magnify the errors already present. This is called *cancellation* and is probably the single biggest reason that bad floating point algorithms are bad.

Remember, floating point addition is not associative: consider

$$z_1 = ((y+x)-x) \qquad \text{and} \qquad z_2 = (y+(x-x))).$$

In exact arithmetic $z_1 = y = z_2$ for all x and y. Let $x = 12345.6$ and $y = .4567890$. In 5-decimal digit rounding arithmetic we have

$$\mathrm{fl}(z_1) = \mathrm{fl}(\mathrm{fl}(.45679 + 12346) - 12346) = \mathrm{fl}(12346 - 12346) = 0,$$

and

$$\mathrm{fl}(z_2) = \mathrm{fl}(.45679 + \mathrm{fl}(12346 - 12346)) = \mathrm{fl}(.45679 + 0) = .45679.$$