Floating Point Numbers and Round-off Error

Floating Point Numbers

Every real number x can be written in *normalized form* in base β as

$$x = \pm r \times \beta^n, \text{ with } \frac{1}{\beta} \leq r < 1, \text{ and } n \text{ an integer}$$

with the obvious exception for $x = 0$. r is called the *mantissa* and n the *exponent* or *characteristic*. In shorthand, we can $x = \pm rEn$, ($\pi = 0.314159265E1$)

A computer can only store a finite number of different mantissas and exponents so some mechanism must be used to map the real numbers onto the computer numbers. This mechanism is called *rounding*. We will discuss only one type of rounding called rounding (others are called chopping, symmetric rounding, etc.). So, rounding means two things: it is the general process for mapping real numbers to computer numbers and it is also the particular process we are about to discuss. First, let us describe two computers which come in handy for examples.

DDC-k: (Decimal k-Digit Computer), Base $\beta = 10$, computer numbers are of the form

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

where $0 \leq d_i \leq 9$, $d_1 \geq 1$ and $-99 \leq n \leq 99$. The average calculator is a DDC-10 or DDC-12.

SPC: (Single precision computer), Base $\beta = 2$, computer numbers are of the form

$$\pm 0.b_1 b_2 \ldots b_{24} \times 2^n$$

where b_i is either 0 or 1, $b_1 = 1$ (and thus is not stored), and $-126 \leq n \leq 127$. This is essentially standard single precision on a computer (`REAL*4` in Fortran or `float` in C).

Rounding

First we take care of the sign and the exponent. The sign is stored as is (usually in 1 bit). The exponent is stored as is, if it is within the given range, otherwise we have *underflow* if the exponent is too small, or *overflow* if it is too big, these and other exceptions are dealt with differently on different machines, sometimes underflow is set to 0.

For the mantissa we apply *rounding*. Rounding produces the computer number closest to the real number. Notation: if x is a real number, $fl(x)$ is the computer representation of that number. Assume that the computer can store k digits (in base β) for the mantissa. Thus if

$$x = \pm 0.d_1 d_2 d_3 d_4 \ldots \times \beta^n$$

then with rounding

$$fl(x) = \pm 0.e_1 e_2 e_3 e_4 \ldots e_k \times \beta^n$$

where $e_k = d_k$ if $d_{k+1} < \beta/2$ or $e_k = d_k + 1$ if $d_{k+1} \geq \beta/2$, and the rest of the digits $e_1, \ldots, e_{k-1}$ are the d_i appropriately adjusted, i.e. if $d_k = \beta - 1$ and $d_{k+1} > \beta/2$, then $e_k = 0$ and $e_{k-1} = d_{k-1} + 1$, etc. In some cases the exponent could also change (and cause overflow).

For example, on a DDC-4, $fl(0.49994E0) = 0.4999E0$, $fl(0.49995E2) = 0.5000E2$, $fl(0.99995E2) = 0.1000E3$.

Roundoff-Error

So, $x \approx fl(x)$, and our first Numerical Analysis result is to precisely understand this approximation. Thus, we want to look at $x - fl(x)$. Using the above notation and assuming $x > 0$ with rounding, we have two cases to consider: (I) $d_{k+1} < \beta/2$ and (II) $d_{k+1} \geq \beta/2$. In Case I: $e_i = d_i$, $i = 1, \ldots, k$, so

$$x - fl(x) = 0.0\ldots 0d_{k+1}\ldots \times \beta^n = d_{k+1}.d_{k+2}\ldots \times \beta^{n-k-1}.$$

And, since $d_{k+1} < \beta/2$ we get $x - fl(x) \leq \beta/2 \times \beta^{n-k-1} = \frac{1}{2}\beta^{n-k}$. In Case II: we can take $e_i = d_i$, $i = 1, \ldots, k-1$ and $e_k = d_k + 1$, so, with borrowing during the subtraction, we get

$$x - fl(x) = -0.0\ldots 0(\beta - d_{k+1})(\beta - d_{k+2})\ldots \times \beta^n = -(\beta - d_{k+1}).\ldots \times \beta^{n-k-1}.$$

Now we have $d_{k+1} \geq \beta/2$ so that $\beta - d_{k+1} \leq \beta/2$ and we get $x - fl(x) \geq -\beta/2 \times \beta^{n-k-1} = -\frac{1}{2}\beta^{n-k}$. Thus, we get our first result

$$|x - fl(x)| \leq \frac{1}{2}\beta^{n-k}.$$

Next, if $x \neq 0$, then $|x| \geq \beta^{-1}\beta^n$ and so $1/|x| \leq \beta^{1-n}$. So we get our second result

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\beta^{n-k}\beta^{1-n} = \frac{1}{2}\beta^{1-k}.$$

This number $\epsilon_{mach} = \frac{1}{2}\beta^{1-k}$ is called *Machine Epsilon* and is the primary constant for expressions of machine accuracy. A useful expression involving ϵ_{mach} and derived from above, is

$$fl(x) = x(1+\delta), \qquad |\delta| \leq \epsilon_{mach}$$

In words, this expression gives us the following guideline:

> *Don't expect numbers on a computer to be what you think they are.*

> Note: on a DDC-4, $\epsilon_{mach} = \frac{1}{2}10^{1-4} = 5 \times 10^{-4}$, a DDC-$k$, $\epsilon_{mach} = 5 \times 10^{-k}$, and on a SPC, $\epsilon_{mach} = \frac{1}{2}2^{1-24} = 2^{-24} = 5.96 \times 10^{-8}$, so roughly speaking, a SPC is like a DDC-8. Double precision is roughly like a DDC-16.

Operations

The rule a computer must follow for basic arithmetic operations is that result should be the (rounded) same as exact arithmetic. So if x and y are two computer numbers, $\circ$ is the exact operation, and $\bullet$ the computer version, we have

$$x \bullet y = fl(x \circ y).$$

Thus each operation generates some roundoff error, so $x \bullet y = (x \circ y)(1+\delta)$, $|\delta| \leq \epsilon_{mach}$.