Computer representation of numbers

Real numbers are usually represented in the computer as **floating-point** numbers (fp). A fp x in **base** b can be represented as

$$x = (f)_b \times b^{(e)_b}$$

$f = \pm(.d_1 d_2 \ldots d_t)_b$ is the **mantissa**.
$e = \pm(c_{s-1} c_{s-2} \ldots c_0)_b$ is the **exponent**.
x is said to have t digits **precision**.

The fp is **normalized** if $d_1 \neq 0$, or else $d_1 = d_2 = \ldots d_t = 0$. We will assume normalized numbers, unless otherwise stated.
The exponent is also limited:$L \leq e \leq U$.
Usually, $-L = U = (aa \ldots a)_b$, $a = b - 1$.

According to the above simplified form, the largest fp is
$N_{max} = (.aa \ldots a)_b \times b^{(aa \ldots a)_b}$
and the smallest non-zero fp in absolute value is
$N_{min} = (.10 \ldots 0)_b \times b^{-(aa \ldots a)_b}$.

Notation:$\mathbf{R}_b(t,s)$ denotes set of all base b fp with absolute value inside these ranges (or else 0).

Overflow (OFL) occurs whenever a fp $> N_{max}$ or $< -N_{max}$ has to be stored in the computer.
Underflow (UFL) occurs whenever a fp is in the range $(-N_{min}, N_{min})$ has to be stored in the computer.

Notes:
• $\mathbf{R}_b(t,s)$ is finite, while $\mathbf{R}$ is infinite.
• $\mathbf{R}$ is dense, while $\mathbf{R}_b(t,s)$ is not. There are many "holes" in the real axis, which are not uniform, and are not represented on the computer.
• The representable numbers are concentrated towards 0.

A real number $x = \pm(d_k d_{k-1} d_0 . d_1 \ldots)_b$ can be easily represented in a fp form. First normalize the mantissa:

$$x = (d_k d_{k-1} d_0 . d_1 \ldots)_b = (.D_1 D_2 \ldots)_b \times b^{k+1}$$

Then, there are two common ways to convert $x \in \mathbf{R}$ to a fp $fl(x) \in \mathbf{R}_b(t,s)$.

a) **chopping**: chop after digit t of the mantissa.
b) **rounding**: chop after digit t, then round D_t up or down depending on whether $D_{t+1} \geq b/2$ or $D_{t+1} \leq b/2$, or in other words, add $b/2$ to D_{t+1}, then chop.

Example in 2 binary digits:

x	chop	rounding
$.101 \times 2^0$	$.10 \times 2^0$	$.11 \times 2^0$
$.111 \times 2^0$	$.11 \times 2^0$	$.10 \times 2^1$

In 1985, the Institute of Electrical and Electronics Engineers (IEEE) approved the so-called **IEEE Standard** for binary fp representation and arithmetic operations.

IEEE Standard fp are of the form

$$(-1)^q \times (d_0 . d_1 d_2 \ldots d_{t-1})_b \times 2^e$$

where $q = 0$ or $1, (-1)^q$ is the sign of the number.
$d_i = 0$ or 1, and $L \leq e \leq U$.
In single precision,
$t = 24, L = -126$ and $U = +127$.
In double precision,
$t = 53, L = -1022$ and $U = +1023$.
The mantissa is normalized: $d_0 = 1$.
The IEEE Standard uses rounding.

The IEEE Standard also includes some "special" numbers: $+$inf, $-$inf, NaN (Not-a-Number)
to handle indeterminate values that may arise from some computations.

Round-off error

The difference between x and $fl(x)$ is called the **round-off** error. It is roughly proportional to x, so we can write $fl(x) - x = x\sigma$, or $fl(x) = x(1 + \sigma)$, where σ is the **relative round-off** error. σ may depend on x, but there is a bound for it, independent of x.

- $|\sigma| \leq b^{1-t}$ if normalized numbers and chopping are assumed.
- $|\sigma| \leq \frac{1}{2}b^{1-t}$ if normalized numbers and rounding are assumed.

The bound for $|\sigma|$ for rounding is called **unit round-off** and is often denoted by μ.
An approximation x' to x is said to be **correct in r significant b-digits**, if $|\frac{x-x'}{x}| \leq \frac{1}{2}b^{1-r}$.

