Numerical Analysis

NTNU

Tsung-Min Hwang

September 14, 2003

1 Floating-Point Number and Roundoff Error

1 Floating-Point Number and Roundoff Error

- Normalized scientific notation for the decimal number system of x:

$$x = \pm r \times 10^n,$$

where

$$\frac{1}{10} \leq r < 1,$$

and n is an integer (positive, negative, or zero).

1 Floating-Point Number and Roundoff Error

- Normalized scientific notation for the decimal number system of x:

$$x = \pm r \times 10^n,$$

where

$$\frac{1}{10} \leq r < 1,$$

and n is an integer (positive, negative, or zero).

  - r is called the mantissa and n is the exponent.
  - The leading digit in the fraction is not zero.

1 Floating-Point Number and Roundoff Error

- Normalized scientific notation for the decimal number system of x:

$$x = \pm r \times 10^n,$$

where

$$\frac{1}{10} \leq r < 1,$$

and n is an integer (positive, negative, or zero).

  – r is called the mantissa and n is the exponent.

  – The leading digit in the fraction is not zero.

  – For example,

$$\begin{aligned} 42.965 &= 0.42965 \times 10^2, \\ -0.00234 &= -0.234 \times 10^{-2}. \end{aligned}$$

Computer Arithmetic

Computer Arithmetic

- Scientific notation for the binary number system of x:

$$x = \pm q \times 2^m$$

with

$$\frac{1}{2} \leq q < 1,$$

and some integer m.

Computer Arithmetic

- Scientific notation for the binary number system of x:

$$x = \pm q \times 2^m$$

with

$$\frac{1}{2} \leq q < 1,$$

and some integer m. For example,

$$\begin{aligned}(1001.1101)_2 &= 1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} \\ &= 0.10011101 \times 2^4 \\ &= (9.8125)_{10}\end{aligned}$$

Computer Arithmetic

Example 1.1 *What is the binary representation of* $\frac{2}{3}$*?*

Computer Arithmetic

Example 1.1 *What is the binary representation of $\frac{2}{3}$?*

Solution: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\ldots)_2.$$

Computer Arithmetic

Example 1.1 *What is the binary representation of $\frac{2}{3}$?*

Solution: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\ldots)_2.$$

Multiply by 2 to obtain

$$\frac{4}{3} = (a_1.a_2a_3\ldots)_2.$$

Therefore, we get $a_1 = 1$ by taking the integer part of both sides.

Computer Arithmetic

Example 1.1 *What is the binary representation of $\frac{2}{3}$?*

Solution: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\ldots)_2.$$

Multiply by 2 to obtain

$$\frac{4}{3} = (a_1.a_2a_3\ldots)_2.$$

Therefore, we get $a_1 = 1$ by taking the integer part of both sides. Subtracting 1, we have

$$\frac{1}{3} = (0.a_2a_3a_4\ldots)_2.$$

Example 1.1 *What is the binary representation of $\frac{2}{3}$?*

Solution: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\ldots)_2.$$

Multiply by 2 to obtain

$$\frac{4}{3} = (a_1.a_2a_3\ldots)_2.$$

Therefore, we get $a_1 = 1$ by taking the integer part of both sides. Subtracting 1, we have

$$\frac{1}{3} = (0.a_2a_3a_4\ldots)_2.$$

Repeating the previous step, we arrive at

$$\frac{2}{3} = (0.101010\ldots)_2.$$

■

Computer Arithmetic

- Only a relatively small subset of the real number system is used for the representation of all the real numbers.

Computer Arithmetic

- Only a relatively small subset of the real number system is used for the representation of all the real numbers.
- This subset, which are called the *floating-point numbers*, contains only rational numbers, both positive and negative.

Computer Arithmetic

- Only a relatively small subset of the real number system is used for the representation of all the real numbers.
- This subset, which are called the *floating-point numbers*, contains only rational numbers, both positive and negative.
- When a number can not be represented exactly with the fixed finite number of digits in a computer, a near-by floating-point number is chosen for approximate representation.

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

  – $a_1 \neq 0$, hence $a_1 = 1$.

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

  – $a_1 \neq 0$, hence $a_1 = 1$.

  – If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t.

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

  - $a_1 \neq 0$, hence $a_1 = 1$.
  - If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t.
  - There are two ways of performing this termination.

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

  - $a_1 \neq 0$, hence $a_1 = 1$.
  - If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t.
  - There are two ways of performing this termination.

1. **chopping:** simply discard the excess bits $a_{t+1}, a_{t+2}, \ldots$ to obtain

$$fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m.$$

Computer Arithmetic

- For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0,$$

denote the normalized scientific binary representation of x.

  – $a_1 \neq 0$, hence $a_1 = 1$.

  – If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t.

  – There are two ways of performing this termination.

    1. **chopping:** simply discard the excess bits $a_{t+1}, a_{t+2}, \ldots$ to obtain

    $$fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m.$$

    2. **rounding up:** add $2^{-(t+1)} \times 2^m$ to x and then chop the excess bits to obtain a number of the form

    $$fl(x) = \pm 0.\delta_1 \delta_2 \cdots \delta_t \times 2^m.$$

    In this method, if $a_{t+1} = 1$, we add 1 to a_t to obtain $fl(x)$, and if $a_{t+1} = 0$, we merely chop off all but the first t digits.

Computer Arithmetic

Computer Arithmetic

Definition 1.1 (Roundoff error) *The error results from replacing a number with its floating-point form is called* roundoff error *or* rounding error*.*

Computer Arithmetic

Definition 1.1 (Roundoff error) *The error results from replacing a number with its floating-point form is called* roundoff error *or* rounding error*.*

Definition 1.2 (Absolute Error and Relative Error) *If x is an approximation to the exact value $x^\star$, the* absolute error *is $|x^\star - x|$ and the* relative error *is $\frac{|x^\star - x|}{|x^\star|}$, provided that $x^\star \neq 0$.*

Computer Arithmetic

Definition 1.1 (Roundoff error) *The error results from replacing a number with its floating-point form is called* roundoff error *or* rounding error*.*

Definition 1.2 (Absolute Error and Relative Error) *If x is an approximation to the exact value $x^\star$, the* absolute error *is $|x^\star - x|$ and the* relative error *is $\frac{|x^\star - x|}{|x^\star|}$, provided that $x^\star \neq 0$.*

