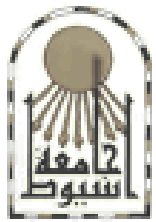
Chapter 1
Error Analysis

Dr. Kareem Elgindy

Lecturer
Mathematics Department, Faculty of Science
Assiut University

19 October, 2015



Outline

Outline

Round-Off Error

- One type of approximation inevitably made in scientific computing is in **representing real numbers on a computer**.
  - In fact, a computer is in general built to handle pieces of information of a fixed size called a **word**[1] The *memory of a computer* is usually designed to store and retrieve data in **word-length quantities**[2].
- Since a real or integer number is usually stored in a word, **only a subset $\mathbb{F}$ of finite dimension of $\mathbb{R}$ can be represented**.
- Since any given integer only has a finite number of digits, **integers can be exactly represented**, **provided that the word-length suffices to store all the digits in its representation**.

[1] The natural data size of a computer. The size of a word varies from one computer to another, depending on the CPU.

[2] The number of bits that a CPU can process at one time, or it is the number of bits actually stored or retrieved in one memory access. Typical word-lengths are 32 and 64 bits.

Round-Off Error

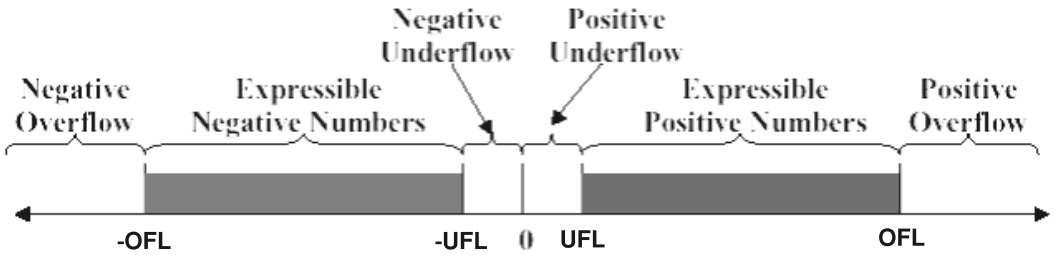


Figure 1: Typical distribution of numbers represented in a computer. OFL refers to the **overflow level** defined to be the largest positive representable number on a computer while UFL refers to the **underflow level** defined to be the smallest positive normalized representable number on a computer.

- Numbers greater than the OFL result in **overflow error**[1], and typically cause the computations to stop.
- Numbers occurring in calculations that have a magnitude less than the UFL result in **underflow error**[2], and are generally set to zero.

[1] An error that occurs when the computer attempts to handle a number that is too large for it.

[2] An error that occurs when a computer attempts to represent a number that is too small for it (that is, a number too close to zero).

Round-Off Error

- Non-integer numbers are considerably more cumbersome since **infinitely many digits are needed to represent most of them**, regardless of which numeral system we employ.
  - This means that **most non-integer numbers cannot be represented in a computer without committing an error**, which is usually referred to as the **round-off error**.
- Before we describe the standard computer representation of non-integer numbers, we need to understand the basic concept of **significant figures**.

Significant Figures

What do we mean by significant figures?

The significant figures of a measured (or calculated) quantity are the **meaningful digits** in it, or **the digits that express the precision of the measurement** instead of its magnitude. There are conventions which you should learn and follow for how to express numbers so as to properly indicate their significant figures:

- ***Non-zero digits are significant***.
- ***Zeroes between non-zero digits are significant***.
- ***All leading zeroes***[a] **are not significant**.
- ***For numbers with decimal points, zeroes to the right of a non-zero digit are significant***.
- ***For numbers without decimal points, trailing zeroes***[b] ***are not significant***.

[a] Zeroes to the left of the first non-zero digit.

[b] A sequence of 0s in any positional representation of a number, after which no other digits follow.

