Numerical Analysis

Prof.dr. hab. Bostan Viorel

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

For example

$$345.78 = (+1) \cdot 3.4578 \cdot 10^2,$$

Decimal floating point numbers

Floating point notation is similar to what is called scientific notation.

For a nonzero number x, we can write it in the form

$$x = \sigma \cdot \overline{x} \cdot 10^e,$$

where $\sigma = \pm 1$, e is an integer, and $1_{10} \leq \overline{x} < 10_{10}$.

Number σ is called **sign**, e is **exponent**, and $\overline{x}$ is called **mantissa** or **significand**.

For example

$$345.78 = (+1) \cdot 3.4578 \cdot 10^2,$$

On a decimal computer or calculator, we store x by instead storing σ, e and $\overline{x}$.

Decimal floating point numbers

Computers have a finite space for storage.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

The number of digits in $\overline{x}$ is called **precision**.

Decimal floating point numbers

Computers have a finite space for storage.

We must restrict the number of digits in $\overline{x}$ and the size of the exponent e.

The number of digits in $\overline{x}$ is called **precision**.

For example on calculator HP-15, the number of digits in mantissa is 10 and

$$-99 \leq e \leq 99$$

Binary floating point numbers

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

$$(0.000110011001100\ldots)_2 = (+1) \cdot (1.10011001100\ldots)_2 \cdot 2^{-(100)_2}$$

Binary floating point numbers

Write

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

with $1_2 \leq \overline{x} < (10)_2 = (2)_{10}$ and e an integer.

For example,

$$(10011.01101)_2 = (+1) \cdot (1.001101101)_2 \cdot 2^{(100)_2}$$

Or another example for $(0.1)_{10} = (0.000110011001100\ldots)_2$

$$(0.000110011001100\ldots)_2 = (+1) \cdot (1.10011001100\ldots)_2 \cdot 2^{-(100)_2}$$

Notice, that the first digit in mantissa is always 1.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots)$$

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter $10-$digit number will fit.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter 10−digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter $10-$digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

On a binary computer, we use a similar notation.

Floating point numbers

When a number x outside a computer or calculator is converted into a machine number, we denote it by $fl(x)$.

On a calculator with precission 10,

$$fl\left(\frac{1}{3}\right) = fl(0.3333333\ldots) = (+1)\cdot(3.333333333)_{10}\cdot 10^{-1}$$

The decimal fraction of finite length will not fit in the registers of the calculator, but the latter $10-$digit number will fit.

Some calculators actually carry more digits internally than they allow to be displayed.

On a binary computer, we use a similar notation.

We will concentrate on a particular form of computer floating point number, that is called the **IEEE floating point standard**.

Binary floating point numbers

Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

Binary floating point numbers

Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

All the numbers admitted by this representation are presented in the table:

Binary floating point numbers

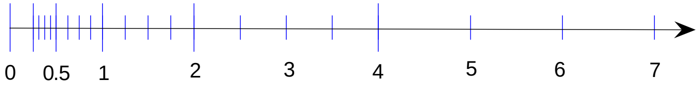
Example 1 Consider a binary floating point representation with precision 3 and $e_{min} = -2 \leq e \leq 2 = e_{max}$

All the numbers admitted by this representation are presented in the table:

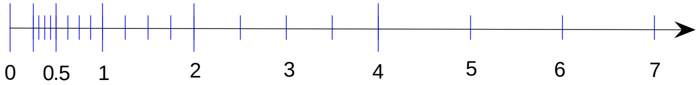
		e				
		-2	-1	0	1	2
$\overline{x}$	$(1.00)_2$	$(0.25)_{10}$	$(0.5)_{10}$	$(1)_{10}$	$(2)_{10}$	$(4)_{10}$
	$(1.01)_2$	$(0.3125)_{10}$	$(0.625)_{10}$	$(1.25)_{10}$	$(2.5)_{10}$	$(5)_{10}$
	$(1.10)_2$	$(0.375)_{10}$	$(0.75)_{10}$	$(1.5)_{10}$	$(3)_{10}$	$(6)_{10}$
	$(1.11)_2$	$(0.4375)_{10}$	$(0.875)_{10}$	$(1.75)_{10}$	$(3.5)_{10}$	$(7)_{10}$

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

Binary floating point numbers

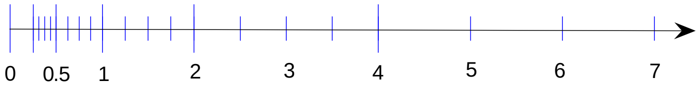


Binary floating point numbers



This representation can be extended to include smaller numbers called **denormalized** numbers.

Binary floating point numbers



This representation can be extended to include smaller numbers called **denormalized** numbers.

These numbers are obtained if $e = e_{min}$ and the first digit of the significand is 0.

Binary floating point numbers

Example 2 Previous example plus denormalized numbers

Binary floating point numbers

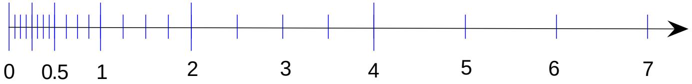
Example 2 Previous example plus denormalized numbers

$$(0.01)_2 \cdot 2^{-1} = \frac{1}{16} = (0.0625)_{10}$$
$$(0.10)_2 \cdot 2^{-1} = \frac{2}{16} = (0.125)_{10}$$
$$(0.11)_2 \cdot 2^{-1} = \frac{3}{16} = (0.1875)_{10}$$

Binary floating point numbers

Example 2 Previous example plus denormalized numbers

$$(0.01)_2 \cdot 2^{-1} = \frac{1}{16} = (0.0625)_{10}$$
$$(0.10)_2 \cdot 2^{-1} = \frac{2}{16} = (0.125)_{10}$$
$$(0.11)_2 \cdot 2^{-1} = \frac{3}{16} = (0.1875)_{10}$$



IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \dots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit,

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \dots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand $\overline{x}$ as 24 bits (only 23 need be stored),

IEEE single precision floating point representation

In IEEE (Institute of Electrical and Electronics Engineers) single precision standard 32 bits are used to store numbers.

