Numerical Analysis

Daming Li

Department of Mathematics, Shanghai JiaoTong University,
Shanghai, 200240, China
Email: lidaming@sjtu.edu.cn

October 21, 2014

Floating-Point Numbers and Roundoff Errors

Most high-speed computers deal with real numbers in the binary system, in contrast to the decimal system that humans prefer to use

$$427.325 = 4\times10^2+2\times10^1+7\times10^0+3\times10^{-1}+2\times10^{-2}+5\times10^{-3}$$

The expression on the right employs powers of 10 together with the digits 0, 1,2, 3, 4, 5, 6, 7, 8, 9.

$$1001.11 = 1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$$

any integer $\beta > 1$ can be used as the base for a number system. Numbers represented in base β will contain digits $0, 1, 2, 3, 4, ..., \beta - 1$.

$$(1001.11101)_2 = (9.90625)_{10}$$

Normalized Scientific Notation

In the decimal system, any real number can be expressed in normalized scientific notation.

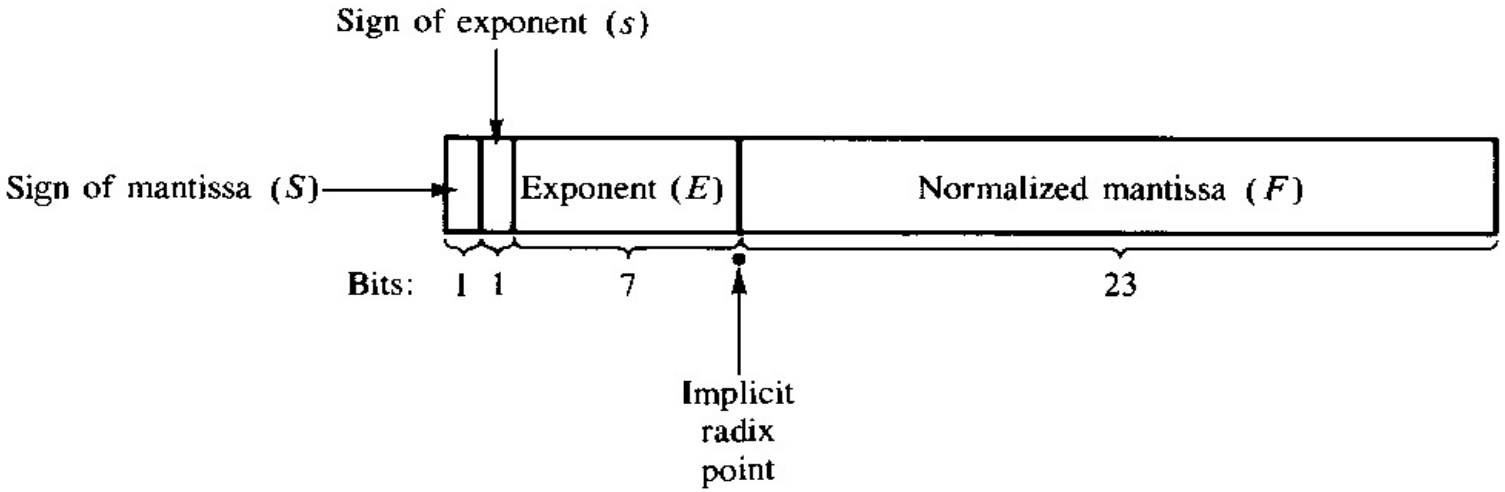
$$732.5051 = .7325051 \times 10^3, \quad -0.005612 = -.5612 \times 10^{-2}$$

$$x = \pm r \times 10^n, \quad \frac{1}{10} \leq r < 1$$

$$x = \pm q \times 2^m, \quad \frac{1}{2} \leq q < 1$$

The number q is called the mantissa and the integer m the exponent. In a binary computer, both q and m will be represented as base 2 numbers.

Hypothetical Computer MARC-32



Nonzero normalized machine numbers are bit strings whose values are decoded as follows:

$$x = (-1)^S \times q \times 2^m, \quad q = (0.1F)_2, \quad m = (-1)^s \times E$$

Largest and smallest number in MARC-32. Distribution of float numbers.

Nearby Machine Numbers

$$x = q \times 2^m, \quad \frac{1}{2} \le q < 1, \quad |m| \le 127$$

What is the machine number closest to x.

$$x = (.a_1 a_2 \cdots a_{24} a_{25} a_{26} \cdots)_2 \times 2^m, \quad a_1 = 1$$

by chopping,

$$x' = (.a_1 a_2 \cdots a_{24})_2 \times 2^m$$

Another nearby machine number lies to the right of x. It is obtained by rounding up.

$$x'' = ((.a_1 a_2 \cdots a_{24})_2 + 2^{-24}) \times 2^m$$

Observe that x' lies to the left of x on the real line.

