0.3 Floating Point Representation of Real Numbers

Most computers deal with real numbers in the binary number system, in contrast to the decimal number system that human prefers. Computer works internally in the binary system but communicates with its human users in the decimal system. The computer must execute conversion procedures. Although the users do not concern with this conversion, they do involve errors.

Computer cannot operate real numbers expressed with more than a fixed number of digits. The word length of the computer places a restriction on the precision with which real numbers can be represented. Even a simple number like $1/10$ cannot be stored exactly in any binary machine. It requires an infinite binary expression $\frac{1}{10} = (0.0001100110011001\cdots)_2$.

For example, if we read 0.1 into a 32-bit computer workstation and then print it out using decimal system, we obtain the following result: 0.100000001490116119384765625000 00000 000000. Usually, we won't notice this conversion error since printing using the default format would show us 0.1.

0.3.1 Floating point formats

In the decimal system, any real number can be expressed in the **normalized scientific notation** format. This means that the decimal point is shifted and appropriate powers of 10 are supplied so that all digits are to the right of the decimal point and the first digit displayed is not 0.

Example:

$$732.5051 = 0.7325051 \times 10^3$$

$$-0.005612 = -0.5612 \times 10^{-2}$$

In general, a nonzero real number x can be represented in the form $x = \pm r \times 10^n$ with r is a number in the range $0.1 \le r < 1$ and n is an integer (positive, negative or zero). Of course, if $x = 0$, then $r = 0$. In all other cases, we can adjust n so that r lies in the given range.

There are several models for computer arithmetic of floating point numbers. The one we choose to discuss is the so-called IEEE 754 Floating Point Standard. The IEEE standard consists of a set of binary representations of real number. A floating point number consists of three parts, the sign, a mantissa, which contains the string of significant bits, and an exponent. The form of a e_1 normalized IEEE floating point number is

$$\pm 1.bbb\cdots b \times 2^p$$

where N b's are the mantissa, p is an M-bit binary number representing the exponent. Normalization means that the leading it must 1.

precision	sign	exponent (M)	mantissa (N)
single	1	8	23
double	1	11	52
Long double	1	15	64

Each floating point number is assigned (1+M+N) bits. It has the form

$$se_1e_2\cdots e_Mb_1b_2\cdots b_N$$

Definition: *The number machine epsilon, denoted* ε_{mach}*, is the distance between 1 and the smallest floating point number greater than 1. For the IEEE double precision floating point standard,* $\varepsilon_{mach} = 2^{-52}$.

Be sure to understand many numbers below ε_{mach} are machine representable, even though adding them to 1 may be no effect.

0.3.2 Rounding

Rounding is an important concept in scientific computing. Consider a positive decimal number x of $0.\square\square\square\cdots\square\square\square$ with m digits to the right of the decimal point. One rounds x to n decimal place (n<m) in a manner that depends on the value of the (n+1)-th digit. If this digit is 0,1,2,3,or 4, then the n-th digit is not changed and all the following digits are discarded. If it is a 5,6,7,8 or 9, then the n-th digit is increased by one unit and the remaining digits are discarded. (The situation with 5 as the (n+1)-st digit can be handled in a variety of ways. For example, some choose to round up only when the previous digit is even, assuming that this happens about half time. For simplicity, we always choose to round up in this situation).

Here are some examples of seven-digit numbers being correctly rounded to four digits

$$0.1735 \leftarrow 0.1735499$$
$$1.000 \leftarrow 0.9999500$$
$$0.4322 \leftarrow 0.4321609$$

Remark: *If x is rounded so that* $\tilde{x}$ *is the n-digit approximation to it. Then*

$$|x-\tilde{x}| \le \frac{1}{2}\times 10^{-n} \qquad (2)$$

To see why this is true, we reason as follows:

If the digit of x is 0,1,2,3 or 4, then $x = \tilde{x} + \varepsilon$ *with* $\varepsilon < \frac{1}{2} \times 10^{-n}$*, and inequality (2) follows.*

If it is 5,6,7,8 or 9 then $\tilde{x} = \hat{x} + 10^{-n}$ *where* $\hat{x}$ *is a number with the same n digits as x and all digits beyond the n-th are zero. Now* $x = \hat{x} + \delta \times 10^{-n}$ *with* $\delta \geq \frac{1}{2}$ *and* $\tilde{x} - x = (1 - \delta) \times 10^{-n}$ *since* $1 - \delta < \frac{1}{2}$*. Inequality (2) follows if x is a decimal number, the chopped or truncated n-digit approximation to it is the number* $\tilde{x}$ *obtains by simply discarding all digits beyond the n-th. First, we have* $|x - \hat{x}| < 10^{-n}$*. The relationship between x and* $\hat{x}$ *is such that x-* $\hat{x}$ *has 0 in the first n places and* $x = \hat{x} + \delta \times 10^{-n}$ *with* $0 \leq \delta < 1$*. Hence, we have* $|x - \hat{x}| = |\delta| \times 10^{-n} < 10^{-n}$ *and inequality (3) follows.*

In binary, we have a similar way to do rounding.

IEEE Rounding to Nearest Rule:
For double precision, if the 53rd bit to the right of the binary point is 0, then the round down (truncate after the 52nd bit). If the 53rd bit is 1, then round up (add 1 to 52 bit), unless all known bits to the right of the 1 are 0's, in which case 1 is added to bit 52 if and only if bit 52 is 1.

Definition: *Denote the IEEE double precision floating point number associated to x, using Rounding to Nearest Rule, by fl(x).*

Similar to the remark above, we have $\frac{|fl(x) - x|}{|x|} \leq \frac{1}{2} \varepsilon_{mach}$

0.3.3 Machine Representation
This section we discuss a few more details about how a floating point representation is implemented on a computer. Each double precision floating point number is assigned an 8 byte word, or 64 bits, to store three parts. The sign is stored in the first bit, followed by 11 bits representing the exponent and the 52 bits following the decimal point representing the mantissa.

The sign bit s is 0 for a positive number and 1 for a negative number. The 11 bits represent the exponent come from the positive binary integer number resulting from adding 1023 to the exponent. For exponents between -1022 and 1023, this covers values of these 11 bits from 1 to

2046. The number 1023 is called the **exponent bias** of the double precision. The special exponent 2047 is used to represent infinity if the mantissa bit string is all zeros, and NaN, which stands for Not a Number, if the mantissa bit string is not all zeros. The special exponent 0, is interpreted as the non-normalized floating point form

$$\pm 0.b_1 b_2 b_3 \cdots b_{52} \times 2^{-1022}$$

These numbers are called subnormal floating point numbers. The smallest representable number in double precision is 2^{-1074}. The subnormal numbers includes +0 and -0. +0 has sign 0, exponent all zeros and mantissa 52 zeros. For -0, all is exactly same, except the sign bit is 1.

0.3.3 Addition of floating point number

Machine addition consists of lining up the decimal points of the two numbers to be added, adding them, and then storing the result again as a floating point number. *The addition itself can be done in higher precision (with more than 52 bits) since the addition takes place in a register dedicated just to that purpose.*

Example: Compare $1 + 2^{-53}$ with 1 using MATLAB

Example: Compare $1 - 2^{-53}$ with 1 using MATLAB