Remark 1.1 *As a measure of accuracy, the absolute error may be misleading and the relative error more meaningful.*

Computer Arithmetic

Computer Arithmetic

- If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

Computer Arithmetic

- If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

$$\begin{aligned}\frac{|x - fl(x)|}{|x|} &= \frac{|0.00\cdots 0a_{t+1}a_{t+2}\cdots \times 2^m|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots \times 2^m|} \\ &= \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t}.\end{aligned}$$

Computer Arithmetic

- If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

$$\begin{aligned}\frac{|x - fl(x)|}{|x|} &= \frac{|0.00\cdots 0a_{t+1}a_{t+2}\cdots \times 2^m|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots \times 2^m|} \\ &= \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t}.\end{aligned}$$

  Since $a_1 \neq 0$, the minimal value of the denominator is $\frac{1}{2}$. The numerator is bounded above by 1.

Computer Arithmetic

- If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

$$\begin{aligned}\frac{|x - fl(x)|}{|x|} &= \frac{|0.00\cdots 0a_{t+1}a_{t+2}\cdots \times 2^m|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots \times 2^m|}\\ &= \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t}.\end{aligned}$$

Since $a_1 \neq 0$, the minimal value of the denominator is $\frac{1}{2}$. The numerator is bounded above by 1. As a consequence

$$\left|\frac{x - fl(x)}{x}\right| \leq 2^{-t+1}.$$

Computer Arithmetic

- If t-digit rounding arithmetic is used and
  - $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m$.

Computer Arithmetic

- If t-digit rounding arithmetic is used and
  - $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m$. A bound for the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2}\cdots|} \times 2^{-t} \leq 2^{-t},$$

since the numerator is bounded above by $\frac{1}{2}$.

Computer Arithmetic

- If t-digit rounding arithmetic is used and
  - $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m$. A bound for the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_t a_{t+1} a_{t+2}\cdots|} \times 2^{-t} \leq 2^{-t},$$

    since the numerator is bounded above by $\frac{1}{2}$.
  - $a_{t+1} = 1$, then $fl(x) = \pm(0.a_1 a_2 \cdots a_t + 2^{-t}) \times 2^m$. The upper bound for relative error becomes

$$\frac{|x - fl(x)|}{|x|} = \frac{|1 - 0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_t a_{t+1} a_{t+2}\cdots|} \times 2^{-t} \leq 2^{-t},$$

    since the numerator is bounded by $\frac{1}{2}$ due to $a_{t+1} = 1$.

Computer Arithmetic

- If t-digit rounding arithmetic is used and
  - $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1a_2\cdots a_t \times 2^m$. A bound for the relative error is

    $$\frac{|x - fl(x)|}{|x|} = \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t} \le 2^{-t},$$

    since the numerator is bounded above by $\frac{1}{2}$.
  - $a_{t+1} = 1$, then $fl(x) = \pm(0.a_1a_2\cdots a_t + 2^{-t}) \times 2^m$. The upper bound for relative error becomes

    $$\frac{|x - fl(x)|}{|x|} = \frac{|1 - 0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t} \le 2^{-t},$$

    since the numerator is bounded by $\frac{1}{2}$ due to $a_{t+1} = 1$.

  Therefore the relative error for rounding arithmetic is

  $$\left|\frac{x - fl(x)}{x}\right| \le 2^{-t} = \frac{1}{2} \times 2^{-t+1}.$$

Computer Arithmetic

- The number $\varepsilon_M \equiv 2^{-t+1}$ is referred to as the *unit roundoff error* or *machine epsilon*.

Computer Arithmetic

- The number $\varepsilon_M \equiv 2^{-t+1}$ is referred to as the *unit roundoff error* or *machine epsilon*. The floating-point representation, $fl(x)$, of x can be expressed as

$$fl(x) = x(1+\delta), \qquad |\delta| \le \varepsilon_M. \tag{1}$$

Computer Arithmetic

- The number $\varepsilon_M \equiv 2^{-t+1}$ is referred to as the *unit roundoff error* or *machine epsilon*. The floating-point representation, $fl(x)$, of x can be expressed as

$$fl(x) = x(1+\delta), \qquad |\delta| \le \varepsilon_M. \tag{1}$$

- In 1985, the IEEE (Institute for Electrical and Electronic Engineers) published a report called *Binary Floating Point Arithmetic Standard 754-1985*. In this report, formats were specified for single, double, and extended precisions, and these standards are generally followed by microcomputer manufactures using floating-point hardware.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

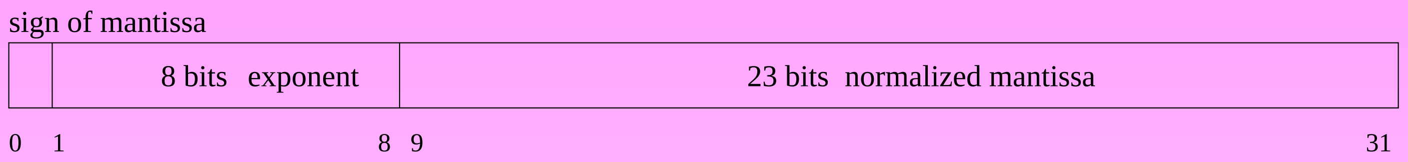

Figure 1: 32-bit single precision.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

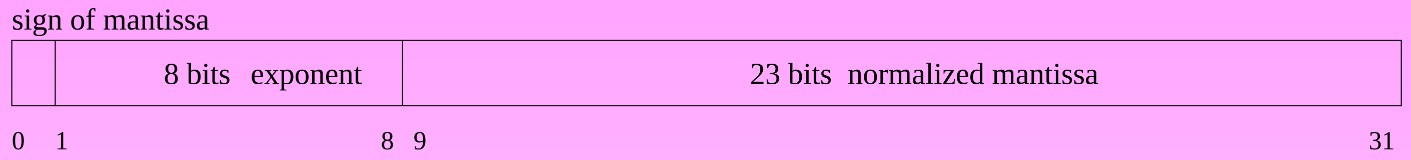

Figure 1: 32-bit single precision.

