Realistic Floating Point Data Types

A bit is a single binary digit—either 0 or 1. Bits are grouped into bytes. A byte is the smallest addressible block of data in computer memory. Typically, a byte consists of 8 bits. Imagine a street lined with houses, where each house contains exactly eight people. The people are the bits, the houses are the bytes, and the street addresses of the houses are the memory locations of each byte.

The IEEE 754 standard defines four floaing point data types: single precision, extended single precision, double precision and extended double precision. These are summarized in the following table:

type	size	mantissa	exponent
single precision	4 bytes	23 bits	8 bits
extended single precision	∼6 bytes	31 bits	11 bits
double precision	8 bytes	52 bits	11 bits
extended double precision	10 bytes	64 bits	15 bits

The computers widely available today contain FPUs which implement the IEEE 754 floating point standard in hardware. Computations in Matlab are done by default using double precision floating point. Moreover, most general purpose programming languages provide floating point data types which correspond directly to the those in the IEEE 754 standard and allow for efficient use of the FPU. One notable exception is the programming language Maple which uses an arbitary precision base 10 floating point representation that conforms to the IEEE 854 standard. When bugs were discovered in the FPU of the early Intel Pentium processors, the makers of Maple could proudly say, our software is unaffected. However, the relative speed disadvantage of implementing floating point operations in software makes Maple unsuitable for many numeric applications.

Generally, we write non-zero floating point numbers in a way such that the first digit appearing in the mantissa is non-zero. For example, in base ten we write

$$-0.03 = -3.0 \times 10^{-2}, \qquad 230 = 2.3 \times 10^{2}, \qquad 0.912 \times 10^{5} = 9.12 \times 10^{4},$$

and so forth. When we do the same thing for base two floating point numbers

$$0.0010111 = 1.0111 \times 2^{-11}$$

we note that the first non-zero digit is 1. Moreover, since the first digit of a normalized base-two floating point number is always 1, then there is no need to store it. Such a method of representing a binary floating point number is said to have a hidden bit. The single precision and double precision formats of the IEEE 754 standard have hidden bits, whereas the extended precision formats do not.

As a digression, let us write down the decimal equivalent of the base two floating point number given above. This is

$$1.0111 \times 2^{-11} = \left(1 + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4}\right) \times 2^{-3} = 0.1796875.$$

Let x^* be an approximation of $x \in \mathbf{R}$. We say that x^* is correct to n decimal places if

$$|e| = |x^* - x| \leq 0.5 \times 10^{-n}.$$

Similarly, we say x^* is correct to n significant digits if

$$|\tilde{e}| = \left|\frac{x^* - x}{x}\right| \leq 5 \times 10^{-n}.$$

Consider the approximation $x^* = 5.4321$ of some number $x \in \mathbf{R}$. If x^* was properly rounded, then it should be correct to 4 decimal places and 5 significant digits. We show this intuition is consistent with the above definitions. First note that the only real numbers which could be properly rounded to 5.4321 are $x \in (5.43205, 5.43215)$. Therefore

$$|e| < 0.00005 = 0.5 \times 10^{-4}$$

is consistent with x^* being correct to 4 decimal places. Similarly

$$|\tilde{e}| < 0.00005/5.43205 \leq 0.9204628087 \times 10^{-5} \leq 5 \times 10^{-5}$$

is consistent with x^* being correct to 5 significant digits.

Generally, a base 10 floating point approximation with n decimal digits in the mantissa will be accurate to n significant digits. Let us use this observation to estimate the number of significant digits in the IEEE 754 single and double precision floating point data types. A normalized mantissa consisting of n decimal digits can be chosen in $9 \cdot 10^{n-1}$ different ways. The mantissa of a normalized single precision floating point number with a hidden bit can be chosen in 2^{23} different ways. Setting $9 \cdot 10^{n-1} \approx 2^{23}$ yields

$$n \approx 23 \log_{10} 2 + 1 - \log_{10} 9 \approx 6.969447391$$

or almost 7 significant digits. Similarly for double precision

$$n \approx 52 \log_{10} 2 + 1 - \log_{10} 9 \approx 15.69931727$$

which is more than 15 significant digits.

Although Matlab uses double precision for all calculations, by default it prints only 5 significant digits for its output. The output format can be changed to show all 15 significant digits by means of the `format long` command.

Matlab Example 4a

```
>> x=sqrt(2)
x = 1.4142
>> format long
>> x
x = 1.41421356237310
```