Round-Off Error

- The standard computer representation of real numbers is the **floating-point format** with a fixed number of bits. In this approach, which is very much like **scientific notation**, the number is expressed as

$$\pm 0.s \times b^e \tag{1}$$

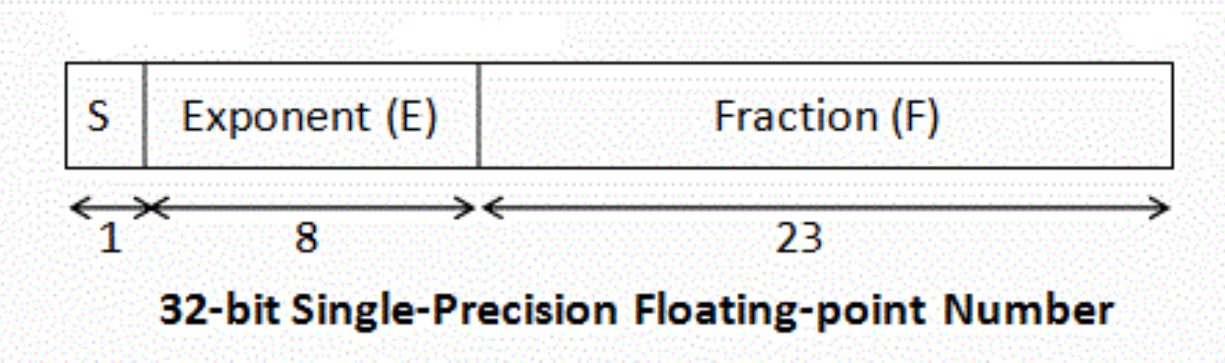
where $s = a_1 a_2 \ldots a_k$ is the **significand** (**coefficient** or **mantissa**) whose precision (length) k is the maximum number of significant digits a_i that are stored, b is the **base of the number system** being used, and e is an integral number called the **exponent** (or **characteristic**), and is bounded by $L \leq e \leq U$, where L and U are the lower and upper limits of the exponent range, respectively.

Round-Off Errors

MATLAB uses **IEEE Standard 754**[a].

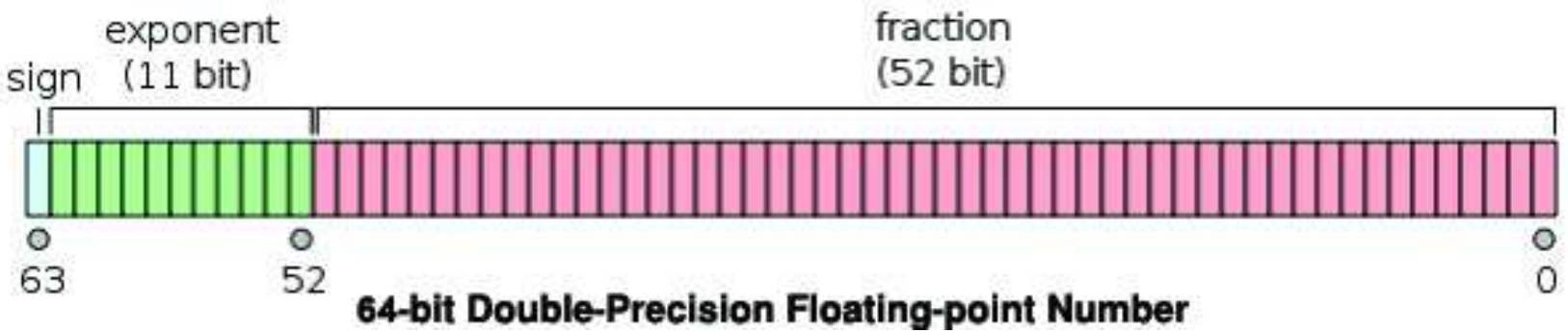
[a]The IEEE Standard 754 is a technical standard for floating-point computation established in 1985 by IEEE.