- The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

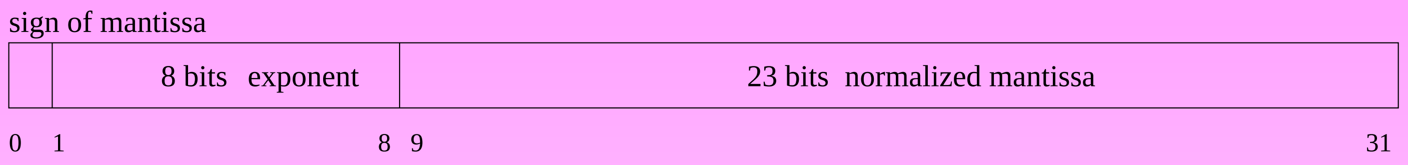

Figure 1: 32-bit single precision.

- The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f.
- The base for the exponent and mantissa is 2, and the actual exponent is $c - 127$. The value of c is restricted by the inequality $0 \le c \le 255$.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

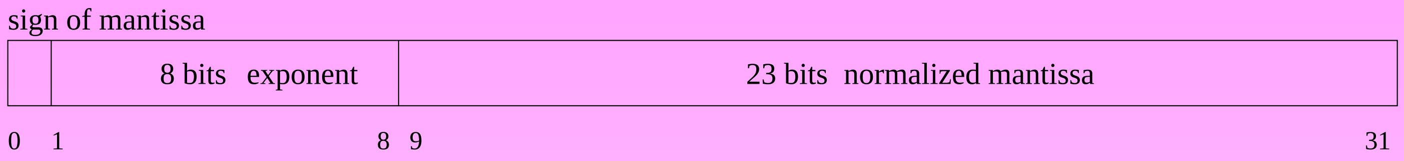

Figure 1: 32-bit single precision.

- The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f.
- The base for the exponent and mantissa is 2, and the actual exponent is $c - 127$. The value of c is restricted by the inequality $0 \leq c \leq 255$.
- The actual exponent of the number is restricted by the inequality $-126 \leq c - 127 \leq 128$.

Computer Arithmetic

☞ The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 1.

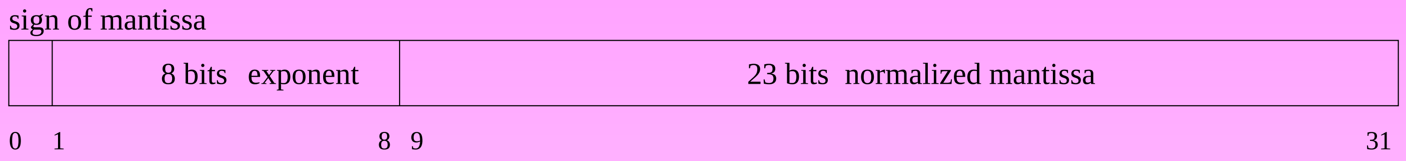

Figure 1: 32-bit single precision.

- The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f.
- The base for the exponent and mantissa is 2, and the actual exponent is $c - 127$. The value of c is restricted by the inequality $0 \leq c \leq 255$.
- The actual exponent of the number is restricted by the inequality $-126 \leq c - 127 \leq 128$.
- A normalization is imposed that requires that the leading digit in fraction be 1, and this digit is not stored as part of the 23-bit mantissa.

Computer Arithmetic

Computer Arithmetic

- The mantissa f actually corresponds to 24 binary digits (i.e., precision $t = 24$),

Computer Arithmetic

- The mantissa f actually corresponds to 24 binary digits (i.e., precision $t = 24$), the machine epsilon is

$$\varepsilon_M = 2^{-24+1} = 2^{-23} \approx 1.192 \times 10^{-7}. \tag{2}$$

- This approximately corresponds to 6 accurate decimal digits. And the first single precision floating-point number greater than 1 is $1 + 2^{-23}$.

Computer Arithmetic

- The mantissa f actually corresponds to 24 binary digits (i.e., precision $t = 24$), the machine epsilon is

$$\varepsilon_M = 2^{-24+1} = 2^{-23} \approx 1.192 \times 10^{-7}. \tag{2}$$

- This approximately corresponds to 6 accurate decimal digits. And the first single precision floating-point number greater than 1 is $1 + 2^{-23}$.
- The largest number that can be represented by the single precision format is approximately $2^{128} \approx 3.403 \times 10^{38}$, and the smallest positive number is $2^{-126} \approx 1.175 \times 10^{-38}$.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

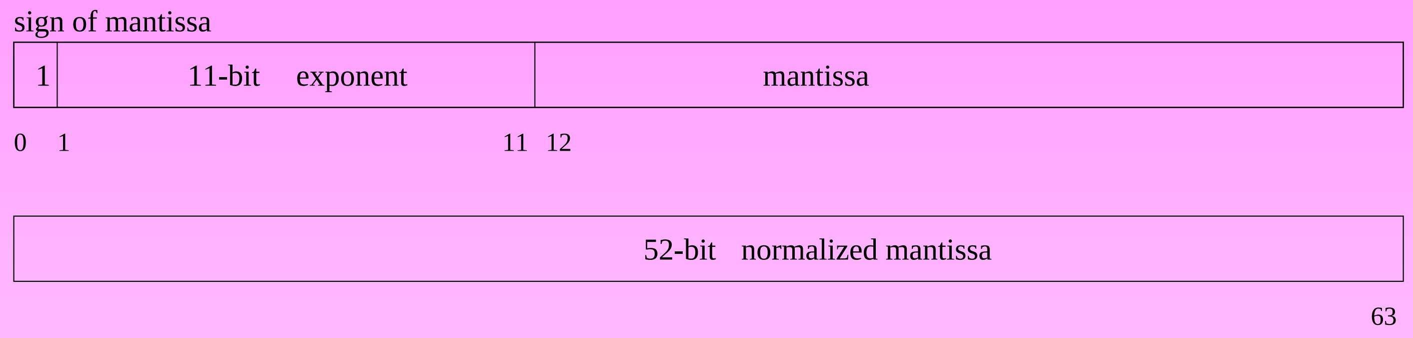

Figure 2: 64-bit double precision.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

