Numerical Analysis I: machine numbers

Victor Eijkhout, Karl Schulz

Computers use a finite number of bits to represent numbers, so only a finite number of number can be represented, and no irrational numbers.

Analogous to scientific notation $x = 0.6 \cdot 10^{24}$:

$$x = \pm \Sigma_{i=1}^{t} d_i \beta^{-i}\, \beta^e$$

- sign bit
- β is the base of the number system
- $0 \leq d_i \leq \beta - 1$ the digits of the *mantissa*
- $e \in [L, U]$ exponent, including sign

Some examples

	β	t	L	U
IEEE single (32 bit)	2	24	-126	127
IEEE double (64 bit)	2	53	-1022	1023
Old Cray 64bit	2	48	-16383	16384
IBM mainframe 32 bit	16	6	-64	63
packed decimal	10	50	-999	999

BCD is tricky: 3 decimal digits in 10 bits

(we will often use $\beta = 10$ in the examples, because it's easier to read for humans, but all practical computers use $\beta = 2$)

Limitations

Overflow: more than $(1-\beta^{-t-1})\beta^U$ or less than $(1-\beta^{-t-1})\beta^L$

Underflow: numbers less than $\beta^{-t-1}\cdot\beta^L$

Normalized numbers: $d_1\neq 0$; then underflow is less than $\beta^{-1}\cdot\beta^L$

Representation error

Error between number x and representation $\tilde{x}$:
absolute $x - \tilde{x}$ or $|x - \tilde{x}|$
relative $\frac{x-\tilde{x}}{x}$ or $\left|\frac{x-\tilde{x}}{x}\right|$

Equivalent: $\tilde{x} = x \pm \epsilon \Leftrightarrow |x - \tilde{x}| \leq \epsilon \Leftrightarrow \tilde{x} \in [x - \epsilon, x + \epsilon]$.

Also: $\tilde{x} = x(1 + \epsilon)$ then relative error $\left|\frac{\tilde{x}-x}{x}\right| \leq \epsilon$

example

Decimal, $t = 3$ bit mantissa: let $x = .1256$, $\tilde{x}_{\mathrm{round}} = .126$, $\tilde{x}_{\mathrm{truncate}} = .125$

Error in the 4th digit: $|\epsilon| < \beta^t$ (this example had no exponent, how about if it does?)

Note that $.126 \pm .0005$ has 3 digits correct, 3 significant; $.006 \pm .0005$ has 3 correct but only 1 significant

Normalized numbers

Require first digit in the mantissa to be nonzero.
Equivalent: mantissa part $1/beta \leq x_m < 1$

Unique representation for each number,
also: in binary this makes the first digit 1, so we don't need to store that.

Machine precision

Any real number can be represented to a certain precision:
$\tilde{x} = x(1+\epsilon)$ where
truncation: $\epsilon = \beta^{-t}$
rounding: $\epsilon = \frac{1}{2}\beta^{-t}$

This is called *machine precision*: maximum relative error.

32-bit single precision: $mp \approx 10^{-7}$
64-bit double precision: $mp \approx 10^{-16}$

Maximum attainable accuracy.

Another definition of machine precision: smallest number ϵ such that $1+\epsilon > 1$.

Addition

1. align exponents
2. add mantissas
3. adjust exponent to normalize

Example: $.100 + .200 \times 10^{-2} = .100 + .002 = .102$. This is exact, but what happens with $.100 + .255^{-2}$?

Example: $.500 \times 10^{1} + .504 = (.500 + .0504) \times 10^{1} \to .550 \times 10^{1}$

Any error comes from truncating the mantissa: if x is the true sum and $\tilde{x}$ the computed sum, then $\tilde{x} = x(1+\epsilon)$ with $|\epsilon| < 10^{-3}$

Analysis of addition

Let $s = x_1 + x_2$, and $x = \tilde{s} = \tilde{x}_1 + \tilde{x}_2$ with $\tilde{x}_i = x_i(1 + \epsilon_i)$

$$
\begin{aligned}
\tilde{x} &= \tilde{s}(1+\epsilon_3) \\
&= x_1(1+\epsilon_1)(1+\epsilon_3) + x_2(1+\epsilon_2)(1+\epsilon_3) \\
&= x_1 + x_2 + x_1(\epsilon_1+\epsilon_3) + x_2(\epsilon_1+\epsilon_3) \\
\Rightarrow \tilde{x} &= s(1+2\epsilon)
\end{aligned}
$$

$\Rightarrow$ errors are added

Assumptions: all ϵ_i approximately equal size and small;
$x_i > 0$

Multiplication

1. add exponents
2. multiply mantissas
3. adjust exponent

Example:
$.123 \times .567 \times 10^1 = .069741 \times 10^1 \rightarrow .69741 \times 10^0 \rightarrow .697 \times 10^0$.

What happens with relative errors?

Subtraction

Example: $.124 - .123 = .001 \rightarrow .1 \times 10^{-2}$: only one significant digit.

- *Cancellation* leads to loss of precision
- subsequent operations with this result are inaccurate
- $\Rightarrow$ avoid subtracting numbers that are likely close.

Example: $ax^2 + bx + c = 0 \rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
suppose $b > 0$ and $b^2 \gg 4ac$ then the "$+$" solution will be inaccurate
Better: compute $x_- = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$ and use $x_+ \cdot x_- = -c/a$.

Serious example

Evaluate $\Sigma_{n=1}^{10000} \frac{1}{n^2} = 1.644834$
in 7 digits: machine precision is 10^{-7} in single precision

First term is 1, so partial sums are ≥ 1, so $1/n^2 < 10^{-6}$ gets ignored
$\Rightarrow$ last 7000 terms are ignored
$\Rightarrow$ sum is 1.644725: 4 correct digits

Solution: sum in reverse order; exact result in single precision
Why?
consider ratio of two terms:

$$\frac{n^2}{(n-1)^2} = \frac{n^2}{n^2-2n+1} = \frac{1}{1-2/n+1/n^2} \approx 1 + \frac{2}{n}$$

with aligned exponents:

$$\begin{array}{rl|ll} n-1: & .00\cdots0 & 10\cdots00 & \\ n: & .00\cdots0 & 00\cdots01 & 0\cdots0 \\ & & \quad n \text{ bits} & \end{array}$$

The last digit in the smaller number is not lost if $n < 1/\epsilon$

Another serious example

Previous example was due to finite representation; this example is more due to algorithm itself.

Consider $y_n = \int_0^1 \frac{x^n}{x-5} dx = \frac{1}{n} - 5y_{n-1}$ (monotonically decreasing)
$y_0 = \ln 6 - \ln 5$.

In 3 decimal digits:

computation		correct result
$y_0 = \ln 6 - \ln 5 = .182\vert 322 \times 10^1 \ldots$		1.82
$y_1 = .900 \times 10^{-1}$		.884
$y_2 = .500 \times 10^{-1}$		.0580
$y_3 = .830 \times 10^{-1}$	going up?	.0431
$y_4 = -.165$	negative?	.0343

Reason? Define error as $\tilde{y}_n = y_n + \epsilon_n$, then

$$\tilde{y}_n = 1/n - 5\tilde{y}_{n-1} = 1/n + 5n_{n-1} + 5\epsilon_{n-1} = y_n + 5\epsilon_{n-1}$$

so $\epsilon_n \geq 5\epsilon_{n-1}$: exponential growth.

Consequences of roundoff

Multiplication and addition are not associative:
problems for parallel computations.

Operations with "same" outcomes are not equally stable:
matrix inversion is unstable, elimination is stable