- On a typical computer system, a **"single precision (SP)" (32-bit) binary floating-point number** has a coefficient of 23 bits, an exponent of 8 bits, and one sign bit, for a total of 32 bits or 4 bytes.
  - This gives at least 7 decimal digits of precision.



Round-Off Errors

- On the other hand, a **"double precision (DP)" (64-bit) binary floating-point number (MATLAB's default numerical data type)** has a coefficient of 52 bits, an exponent of 11 bits, and one sign bit, for a total of 64 bits or 8 bytes.
  - This gives at least 15 decimal digits of precision.



Round-Off Errors

- Say $s = a_1 a_2 \ldots a_k$. A floating-point system is said to be **normalized** if the leading digit a_1 **is always non-zero unless the number represented is zero**.
  - Thus, in a normalized floating-point system, the digits of the mantissa of a given non-zero floating-point number always satisfy

$$1 \leq a_1 \leq b-1, \tag{2a}$$
$$0 \leq a_i \leq b-1, \quad i = 2, \ldots, k. \tag{2b}$$

Floating-point systems are usually normalized because:

(i) **The representation of each number is unique**.

(ii) **No digits are wasted on leading zeroes**, thereby maximizing precision.

(iii) In a binary $(b = 2)$ system, **the leading bit is always one and thus need not be stored (the "hidden bit")**, thereby gaining one extra bit of precision for a given field width. This technique is called **hidden bit normalization**.

Outline

Rounding

Any positive real number x within the numerical range of the machine can be **normalized** to the decimal form

$$x = 0.a_1 \ldots a_k a_{k+1} a_{k+2} \ldots \times 10^n, \quad 1 \leq a_1 \leq 9, \text{ and } \quad 0 \leq a_i \leq 9, \tag{3}$$

for each $i = 2, 3, \ldots$. The floating-point form of x, denoted **fl(x)**, is obtained by terminating the mantissa of x at k digits.

In **base-10** floating-point systems, there are **two common ways** of performing this termination:

- One method, called **chopping** (**rounding towards zero**), is to simply chop off the digits $a_{k+1} a_{k+2} \ldots$. This produces the normalized decimal floating-point form

$$\text{fl}(x) = 0.a_1 a_2 a_3 \ldots a_k \times 10^n. \tag{4}$$

Rounding

- The other method, called **rounding to the nearest** (or simply **rounding**):
  - For rounding, when $a_{k+1} \geq 5$, we add 1 to a_k, and chop off all but the first k decimal digits to obtain $\text{fl}(x)$; that is, we ***round up***, and we obtain a number of the form:

$$\text{fl}(x) = (0.a_1a_2a_3\ldots a_k + 0.\underbrace{000\ldots1}_{k-\text{digits}}) \times 10^n = 0.\delta_1\delta_2\delta_3\ldots\delta_k \times 10^m. \tag{5}$$

  Notice that in this case, **the digits (and even the exponent) might change**.

  - When $a_{k+1} < 5$, we simply chop off all but the first k digits; so we ***round down***. In this case, $\delta_i = a_i$, for each $i = 1, 2, \ldots, k$, and $m = n$.

Rounding

Example 1

Determine the five digit (a) chopping and (b) rounding values of the irrational number π.

Solution 1

The number π has an infinite decimal expansion of the form $\pi = 3.14159265\ldots$. Written in **normalized decimal form**, we have

$$\pi = 0.314159265\ldots \times 10^1.$$

(a) The floating-point form of π using five digit chopping is

$$\text{fl}(\pi) = 0.31415 \times 10^1.$$

Hence, the five digit chopping value of π is 3.1415.

Rounding

Solution 1

(b) The sixth digit of the decimal expansion of π is a 9, so the floating-point form of π using five digit rounding is

$$\text{fl}(\pi) = (0.31415 + 0.00001) \times 10^1 = 0.31416 \times 10^1.$$

Hence, the five digit rounding value of π is 3.1416.

