## How are numbers stored on the computer? (§1.2)

First, remember what we call "scientific notation"

For any decimal number $x$ (assume finite nonzero digits) we can write

$$x = a \times 10^b$$

where $1 \le |a| < 10$

Exception: When $x=0$, we simply set $a=b=0$

Examples

| $x$ (decimal notation) | $x$ (scientific notation) |
|---|---|
| 2011 | $2.011 \times 10^3$ |
| 412 | $4.12 \times 10^2$ |
| 3.14 | $3.14 \times 10^0$ |
| 0.000789 | $7.89 \times 10^{-4}$ |
| 0.2091 | $2.091 \times 10^{-1}$ |

Every decimal (or base-10) number can be written

$$a_n \cdots a_4 a_3 a_2 a_1 a_0 . a_{-1} a_{-2} a_{-3} \cdots a_{-l} = \sum_{i=-l}^{+k} a_i 10^i$$

↑ decimal point

$a_i \in \{0,1,2,\cdots,9\}$

For example

| | $a_4$ | $a_3$ | $a_2$ | $a_1$ | $a_0$ . | $a_{-1}$ | $a_{-2}$ | $a_{-3}$ | $a_{-4}$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3.14 | | | ... | 0 | 3 | 1 | 4 | 0 | ... | $= 3\times10^{0}+1\times10^{-1}+4\times10^{-2}$ |
| 0.037 | | | | | ... | 0 | 3 | 7 | 0 ... | $= 3\times10^{-2}+7\times10^{-3}$ |
| 2001 | ... 0 | 2 | 0 | 1 | 1 | 0 ... | | | | $= 2\times10^{3}+1\times10^{1}+1\times10^{0}$ |

Binary numbers (base $-2$) are written

$$b_k \ldots b_3 b_2 b_1 b_0 . b_{-1} b_{-2} \ldots b_{-\ell}\ {}_{(2)}$$ but equal $\sum_{i=-\ell}^{i=k} b_i 2^i$

(compare with $\sum_{i=-\ell}^{k} b_i 10^i$)

and every $b_i = 0$ or $1$.

example:

$$5.75 = 4 + 1 + 0.5 + 0.25$$

$$= 1\times2^{2} + 1\times2^{0} + 1\times2^{-1} + 1\times2^{-2} = 101.11_{(2)}$$

($b_2$, $b_0$, $b_{-1}$, $b_{-2}$)

$$17.5 = 16 + 1 + 0.5$$

$$= 1\times2^{4} + 1\times2^{0} + 1\times2^{-1} = 10001.1_{(2)}.$$

($b_4$, $b_0$, $b_{-1}$)

Note that certain numbers which are finite decimals, actually are periodic in binary

e.g $0.4_{(10)} = 0.0110\underline{0011}\underline{0011}\ldots_{(2)} = 0.011\overline{0011}_{(2)}$

## Machine numbers (binary floating point numbers)

The numbers stored on the computer are, essentially "binary numbers" in scientific notation

$$x = \pm a \cdot 2^{b}$$

$a$ is called the mantissa

$b$ is called the exponent

The convention here is that $\frac{1}{2} \leq a < 1$. The idea is that, for any number $x$, we can always divide it by an appropriate power of 2, s.t the result will be within $[\frac{1}{2}, 1)$.

e.g. $x = 5$

Try $b=0$ $\quad 5 = x = a \cdot 2^0 \Rightarrow a = 5$ too much

$b=2$ $\quad 5 = x = a \cdot 2^2 \Rightarrow a = 1.25$ still too much

$b=3$ $\quad 5 = a \cdot 2^3 \Rightarrow a = 0.625$ ok!

Lastly, we have to write $a = 0.625$ in binary

$$0.625 = 0.5 + 0.125 = 1 \times 2^{-1} + 1 \times 2^{-3} = 0.101_{(2)}$$

Thus $\quad 5 = x = +\, 0.101_{(2)} \times 2^3$

How about a different way?

Remember, in decimal:

$$314 = 31.4 \times 10^{-1} = 0.314 \times 10^{1} = \ldots \text{ etc.}$$

In binary

$$5_{(10)} = 101_{(2)}$$

$$= 10.1 \times 2^1 = 1.01 \times 2^2 = 0.101 \times 2^3$$

In general, a machine number is stored as: 1/20/2011

$$X = \pm\ 0.1\, a_1 a_2 \cdots a_{k-1} a_k \cdot 2^{b}$$

Single Precision $k = 23$ $\quad -126 \le b \le 127$

(max number $\approx \pm 3.4 \times 10^{38}$)

Double Precision $k = 52$ $\quad -1022 \le b \le 1023$

(max number $\approx \pm 1.8 \times 10^{308}$)

S.P has "23 binary significant digits". How much is that in decimal?