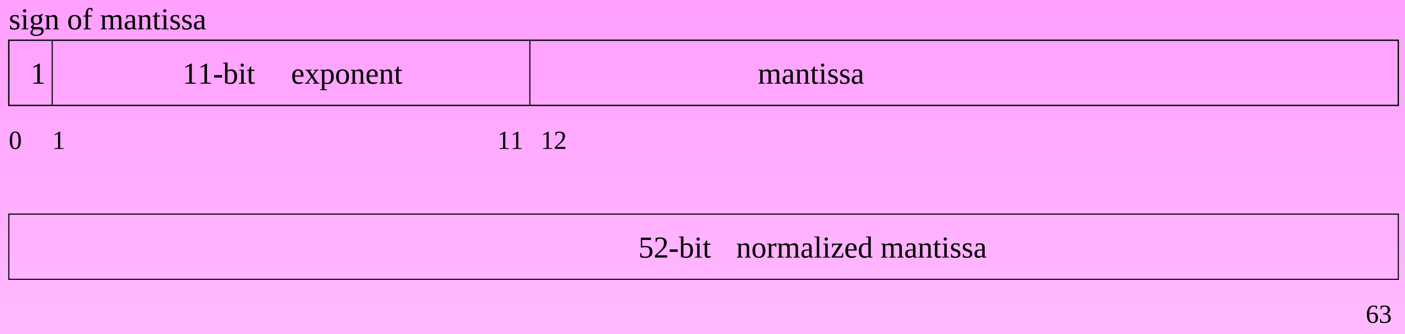

Figure 2: 64-bit double precision.

- The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent c and a 52-bit mantissa f.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

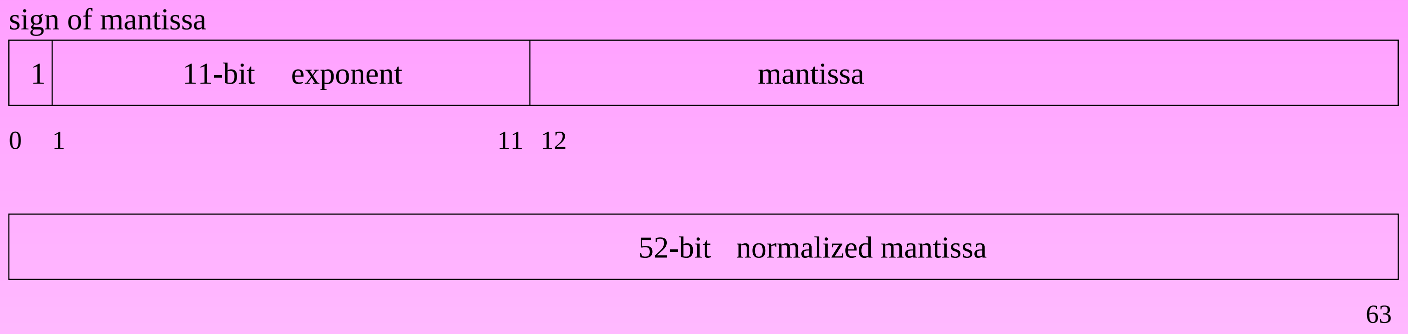

Figure 2: 64-bit double precision.

- The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent c and a 52-bit mantissa f.
- The actual exponent is $c - 1023$.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

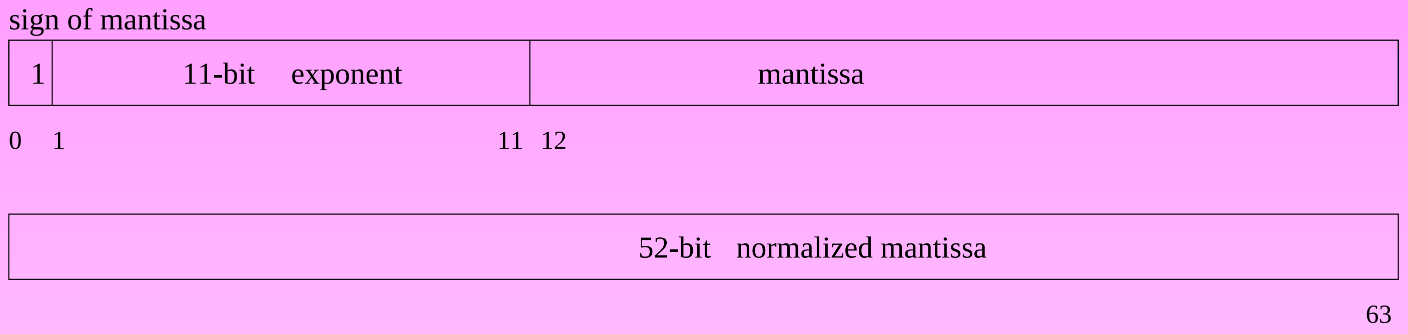

Figure 2: 64-bit double precision.

- The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent c and a 52-bit mantissa f.
- The actual exponent is $c - 1023$.
- The machine epsilon

$$\varepsilon_M = 2^{-52} \approx 2.220 \times 10^{-16},$$

which provides between 15 and 16 decimal digits of accuracy.

Computer Arithmetic

☞ A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.

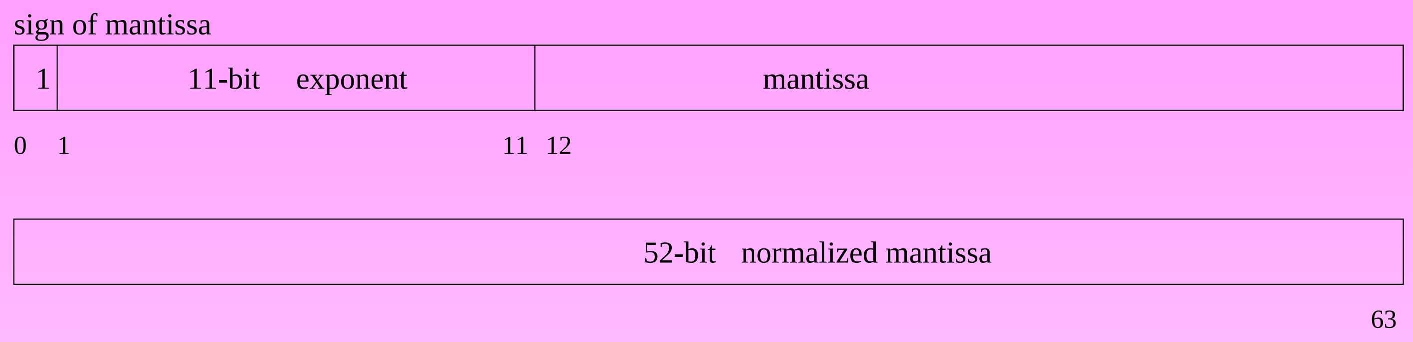

Figure 2: 64-bit double precision.