Rounding

If k decimal digits and **chopping** are used for the machine representation of

$$x = 0.a_1a_2 \ldots a_t a_{k+1} \ldots \times 10^e,$$

then

$$\left|\frac{x - \mathrm{fl}(x)}{x}\right| = \left|\frac{0.a_1a_2 \ldots a_k a_{k+1} \ldots \times 10^e - 0.a_1a_2 \ldots a_k \times 10^e}{0.a_1a_2 \ldots \times 10^e}\right|$$
$$= \left|\frac{0.a_{k+1}a_{k+2} \ldots \times 10^{e-k}}{0.a_1a_2 \ldots \times 10^e}\right| = \left|\frac{0.a_{k+1}a_{k+2} \ldots}{0.a_1a_2 \ldots}\right| \times 10^{-k}.$$

Since $a_1 \neq 0$, the minimal value of the denominator is 0.1. The numerator is bounded above by 1. As a consequence,

$$\left|\frac{x - \mathrm{fl}(x)}{x}\right| < \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}.$$

Rounding

In a similar manner, an upper bound for the relative error when using k-digit **rounding** arithmetic can be determined as follows:

- If $a_{k+1} < 5$, then $\text{fl}(x) = 0.a_1a_2 \ldots a_k \times 10^e$. Similar to the previous derivation, we find that

$$\left|\frac{x - \text{fl}(x)}{x}\right| = \left|\frac{0.a_{k+1}a_{k+2}\ldots}{0.a_1a_2\ldots}\right| \times 10^{-k} < \frac{0.5}{0.1} \times 10^{-k}$$

$$\Rightarrow \boxed{\left|\frac{x - \text{fl}(x)}{x}\right| < 0.5 \cdot 10^{-k+1}.}$$

Rounding

- If $a_{k+1} \geq 5$, then $\text{fl}(x) = 0.a_1a_2 \ldots a_k \times 10^e + 10^{e-k}$. Therefore

$$\begin{aligned}
\left|\frac{x - \text{fl}(x)}{x}\right| &= \left|\frac{0.a_1a_2 \ldots a_ka_{k+1} \ldots \times 10^e - 0.a_1a_2 \ldots a_k \times 10^e - 10^{e-k}}{0.a_1a_2 \ldots \times 10^e}\right| \\
&= \left|\frac{0.a_{k+1}a_{k+2} \ldots \times 10^{e-k} - 10^{e-k}}{0.a_1a_2 \ldots \times 10^e}\right| \\
&= \left|\frac{0.a_{k+1}a_{k+2} \ldots - 1}{0.a_1a_2 \ldots}\right| \times 10^{-k} \leq \frac{0.5}{0.1} \times 10^{-k}.
\end{aligned}$$

$$\Rightarrow \boxed{\left|\frac{x - \text{fl}(x)}{x}\right| \leq 0.5 \times 10^{-k+1}.}$$

Rounding

Hence rounding to nearest is the most accurate.

Remark 1.

Rounding to the nearest is the **default rounding rule** in IEEE systems.

Rounding

Remark 2.

The upper bound for the **absolute error** when using ***k*-digit chopping arithmetic** for the machine representation of

$$x = 0.a_1a_2 \dots a_k a_{k+1} \dots \times 10^n,$$

is

$$\boxed{|x - \text{fl}(x)| < 10^{-k}.}$$

In a similar manner, an upper bound for the absolute error when using ***k*-digit rounding arithmetic** is $\boxed{0.5 \times 10^{-k}}$.

Rounding

Remark 3.

Whereas it is true that round-off errors are usually small, **when repeated within long and complex algorithms, they may give rise to catastrophic effects**.

Rounding

Example 2

Compute the upper bound for the **relative error** when using **3-digit rounding arithmetic** for the machine representation of a certain real number x.

Solution 2

$$\left|\frac{x - \text{fl}(x)}{x}\right| \leq 0.5 \times 10^{-3+1} = 0.005 = 0.5\%.$$