Nearby Machine Numbers

If x is presented better by x',

$$|x - x'| \leq \frac{1}{2}|x'' - x'| = \frac{1}{2} \times 2^{-24} \times 2^m = 2^{m-25}$$

$$\frac{|x - x'|}{|x|} \leq \frac{2^{m-25}}{q \times 2^m} \leq \frac{2^{m-25}}{\frac{1}{2} \times 2^m} = 2^{-24}$$

If x is presented better by x'',

$$\frac{|x - x''|}{|x|} \leq 2^{-24}$$

x is a nonzero real number within the range of the machine, then the machine number $fl(x)$ closest to x satisfies

$$\frac{|x - fl(x)|}{|x|} \leq 2^{-24}, \quad fl(x) = x(1 + \delta), \quad |\delta| \leq 2^{-24}$$

The number 2^{-24} that occurs in the preceding inequalities is called the unit roundoff error for the MARC-32.

Nearby Machine Numbers

If m is too large, we say that an overflow has occurred. If m is too small, we say that an underflow has occurred.
Roundoff errors must be expected to be present whenever data are read into a computer. Even a simple number like $1/10$ cannot be stored exactly in the MARC-32 computer.

$$\frac{1}{10} = (0.0001\ 1001\ 1001\ 1001 \cdots)_2$$

Floating-Point Error Analysis

As a model. We assume that whenever two machine numbers are to be combined arithmetically, the combination is first correctly formed, then normalized, rounded off, and finally stored in memory in a machine word.
Let the symbol $\odot$ stand for any one of the four basic arithmetic operations $+$, $-$, $\times$, or $/$. If x and y are machine numbers, and if $x \odot y$ is to be computed and stored, then the closest that we can come to actually fitting $x \odot y$ in a single machine word is to round it off to $fl(x \odot y)$ and store that number.

Machine Epsilon

If a machine operates with base β and carries n places in the mantissa of its floating-point numbers, then

$$fl(x) = x(1+\delta), \quad |\delta| \leq \epsilon$$

$$fl(x \odot y) = (x \odot y)(1+\delta), \quad |\delta| \leq \epsilon$$

$\epsilon = \frac{1}{2}\beta^{1-n}$ in the case of correct rounding and $\epsilon = \beta^{1-n}$ in the case of chopping. The number ϵ is the unit roundoff error and is a characteristic of the computing machine, its operating system, and the mode of computation (whether single- or multiple-precision).

Relative Error Analysis

We present next a theorem that illustrates how the relative roundoff error in long calculations can be analyzed. The theorem states roughly that in adding $n + 1$ positive machine numbers, the relative roundoff error should not exceed $n\epsilon$.
Let $x_0, x_1, ..., x_n$ be positive machine numbers in a computer whose unit roundoff error is ϵ. Then the relative roundoff error in computing $\sum_{i=0}^{n} x_i$ in the usual way is at most $(1 + \epsilon)^n - 1$. This quantity is approximately $n\epsilon$.

Machine Epsilon

Let $S_k = x_0 + x_1 + \cdots + x_k$ and S_k^* be what the computer calculates instead of S_k. The recursive formula for these quantities are

$$\begin{cases} S_0 = x_0 \\ S_{k+1} = S_k + x_{k+1} \end{cases}, \quad \begin{cases} S_0^* = x_0 \\ S_{k+1}^* = fl(S_k^* + x_{k+1}) \end{cases}$$

Define

$$\rho_k = \frac{S_k^* - S_k}{S_k}, \quad \delta_k = \frac{S_{k+1}^* - (S_k^* + x_{k+1})}{S_k^* + x_{k+1}}$$

$$\begin{aligned} \rho_{k+1} &= \frac{S_{k+1}^* - S_{k+1}}{S_{k+1}} = \frac{(S_k^* + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\ &= \frac{(S_k(1+\rho_k) + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\ &= \delta_k + \rho_k \frac{S_k}{S_{k+1}}(1+\delta_k) \end{aligned}$$

Since $S_k < S_{k+1}$ and $|\delta_k| \le \varepsilon$,

Machine Epsilon

$$|\rho_{k+1}| \leq \varepsilon + |\rho_k|(1+\varepsilon) = \varepsilon + \theta|\rho_k|$$

where $\theta = 1 + \varepsilon$. Thus

$$|\rho_n| \leq \varepsilon + \theta\varepsilon + \theta^2\varepsilon + \cdots + \theta^{n-1}\varepsilon = (1+\varepsilon)^n - 1 \approx 1 + n\varepsilon$$

Absolute and Relative Errors; Loss of Significance

absolute error : $|x - x^*|$
relative error : $\frac{|x-x^*|}{|x|}$
In scientific measurements, it is almost always the relative error that is significant.
Loss of Significance: Although roundoff errors are inevitable and difficult to control, there are other types of errors in computation that are under our control. The subject of numerical analysis is largely preoccupied with understanding and controlling errors of various kinds. Here we shall take up one type of error that is often the result of careless programming.