- The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent c and a 52-bit mantissa f.
- The actual exponent is $c - 1023$.
- The machine epsilon

$$\varepsilon_M = 2^{-52} \approx 2.220 \times 10^{-16},$$

which provides between 15 and 16 decimal digits of accuracy.

- Range of approximately $2^{-1022} \approx 2.225 \times 10^{-308}$ to $2^{1024} \approx 1.798 \times 10^{308}$.

Computer Arithmetic

- Table 1 summarizes some characteristics of IEEE standard floating-point representations.

	single precision	double precision
ε_M	$2^{-23} \approx 1.192 \times 10^{-7}$	$2^{-52} \approx 2.220 \times 10^{-16}$
smallest positive number	$2^{-126} \approx 1.175 \times 10^{-38}$	$2^{-1022} \approx 2.225 \times 10^{-308}$
largest number	$2^{128} \approx 3.403 \times 10^{38}$	$2^{1024} \approx 1.798 \times 10^{308}$
decimal precision	6	15

Table 1: Some characteristics of IEEE standard floating-point numbers

Computer Arithmetic

- Table 1 summarizes some characteristics of IEEE standard floating-point representations.

	single precision	double precision
ε_M	$2^{-23} \approx 1.192 \times 10^{-7}$	$2^{-52} \approx 2.220 \times 10^{-16}$
smallest positive number	$2^{-126} \approx 1.175 \times 10^{-38}$	$2^{-1022} \approx 2.225 \times 10^{-308}$
largest number	$2^{128} \approx 3.403 \times 10^{38}$	$2^{1024} \approx 1.798 \times 10^{308}$
decimal precision	6	15

Table 1: Some characteristics of IEEE standard floating-point numbers

- For the most accuracy, computations should be done using double precision floating-point numbers, however, the execution time is much higher.

Computer Arithmetic

- Table 1 summarizes some characteristics of IEEE standard floating-point representations.

	single precision	double precision
ε_M	$2^{-23} \approx 1.192 \times 10^{-7}$	$2^{-52} \approx 2.220 \times 10^{-16}$
smallest positive number	$2^{-126} \approx 1.175 \times 10^{-38}$	$2^{-1022} \approx 2.225 \times 10^{-308}$
largest number	$2^{128} \approx 3.403 \times 10^{38}$	$2^{1024} \approx 1.798 \times 10^{308}$
decimal precision	6	15

Table 1: Some characteristics of IEEE standard floating-point numbers

- For the most accuracy, computations should be done using double precision floating-point numbers, however, the execution time is much higher.

- If a number $x = \pm q \times 2^m$ with m outside the computer's possible range (too large or too small), then we say that an *overflow* or an *underflow* has occurred.

Computer Arithmetic

- $+$Inf and $-$Inf correspond to two quite different numbers, $+\infty$ and $-\infty$. A NaN stands for Not a Number and is an error pattern rather than a number. Table 2 lists the IEEE exception handling standard.

big*big	$\pm$ Inf	overflow
number/0.0	$\pm$ Inf	division
0.0/0.0	NaN	invalid
small/big	subnormal number	underflow
2.0/3.0	rounded	

Table 2: IEEE exception handling.

2 Floating-Point Error Analysis

2 Floating-Point Error Analysis

- Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$.

2 Floating-Point Error Analysis

- Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$.
- Whenever two machine numbers x and y are to be combined arithmetically, the computer will produce $fl(x \odot y)$ instead of $x \odot y$.

2 Floating-Point Error Analysis

- Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$.
- Whenever two machine numbers x and y are to be combined arithmetically, the computer will produce $fl(x \odot y)$ instead of $x \odot y$.
- Under (1), the relative error of $fl(x \odot y)$ satisfies

$$fl(x \odot y) = (x \odot y)(1 + \delta), \quad \delta \leq \varepsilon_M, \tag{3}$$

where ε_M is the unit roundoff.

2 Floating-Point Error Analysis

- Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$.
- Whenever two machine numbers x and y are to be combined arithmetically, the computer will produce $fl(x \odot y)$ instead of $x \odot y$.
- Under (1), the relative error of $fl(x \odot y)$ satisfies

$$fl(x \odot y) = (x \odot y)(1 + \delta), \quad \delta \leq \varepsilon_M, \tag{3}$$

where ε_M is the unit roundoff.

- But if x, y are not machine numbers, then they must first rounded to floating-point format before the arithmetic operation and the resulting relative error becomes

$$fl(fl(x) \odot fl(y)) = (x(1 + \delta_1) \odot y(1 + \delta_2))(1 + \delta_3),$$

where $\delta_i \leq \varepsilon_M, i = 1, 2, 3.$

Computer Arithmetic

- The analysis (3) can be extended to arithmetic operations on three floating-point numbers.

Computer Arithmetic

- The analysis (3) can be extended to arithmetic operations on three floating-point numbers. For example,

$$\begin{aligned} fl(x(y+z)) &= (x \cdot fl(y+z))(1+\delta_1) \\ &= (x(y+z)(1+\delta_2))(1+\delta_1) \\ &= x(y+z)(1+\delta_1+\delta_2+\delta_1\delta_2) \\ &\approx x(y+z)(1+\delta_1+\delta_2) \\ &= x(y+z)(1+\delta_3) \end{aligned}$$

3 Loss of Significance

☞ One of the most common error-producing calculations involves the cancellation of significant digits due to

3 Loss of Significance

☞ One of the most common error-producing calculations involves the cancellation of significant digits due to the subtraction of nearly equal numbers

3 Loss of Significance

☞ One of the most common error-producing calculations involves the cancellation of significant digits due to the subtraction of nearly equal numbers (or the addition of one very large number and one very small number).

