< >

Appendix A1

Appendix

A1 Machine Errors

Typically, in a computer *real* numbers are stored as follows:

○	e (exponent; 8 bits)	m (mantissa; 23 bits)

or, in a more usual notation,

$$x = \circ\, m \leq 2^{e\,-\,e_0}$$

- The mantissa m is *normalized*, i.e. shifted to the left as far as possible, such that there is a 1 in the first position; each left-shift by one position makes the exponent e smaller by 1. (Since the leftmost bit of m is then known to be 1, it need not be stored at all, permitting one further left-shift and a corresponding gain in accuracy; m then has an effective length of 24 bits.)
- The *bias* e_0 is a fixed, machine-specific *integer* number to be added to the "actual" exponent $e - e_0$, such that the stored exponent e remains positive.

EXAMPLE: With a *bias* of $e_0 = 151$ (and keeping the high-end bit of the mantissa) the internal representation of the number 0.25 is, using $1/4 = (1 \leq 2^{22}) \leq 2^{-24}$ and $-24 + 151 = 127$,

$$\frac{1}{4} = \boxed{+}\boxed{127}\boxed{1\;0\;0\;\ldots\;0\;0}$$

Before any addition or subtraction the exponents of the two arguments must be equalized; to this end the *smaller* exponent is increased, and the respective mantissa is right-shifted (decreased). All bits of the mantissa that are thus being "expelled" at the right end are lost for the accuracy of the result. The resulting error is called *roundoff error*. By *machine accuracy* we denote the smallest number that, when added to 1.0, produces a result $\neq 1.0$. In the above example the number $2^{-22} \equiv 2.38 \leq 10^{-7}$, when added to 1.0, would just produce a result $\neq 1.0$, while the next smaller representable number $2^{-23} \equiv 1.19 \leq 10^{-7}$ would leave not a rack behind:

$$\begin{array}{ll}
1.0 & \boxed{+}\boxed{129}\boxed{1\;0\;0\;\ldots\;0\;0} \\
+2^{-22} & \boxed{+}\boxed{107}\boxed{1\;0\;0\;\ldots\;0\;0} \\
= & \boxed{+}\boxed{129}\boxed{1\;0\;0\;\ldots\;0\;1}
\end{array}$$

but:

$$\begin{array}{ll}
1.0 & \boxed{+}\boxed{129}\boxed{1\;0\;0\;\ldots\;0\;0} \\
+2^{-23} & \boxed{+}\boxed{106}\boxed{1\;0\;0\;\ldots\;0\;0} \\
= & \boxed{+}\boxed{129}\boxed{1\;0\;0\;\ldots\;0\;0}
\end{array}$$

A particularly dangerous situation arises when two almost equal numbers have to be subtracted. For example:

$$\begin{array}{ll}
 & \boxed{+}\boxed{35}\boxed{1\;1\;1\;\ldots\;1\;1\;1} \\
- & \boxed{+}\boxed{35}\boxed{1\;1\;1\;\ldots\;1\;1\;0} \\
= & \boxed{+}\boxed{35}\boxed{0\;0\;0\;\ldots\;0\;0\;1} \\
= & \boxed{+}\boxed{14}\boxed{1\;0\;0\;\ldots\;0\;0\;0}
\end{array}$$

Note that in the last (normalization) step the mantissa is arbitrarily filled up by zeros; the uncertainty of the result is 50%.

An important application:
There is an everyday task in which such small differences may arise: solving the **quadratic equation** $ax^2 + bx + c = 0$. The usual formula

$$x_{1,2} = \frac{- b \circ \sqrt{b^2 - 4ac}}{2a} \qquad (8.91)$$

will yield inaccurate results whenever $ac << b^2$. Since in writing a program one must always provide for the worst possible case, it is recommended to use the equivalent but less error-prone formula

$$x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q} \qquad (8.92)$$

with

$$q \equiv -\frac{1}{2} \square\, b + sgn(b) \sqrt{b^2 - 4ac} \,\square \qquad (8.93)$$

<u>EXERCISE:</u> Assess the machine accuracy of your computer by trying various negative powers of 2, each time adding and subtracting the number 1.0 and checking whether the result is zero.

---

vesely 2006

< >