Loss of Significance: Subtraction of Nearly Equal Quantities

$$\begin{aligned} x &= .3721478693 \\ y &= .3720230572 \\ x - y &= .0001248121 \end{aligned}$$

If this calculation were to be performed in a decimal computer having a five-digit mantissa,

$$\begin{aligned} fl(x) &= .37215 \\ fl(y) &= .37202 \\ fl(x) - fl(y) &= .00013 \end{aligned}$$

The relative error is then very large:

$$\left|\frac{x - y - [fl(x) - fl(y)]}{x - y}\right| = \left|\frac{.0001248121 - .00013}{.0001248121}\right| \approx 4\%$$

Loss of Significance: Subtraction of Nearly Equal Quantities

The assignment statement

$$y \leftarrow \sqrt{x^2 + 1} - 1$$

involves subtractive cancellation and loss of significance for small values of x. Rewrite

$$y = \frac{x^2}{\sqrt{x^2 + 1} + 1}$$

Loss of Significance: Subtraction of Nearly Equal Quantities

Exactly how many significant binary bits are lost in the subtraction $x - y$ when x is close to y?
If x and y are positive normalized floating-point binary machine numbers such that $x > y$ and

$$2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p}$$

then at most q and at least p significant binary bits are lost in the subtraction $x - y$.

Loss of Significance: Subtraction of Nearly Equal Quantities

Proof. We only prove the lower bound. The normalized binary floating-point forms for x and y are

$$x = r \times 2^n, \quad y = s \times 2^m \quad \left(\frac{1}{2} \le r, s < 1\right)$$

Since x is larger than y, the computer may have to shift y so that y has the same exponent as x before performing the subtraction of y from x. Hence, we must write y as $y = (s \times 2^{m-n}) \times 2^n$ and then we shall have $x - y = (r - s \times 2^{m-n}) \times 2^n$. The mantissa of this number satisfies

$$r - s \times 2^{m-n} = r\left(1 - \frac{y}{x}\right) < 2^{-p}$$

To normalize the computer representation of $x - y$, a shift of at least p bits to the left is required. Then at least p spurious zeros are attached to the right end of the mantissa, which means that at least p bits of precision have been lost.

Stable and Unstable Computations; Conditioning

Numerical processes: stable and unstable. Closely related are the concepts of well-conditioned problems and badly-conditioned problems.

Numerical Instability

$$\begin{cases} x_0 = 1, \quad x_1 = 1/3 \\ x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1} \end{cases}$$

with the solution $x_n = 1/3^n$. But this algorithm is unstable. there is a possibility that the error in x_1 will be propagated into x_{15} with a factor of $(13/3)^{14}$.

A numerical process is unstable if small errors made at one stage of the process are magnified in subsequent stages and seriously degrade the accuracy of the overall calculation.

Conditioning

The words condition and conditioning are used informally to indicate how sensitive the solution of a problem may be to small relative changes in the input data. A problem is ill conditioned if small changes in the data can produce large changes in the answers. For certain types of problems, a condition number can be denned. If that number is large, it indicates an ill-conditioned problem.

If x is perturbed slightly, what is the effect on $f(x)$?

$$f(x+h) - f(x) = f'(\xi)h \approx hf'(x)$$

$$\frac{f(x+h) - f(x)}{f(x)} \approx \frac{hf'(x)}{f(x)} = \left[\frac{xf'(x)}{f(x)}\right]\left(\frac{h}{x}\right)$$

Condition number: $\frac{xf'(x)}{f(x)}$

Conditioning: examples

Locating a zero (or root) of a function f. We assume that r is a simple root, so that $f'(r) \neq 0$. If we perturb the function f to $F = f + \epsilon g$, where is the new root?

$$F(r+h) = f(r+h) + \epsilon g(r+h) = 0$$

$$\left[f(r) + hf'(r) + \frac{1}{2}h^2 f''(\xi)\right] + \epsilon\left[g(r) + hg'(r) + \frac{1}{2}h^2 g''(\eta)\right] = 0$$

$$h \approx -\epsilon \frac{g(r)}{f'(r) + \epsilon g'(r)} \approx -\epsilon \frac{g(r)}{f'(r)}$$

Conditioning: examples

As an illustration of this analysis, we consider a classic example given by Wilkinson. Let

$$f(x) = \prod_{k=1}^{20}(x-k), \quad g(x) = x^{20}$$

The roots of f are obviously the integers $1, 2, ..., 20$. How is the root $r = 20$ affected by perturbing f to $f + \epsilon g$. The answer is

$$h \approx= -\epsilon \frac{g(20)}{f'(20)} \approx -\epsilon \times 10^9$$

Conditioning: examples

Solving linear system: $Ax = b$. Condition number:

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|$$

If the solution of $Ax = b$ is rather insensitive to small changes in b, then small perturbations in b result in only small perturbations in the computed solution x. In this case, A is said to be well conditioned. This situation corresponds to the condition number $\kappa(A)$ being of only modest size. On the other hand, if the condition number is large, then A is ill conditioned and any numerical solution of $Ax = b$ must be accepted with a great deal of skepticism.