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1d_2 \cdots d_p\alpha_{p+1}\alpha_{p+2} \cdots \alpha_t \times 10^n,$$

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1d_2\cdots d_p\alpha_{p+1}\alpha_{p+2}\cdots\alpha_t \times 10^n,$$

and

$$fl(y) = 0.d_1d_2\cdots d_p\beta_{p+1}\beta_{p+2}\cdots\beta_t \times 10^n.$$

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1d_2\cdots d_p\alpha_{p+1}\alpha_{p+2}\cdots\alpha_t \times 10^n,$$

and

$$fl(y) = 0.d_1d_2\cdots d_p\beta_{p+1}\beta_{p+2}\cdots\beta_t \times 10^n.$$

Then the floating-point form of $x - y$ is

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1}\sigma_{p+2}\cdots\sigma_t \times 10^n,$$

where

$$0.\sigma_{p+1}\sigma_{p+2}\cdots\sigma_t = 0.\alpha_{p+1}\alpha_{p+2}\cdots\alpha_t - 0.\beta_{p+1}\beta_{p+2}\cdots\beta_t.$$

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1 d_2 \cdots d_p \alpha_{p+1} \alpha_{p+2} \cdots \alpha_t \times 10^n,$$

and

$$fl(y) = 0.d_1 d_2 \cdots d_p \beta_{p+1} \beta_{p+2} \cdots \beta_t \times 10^n.$$

Then the floating-point form of $x - y$ is

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t \times 10^n,$$

where

$$0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t = 0.\alpha_{p+1} \alpha_{p+2} \cdots \alpha_t - 0.\beta_{p+1} \beta_{p+2} \cdots \beta_t.$$

- The floating-point number used to represent $x - y$ has at most $t - p$ digits of significance.

Computer Arithmetic

- Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1 d_2 \cdots d_p \alpha_{p+1} \alpha_{p+2} \cdots \alpha_t \times 10^n,$$

and

$$fl(y) = 0.d_1 d_2 \cdots d_p \beta_{p+1} \beta_{p+2} \cdots \beta_t \times 10^n.$$

Then the floating-point form of $x - y$ is

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t \times 10^n,$$

where

$$0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t = 0.\alpha_{p+1} \alpha_{p+2} \cdots \alpha_t - 0.\beta_{p+1} \beta_{p+2} \cdots \beta_t.$$

- The floating-point number used to represent $x - y$ has at most $t - p$ digits of significance. However, in most computers, $x - y$ will be assigned t digits, with the last p digits being either zero or randomly assigned.

Computer Arithmetic

Example 3.1 *If* $x = 0.3721478693$ *and* $y = 0.3720230572$*, what is the relative error in the computation of* $x - y$ *using five decimal digits of accuracy?*

Solution:

Computer Arithmetic

Example 3.1 *If* $x = 0.3721478693$ *and* $y = 0.3720230572$*, what is the relative error in the computation of* $x - y$ *using five decimal digits of accuracy?*

Solution: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

Computer Arithmetic

Example 3.1 *If $x = 0.3721478693$ and $y = 0.3720230572$, what is the relative error in the computation of $x - y$ using five decimal digits of accuracy?*

Solution: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

But both x and y will be rounded to five decimal digits before subtraction.

Computer Arithmetic

Example 3.1 *If $x = 0.3721478693$ and $y = 0.3720230572$, what is the relative error in the computation of $x - y$ using five decimal digits of accuracy?*

Solution: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

But both x and y will be rounded to five decimal digits before subtraction. Thus

$$\begin{aligned} fl(x) &= 0.37215 \\ fl(y) &= 0.37202 \\ fl(x) - fl(y) &= 0.00013 = 0.13000 \times 10^{-3} \end{aligned}$$

Computer Arithmetic

Example 3.1 *If $x = 0.3721478693$ and $y = 0.3720230572$, what is the relative error in the computation of $x - y$ using five decimal digits of accuracy?*

Solution: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

But both x and y will be rounded to five decimal digits before subtraction. Thus

$$\begin{aligned} fl(x) &= 0.37215 \\ fl(y) &= 0.37202 \\ fl(x) - fl(y) &= 0.00013 = 0.13000 \times 10^{-3} \end{aligned}$$

Therefore the relative error is

$$\frac{(x-y) - (fl(x) - fl(y))}{x - y} \approx 0.04 = 4\%.$$

■

Computer Arithmetic

☞ How many significant binary bits are lost in the subtraction when x is close to y?

Computer Arithmetic

☞ How many significant binary bits are lost in the subtraction when x is close to y?

Theorem 3.1 *If $x \geq 0$ and $y \geq 0$ are normalized floating-point binary 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 digits are lost in the subtraction $x - y$.

Computer Arithmetic

Proof: Write

$$x = r \times 2^n, \ \frac{1}{2} \le r < 1 \quad \text{and} \quad y = s \times 2^m, \ \frac{1}{2} \le s < 1.$$

Since $x > y$, we must shift the decimal digits of y to the right

$$y = (s \times 2^{m-n}) \times 2^n.$$

Then

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

By assumption $2^{-q} \le 1 - \frac{y}{x} \le 2^{-p}$, hence

$$r\left(1 - \frac{y}{x}\right) < 1 \cdot 2^{-p} = 2^{-p}.$$

This means that to normalize the result $x - y$, a shift of at least p bits to the left is required.

Similarly,

$$r\left(1 - \frac{y}{x}\right) \ge \frac{1}{2} \cdot 2^{-q} = 2^{-(q+1)},$$

and a shift of at most q bits to the right is required. ■

Computer Arithmetic

☞ Sometimes, loss of significance can be avoided by rewriting the mathematical formula.

Computer Arithmetic

☞ Sometimes, loss of significance can be avoided by rewriting the mathematical formula.

Example 3.2 *Consider the two equivalent functions*

$$f(x) = x(\sqrt{x+1} - \sqrt{x}) \quad \textit{and} \quad g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Compare the function evaluation of $f(500)$ *and* $g(500)$ *using 6 digits and rounding.*

Computer Arithmetic

☞ Sometimes, loss of significance can be avoided by rewriting the mathematical formula.

Example 3.2 *Consider the two equivalent functions*

$$f(x) = x(\sqrt{x+1} - \sqrt{x}) \quad \textit{and} \quad g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Compare the function evaluation of $f(500)$ and $g(500)$ using 6 digits and rounding.