Computer arithmetic

Let $x, y \in \mathbf{R}_b(t, s)$ and $o \in \{+, -, \times, /\}$. Then for the computer's fp operation $\overline{o}$ corresponding to o the following generally holds:

$$xoy \neq fl(x\overline{o}y) \text{ and } x\overline{o}y = fl(xoy)$$

Example (in 2 decimal digits)
$x = .2 \times 10^1, y = .58 \times 10^{-5}$
$x + y = .2000058 \times 10^1$
Usually the computer can temporarily store more than t digits but the final result will be

$$x\overline{+}y = fl(x + y) = .2 \times 10^1$$

Similarly the computer application $\overline{f}(x)$ of a function f to a number $x \in \mathbf{R}_b(t, s)$ is constructed so that

$$\overline{f}(x) = fl(f(x)).$$

Machine epsilon

The smallest (non-normalized) fp E with the property $1 + E > 1$ is called the **machine epsilon**.

$E = b^{1-t}$ if chopping.
$E = \frac{1}{2}b^{1-t}$ if rounding.
Thus, $-E \leq \sigma \leq E$.

For IEEE Standard numbers, $E = 2^{1-t}$, i.e.
single prec., $E = 2^{-23} \approx 1.192 \times 10^{-7}$ and
dbl prec., $E = 2^{-52} \approx 2.220 \times 10^{-16}$

Simple error propagation

As soon as an error arises, it may then be amplified or reduced in subsequent operations.

Let $x, y \in \mathbf{R}$.
Then $fl(x) = x(1 + \sigma_1), fl(y) = y(1 + \sigma_2)$.

a) Multiplication.
The computer will compute
$fl(x)\overline{\times}fl(y) = fl(fl(x) \times fl(y))$
$= (x(1 + \sigma_1)y(1 + \sigma_2))(1 + \sigma_3)$
$= xy(1 + \sigma_1 + \sigma_2 + \sigma_1\sigma_2)(1 + \sigma_3)$
$\approx xy(1 + \sigma_1 + \sigma_2 + \sigma_3) = xy(1 + \sigma_{xy})$
where $|\sigma_{xy}| \leq 3E$.

b) Division. Similar to a).

c) Addition. Exercise.

Consider the case $x \approx y$. The error blows up.
Example (in 8 decimal digits with chopping)
Let $x = .123456790 \times 10^0$
and $y = .123456789 \times 10^0$.
$fl(fl(x) - fl(y)) = .10000000 \times 10^{-7}$.
while
$x - y = .10000000 \times 10^{-8}$.

In the above $fl(fl(x) - fl(y))$ no significant decimal digits are correct. Although the relative error in the fp representation of y is low (10^{-8}), the relative error in $fl(fl(x) - fl(y))$ is high (10^0).

Subtracting nearly equal numbers can be dangerous and the result may have no correct digits at all. This phenomenon is called **Catastrophic cancellataion**

Error propagation in computation

Consider some computation with input x and output $f(x)$.
Let $x \in$**R** with $fl(x) = x(1 + \sigma_x)$ and $f(x)$ is twice differentiable about x.
Then, if $f(x) \neq 0$, using Taylor's series we get

$$f(fl(x)) \approx f(x)(1 + \sigma_f) \text{ with } \sigma_f = \frac{xf'(x)}{f(x)}\sigma_x.$$

Since $|\sigma_x| < E$, we get

$$|\sigma_f| \leq |\frac{xf'(x)}{f(x)}|E.$$

The factor $\kappa_f = |\frac{xf'(x)}{f(x)}|$ is called the **condition number** of $f(x)$ and is a measure of the relative sensitivity of the computation of $f(x)$ on relatively small changes in x, or in other words a measure of how the relative error in x propagates in $f(x)$.

A computation is called **well-conditioned** if relatively small changes in the input, produce relatively small changes in the output, otherwise it is called **ill-conditioned**.

Example.
Let $f(x) = \sqrt{x}$. Then, the condition number of $f(x)$ is $|\frac{xf'(x)}{f(x)}| = 1/2$. For this function, the condition number is small and independent of x. Thus, the computation of the square root of a number is a well-conditioned computation.

If, for some function, $\kappa_f > E^{-1}$, we risk having no correct digits at all in the computed result $\overline{f}(x)$.

Stability of algorithms

Stability is a concept similar to conditioning, but it refers to a numerical algorithm. A numerical algorithm is stable if small changes in the algorithm parameters have a small effect on the algorithm output, otherwise it is called **unstable**.

Example (in 3 decimal digits)

$$(a - b)^2 = a^2 - 2ab + b^2$$

Let $a = 15.6, b = 15.7$
$(a - b)^2 = (-.1)^2 = .1 \times 10^{-1}$
$a^2 - 2ab + b^2 = 243.36 - 2 \times 244.92 + 246.49$
$\to 243 - 490 + 246 = -.1 \times 10^1$.

Computing the rhs we don't even get the sign correct! In this case, computing the lhs is a more stable algorithm than the rhs.