Rule of thumb $2^{10} \approx 10^{3}$

i.e. 10 binary significant digits $\approx$ 3 decimal

Thus: single precision $\Rightarrow$ about 7 decimal significant digits

double precision $\Rightarrow$ a bit more than 15!

## Absolute and relative error

1/20/11 L6

All computations are approximate, due to the limited precision on computer.

Exact quantity : $q$

Quantity on the computer : $\hat{q}$

2 measures of Error :

Absolute error = $|q - \hat{q}|$

important when we want to examine accuracy within a certain interval

Relative error = $\frac{|q - \hat{q}|}{|q|}$

important when we care about error as % of actual value.

Rounding When only $n$ digits

Round to closest

$\pi = 3.14159265$

| | | | Truncate |
|---|---|---|---|
| $\cong$ | 3.1 | (2 s.d) | 3.1 |
| | 3.14 | (3 s.d) | 3.14 |
| | 3.142 | | 3.141 |
| | 3.1416 | | 3.1415 |

Rounding also occurs in binary

$0.10110111011110\ldots$ =

| Truncated | Rounded | # sig. digits |
|---|---|---|
| 0.1 | 1.0 | 1 |
| 0.10 | 0.11 | 2 |
| 0.101 | 0.110 | 3 |
| 0.1011 | 0.1011 | 4 |
| 0.10110 | 0.10111 | 5 |

## Machine $\varepsilon$ (epsilon)

There are a number of (slightly different) definitions in literature. We will define the machine $\varepsilon$ as the smalles positive machine representable number, such that

$$1+\varepsilon \neq 1 \quad \text{(on the computer)}$$

Why isn't the above inequality always true for any $\varepsilon > 0$ ?

e.g.

$$1 = 0.1\underbrace{0000\ldots0}_{23}{}_{(2)} \cdot 2^1$$

$$+\ 2^{-25} = 0.\underbrace{00\ldots0}_{24}01 \times 2^1$$

---

$$1+2^{-25} = 0.1\underbrace{000\ldots0}_{23}01 \times 2^1$$

which is rounded to $0.1\underbrace{0\ldots0}_{23} \times 2^1 = 1$ !

However

$$1+2^{-24} = 0.1\underbrace{000\ldots.0}_{23\text{ digits}}1\times 2^1$$

$$(\text{rounded to}) = 0.1\underbrace{000\ldots.01}_{23}\times 2^1$$

$$> 1 \quad \text{(just barely)}$$

Thus, $\varepsilon = 2^{-24}$ $(\approx 6\times 10^{-8})$

## Roundoff error

The importance of $\varepsilon$ is that it corresponds to the maximum relative error incurred when converting an exact value to a machine-representable number

$$\frac{|\hat{q}-q|}{|q|} \leq \varepsilon$$

$q$ : actual number
$\hat{q}$ : machine representation

Another way to express this: Let $\varepsilon_1 = \dfrac{\hat{q}-q}{q}$

Then $\hat{q} = (1+\varepsilon_1)\,q$, where $|\varepsilon_1| \leq \varepsilon$.

For example, assume the real result of a computation is $1,000,000,000$. $(= q)$.

$$\hat{q} = q \pm \varepsilon q \overset{\varepsilon \approx 6\times10^{-8}\times10^{9}}{=} 1,000,000,000 \pm 60 \; !$$

We have no way of telling whether the actual solution was $1,000,000,005$ or $999,999,970$ $\Rightarrow$ all look the same to the computer.

(Partial) Solution: Use higher precision

(Better) Solution: Make sure that

(1) You can tolerate this error

(2) Other algorithms that use this result can tolerate this error.

# Square roots & quadratic equ revisited

1/20/2011 L1

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

In theory, just use the formula $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

In practice, we saw its dangerous w/ errors.

Alternative?

First try simpler equation:

$$x^2 + a = 0$$

(we know the solution, $x = \sqrt{a}$ !)

Even now computing $\sqrt{..}$ can be expensive.

How about this:

→ start with some initial guess $x^{(1)} = x^*$

→ Iterate the sequence

$$x^{(k+1)} = \frac{(x^{(k)})^2 + a}{x_k}$$

Let us <u>assume</u> that this sequence converges.

What will the limit be?

$$\text{let} \quad \lim x_n = A$$

$$\lim (x_{n+1}) = \lim \left( \frac{x_n^2 + a}{2x_n} \right)$$

$$A = \frac{A^2 + a}{2A}$$

$$\Rightarrow 2A^2 = A^2 + a \Rightarrow A^2 = a \Rightarrow A = \sqrt{a}\,! \text{ (or } -\sqrt{a})$$

thus, if $x_n$ has a limit, it will be the desired square root. Note that only fundamental operations (addition / multiplication / division) were used.