Solution:

$$\begin{aligned} f(500) &= 0.500000 \times 10^3 \times (\sqrt{501} - \sqrt{500}) \\ &= 0.500000 \times 10^3 \times (0.223830 \times 10^2 - 0.223607 \times 10^2) \\ &= 0.500000 \times 10^3 \times 0.223000 \\ &= 0.111500 \times 10^3 \end{aligned}$$

Computer Arithmetic

and

Computer Arithmetic

and

$$\begin{aligned} g(500) &= \frac{500}{\sqrt{501}+\sqrt{500}} \\ &= \frac{0.500000\times 10^3}{0.223830\times 10^2+0.223607\times 10^2} \\ &= \frac{0.500000\times 10^3}{0.447437\times 10^2} \\ &= 0.111748\times 10^2 \end{aligned}$$

Computer Arithmetic

and

$$\begin{aligned} g(500) &= \frac{500}{\sqrt{501}+\sqrt{500}} \\ &= \frac{0.500000\times 10^3}{0.223830\times 10^2 + 0.223607\times 10^2} \\ &= \frac{0.500000\times 10^3}{0.447437\times 10^2} \\ &= 0.111748\times 10^2 \end{aligned}$$

If more digits are used, we can calculated

$$\begin{aligned} f(500) &= 500\times(\sqrt{501}-\sqrt{500}) \\ &= 500\times(22.38302929-22.36067977) \\ &= 500\times 0.022349516 \\ &= 11.1747553 \end{aligned}$$

Hence it can be argued that the formulation $g(x)$ is better. ■

Computer Arithmetic

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$*, when* $a \neq 0$*, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$*, when* $a \neq 0$*, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \textit{and} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ *and discuss the numerical results.*

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$*, when* $a \neq 0$*, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ *and discuss the numerical results.*

Solution: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain

$$x_1 = -0.01610723 \quad \text{and} \quad x_2 = -62.08390.$$

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$, *when* $a \neq 0$, *are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ *and discuss the numerical results.*

Solution: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain

$$x_1 = -0.01610723 \quad \text{and} \quad x_2 = -62.08390.$$

Now we perform the calculations with 4-digit rounding arithmetic.

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$, *when* $a \neq 0$, *are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ *and discuss the numerical results.*

Solution: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain

$$x_1 = -0.01610723 \quad \text{and} \quad x_2 = -62.08390.$$

Now we perform the calculations with 4-digit rounding arithmetic. First we have

$$\sqrt{b^2 - 4ac} = \sqrt{62.10^2 - 4.000} = \sqrt{3856 - 4.000} = \sqrt{3852} = 62.06,$$

and

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000.$$

Computer Arithmetic

Example 3.3 *The quadratic formulas for computing the roots of* $ax^2 + bx + c = 0$*, when* $a \neq 0$*, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \textit{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ *and discuss the numerical results.*

Solution: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain

$$x_1 = -0.01610723 \quad \text{and} \quad x_2 = -62.08390.$$

Now we perform the calculations with 4-digit rounding arithmetic. First we have

$$\sqrt{b^2 - 4ac} = \sqrt{62.10^2 - 4.000} = \sqrt{3856 - 4.000} = \sqrt{3852} = 62.06,$$

and

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000.$$

The relative error in computing x_1 is

$$\frac{|fl(x_1) - x_1|}{|x_1|} = \frac{|-0.02000 + 0.01610723|}{|-0.01610723|} = \frac{0.00389277}{0.01610723} \approx 0.2417.$$

Computer Arithmetic

Computer Arithmetic

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

Computer Arithmetic

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

Computer Arithmetic

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

In this equation, $b^2 = 62.10^2$ is much larger than $4ac = 4$. Hence b and $\sqrt{b^2 - 4ac}$ become two nearly equal numbers. The calculation of x_1 involves the subtraction of two nearly equal numbers.

Computer Arithmetic

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

In this equation, $b^2 = 62.10^2$ is much larger than $4ac = 4$. Hence b and $\sqrt{b^2 - 4ac}$ become two nearly equal numbers. The calculation of x_1 involves the subtraction of two nearly equal numbers.

To obtain a more accurate 4-digit rounding approximation for x_1, we change the formulation by rationalizing the numerator,

Computer Arithmetic

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

In this equation, $b^2 = 62.10^2$ is much larger than $4ac = 4$. Hence b and $\sqrt{b^2 - 4ac}$ become two nearly equal numbers. The calculation of x_1 involves the subtraction of two nearly equal numbers.

To obtain a more accurate 4-digit rounding approximation for x_1, we change the formulation by rationalizing the numerator, that is,

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

Computer Arithmetic

Computer Arithmetic

Then

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610.$$

The relative error in computing x_1 is now reduced to 0.62×10^{-3}. ■

Computer Arithmetic

Then

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610.$$

The relative error in computing x_1 is now reduced to 0.62×10^{-3}. ■

Example 3.4 *Let*

$$\begin{aligned} p(x) &= ((x^3 - 3x^2) + 3x) - 1, \\ q(x) &= ((x - 3)x + 3)x - 1. \end{aligned}$$

Compare the function values at $x = 2.19$.

Computer Arithmetic

Then

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610.$$

The relative error in computing x_1 is now reduced to 0.62×10^{-3}. ■

Example 3.4 *Let*

$$\begin{aligned} p(x) &= ((x^3 - 3x^2) + 3x) - 1, \\ q(x) &= ((x - 3)x + 3)x - 1. \end{aligned}$$

Compare the function values at $x = 2.19$*.*

Solution: Use 3-digit and rounding for $p(2.19)$ and $q(2.19)$.

$$\begin{aligned} \hat{p}(2.19) &= ((2.19^3 - 3 \times 2.19^2) + 3 \times 2.19) - 1 \\ &= ((10.5 - 14.4) + 3 \times 2.19) - 1 \\ &= (-3.9 + 6.57) - 1 \\ &= 2.67 - 1 \\ &= 1.67 \end{aligned}$$

Computer Arithmetic

Computer Arithmetic

$$\begin{aligned}
\hat{q}(2.19) &= ((2.19-3)\times 2.19+3)\times 2.19-1 \\
&= (-0.81\times 2.19+3)\times 2.19-1 \\
&= (-1.77+3)\times 2.19-1 \\
&= 1.23\times 2.19-1 \\
&= 2.69-1 \\
&= 1.69
\end{aligned}$$

Computer Arithmetic

$$\begin{aligned}
\hat{q}(2.19) &= ((2.19-3)\times 2.19+3)\times 2.19-1 \\
&= (-0.81\times 2.19+3)\times 2.19-1 \\
&= (-1.77+3)\times 2.19-1 \\
&= 1.23\times 2.19-1 \\
&= 2.69-1 \\
&= 1.69
\end{aligned}$$

With more digits, one can have

$$p(2.19)=g(2.19)=1.685159$$

Computer Arithmetic

$$\begin{aligned}\hat{q}(2.19) &= ((2.19-3)\times 2.19+3)\times 2.19-1\\ &= (-0.81\times 2.19+3)\times 2.19-1\\ &= (-1.77+3)\times 2.19-1\\ &= 1.23\times 2.19-1\\ &= 2.69-1\\ &= 1.69\end{aligned}$$

With more digits, one can have

$$p(2.19)=g(2.19)=1.685159$$

Hence the absolute errors are

$$|p(2.19)-\hat{p}(2.19)|=0.015159$$

and

$$|q(2.19)-\hat{q}(2.19)|=0.004841,$$

respectively.

$$\begin{aligned}
\hat{q}(2.19) &= ((2.19-3)\times 2.19+3)\times 2.19-1 \\
&= (-0.81\times 2.19+3)\times 2.19-1 \\
&= (-1.77+3)\times 2.19-1 \\
&= 1.23\times 2.19-1 \\
&= 2.69-1 \\
&= 1.69
\end{aligned}$$

With more digits, one can have

$$p(2.19)=g(2.19)=1.685159$$

Hence the absolute errors are

$$|p(2.19)-\hat{p}(2.19)|=0.015159$$

and

$$|q(2.19)-\hat{q}(2.19)|=0.004841,$$

respectively. One can observe that the evaluation formula $q(x)$ is better than $p(x)$. ■

Computer Arithmetic

Computer Arithmetic

Example 3.5 *How to evaluate*

$$y = x - \sin x$$

when x is small?

Computer Arithmetic

Example 3.5 *How to evaluate*

$$y = x - \sin x$$

when x is small?

Solution: Since $x \approx \sin x$ for small x, the computation will cause loss of significance.

Computer Arithmetic

Example 3.5 *How to evaluate*

$$y = x - \sin x$$

when x is small?

Solution: Since $x \approx \sin x$ for small x, the computation will cause loss of significance. Alternatively, use Taylor series for $\sin x$ so that

$$\begin{aligned}
y &= x - \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots \right) \\
&= \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \frac{x^9}{9!} + \cdots \\
&= \frac{x^3}{6} - \frac{x^5}{6 \times 20} + \frac{x^7}{6 \times 20 \times 42} - \frac{x^9}{6 \times 20 \times 42 \times 72} \cdots \\
&= \frac{x^3}{6} \left(1 - \frac{x^2}{20} \left(1 - \frac{x^2}{42} \left(1 - \frac{x^2}{72} (\cdots) \right) \right) \right)
\end{aligned}$$

■

4 Stability and Conditioning

4.1 Numerical Stability

Computer Arithmetic

4 Stability and Conditioning

4.1 Numerical Stability

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

4 Stability and Conditioning

4.1 Numerical Stability

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

Example 4.1 *Consider the following recurrence algorithm*

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

for computing the sequence of $\{x_n = (\frac{1}{3})^n\}$. *This algorithm is unstable.*

Computer Arithmetic

Solution: A computer implementation of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

Computer Arithmetic

Solution: A computer implementation of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

The error present in x_n is multiplied by $\frac{13}{3}$ in computing x_{n+1}.

Computer Arithmetic

Solution: A computer implementation of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

The error present in x_n is multiplied by $\frac{13}{3}$ in computing x_{n+1}. For example, the error will be propagated with a factor of $\left(\frac{13}{3}\right)^{14}$ in computing x_{15}.

Computer Arithmetic

Solution: A computer implementation of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

The error present in x_n is multiplied by $\frac{13}{3}$ in computing x_{n+1}. For example, the error will be propagated with a factor of $\left(\frac{13}{3}\right)^{14}$ in computing x_{15}. Additional roundoff errors in computing $x_2, x_3, \ldots$ may also be propagated and added to that of x_{15}. ■

Computer Arithmetic

4.2 Conditioning

4.2 Conditioning

☞ A problem is ill-conditioned if small changes in the data can produce large changes in the results.

4.2 Conditioning

☞ A problem is ill-conditioned if small changes in the data can produce large changes in the results.

☞ For a nonsingular square matrix A, the condition number of A is defined as

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

with respect to some matrix norm.

4.2 Conditioning

☞ A problem is ill-conditioned if small changes in the data can produce large changes in the results.

☞ For a nonsingular square matrix A, the condition number of A is defined as

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

with respect to some matrix norm.

☞ For a general rectangular matrix, the singular values are used to characterize the condition number

$$\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}},$$

where σ_{max} is the largest singular value of A and σ_{min} the smallest singular value.

Computer Arithmetic

Computer Arithmetic

☞ A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest.

Computer Arithmetic

- ☞ A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest.
- ☞ A well-known ill-conditioned matrix is the Hilbert matrix

$$H_n = [h_{ij}] \in \mathbb{R}^{n\times n}, \qquad \text{where} \qquad h_{ij} = \frac{1}{i+j-1}.$$

Computer Arithmetic

☞ A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest.

☞ A well-known ill-conditioned matrix is the Hilbert matrix

$$H_n = [h_{ij}] \in \mathbb{R}^{n\times n}, \qquad \text{where} \qquad h_{ij} = \frac{1}{i+j-1}.$$

☞ In general, ill-conditioning is not easy to detect.

Computer Arithmetic

☞ A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest.

☞ A well-known ill-conditioned matrix is the Hilbert matrix

$$H_n = [h_{ij}] \in \mathbb{R}^{n\times n}, \qquad \text{where} \qquad h_{ij} = \frac{1}{i+j-1}.$$

☞ In general, ill-conditioning is not easy to detect.

☞ In solving a system of linear equations $Ax = b$ in which A is ill-conditioned, small perturbation in b will cause large perturbation in x.