IEEE single precision has in mantissa 24 binary digits,
exponent e has values between -126 and 127,
or in binary $-(1111110)_2 \leq e \leq (1111111)_2$.

Thus number x has the representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{23} \cdot 2^e.$$

Basically, we store σ as a single bit, the significand $\overline{x}$ as 24 bits (only 23 need be stored), and the exponent fills out 8 bits, including both negative and positive integers.

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

σ	E			$\bar{x}$		
b_1	b_2	$\ldots$	b_9	b_{10}	$\ldots$	b_{32}

IEEE single precision floating point representation

In order to avoid the sign for exponent, denote

$$E = e + 127$$

Obviously, $1 \leq E \leq 254$ with two additional values 0 and 255.

σ	E			$\bar{x}$		
b_1	b_2	$\ldots$	b_9	b_{10}	$\ldots$	b_{32}

Number $x = 0$ is stored in the following way: $E = 0$, $\sigma = 0$ and $b_{10}b_{11}\ldots b_{32} = (00\ldots0)_2$.

IEEE single precision floating point representation

$E = (b_2 \dots b_9)_2$	e	x
$(00000000)_2 = (0)_{10}$	$-(127)_{10}$	$\pm(0.b_{10} \dots b_{32})_2 \cdot 2^{-126}$
$(00000001)_2 = (1)_{10}$	$-(126)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{-126}$
$(00000010)_2 = (2)_{10}$	$-(125)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{-125}$
$\vdots$	$\vdots$	$\vdots$
$(01111111)_2 = (127)_{10}$	$(0)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{0}$
$(10000000)_2 = (128)_{10}$	$(1)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{1}$
$\vdots$	$\vdots$	$\vdots$
$(11111101)_2 = (253)_{10}$	$(126)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{126}$
$(11111110)_2 = (254)_{10}$	$(127)_{10}$	$\pm(1.b_{10} \dots b_{32})_2 \cdot 2^{127}$
$(11111111)_2 = (255)_{10}$	$(128)_{10}$	$\pm\infty$, *dacă* $b_i = 0$ NaN, *otherwise*

IEEE double precision floating point representation

$$x = \sigma \cdot 1.a_1 a_2 \ldots a_{52} \cdot 2^e.$$

with $E = e + 1023$

σ	E			$\bar{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

IEEE double precision floating point representation

$E = (b_2 \dots b_{12})_2$	e	x
$(00000000000)_2 = (0)_{10}$	$-(1023)_{10}$	$\pm(0.b_{13}\dots b_{64})2^{-1022}$
$(00000000001)_2 = (1)_{10}$	$-(1022)_{10}$	$\pm(1.b_{13}\dots b_{64})2^{-1022}$
$(00000000010)_2 = (2)_{10}$	$-(1021)_{10}$	$\pm(1.b_{13}\dots b_{64})2^{-1021}$
$\vdots$	$\vdots$	$\vdots$
$(01111111111)_2 = (1023)_{10}$	$(0)_{10}$	$\pm(1.b_{13}\dots b_{64})2^{0}$
$(10000000000)_2 = (1024)_{10}$	$(1)_{10}$	$\pm(1.b_{13}\dots b_{64})2^{1}$
$\vdots$	$\vdots$	$\vdots$
$(11111111101)_2 = (2045)_{10}$	$(1022)_{10}$	$\pm(1.b_{13}\dots b_{64})2^{1022}$
$(11111111110)_2 = (2046)_{10}$	$(1023)_{10}$	$\pm(1.b_{13}\dots b_{64})_2^{1023}$
$(11111111111)_2 = (2047)_{10}$	$(1024)_{10}$	$\pm\infty$, $b_i = 0$ NaN, *otherwise*

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$(\underbrace{111\ldots11}_{\text{23 ones}})_2$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$(\underbrace{111\ldots11}_{\text{23 ones}})_2 \quad = \quad (1.11\ldots1)_2 \cdot 2^{23}$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$
\begin{aligned}
(\underbrace{111\ldots11}_{23 \text{ ones}})_2 &= (1.11\ldots1)_2 \cdot 2^{23} \\
&= 2^{23} + 2^{22} + \ldots + 2^0
\end{aligned}
$$

Caracteristics of floating point representation

What is the connection of the 24 bits in the significand $\overline{x}$ to the number of decimal digits in the storage of a number x into floating point form?

One way of answering this is to find the integer M for which

1. $0 < x \leq M$ and x an integer implies $fl(x) = x$
2. $fl(M+1) \neq M+1$

This integer M is at least as big as

$$\begin{aligned}
(\underbrace{111\ldots11}_{\text{23 ones}})_2 &= (1.11\ldots1)_2 \cdot 2^{23} \\
&= 2^{23} + 2^{22} + \ldots + 2^{0} \\
&= 2^{24} - 1
\end{aligned}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24}+1$ cannot be stored exactly since its significand will contain $24+1$ binary digits:

$$2^{24}+1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$.

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24}+1$ cannot be stored exactly since its significand will contain $24+1$ binary digits:

$$2^{24}+1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24}+1$ cannot be stored exactly since its significand will contain $24+1$ binary digits:

$$2^{24}+1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

So $M = 2^{24} = 16777216$

Caracteristics of floating point representation

Also, there is one more number to be stored exactly:

$$2^{24} = (1.00\ldots00)_2 \cdot 2^{24}$$

Next integer $2^{24} + 1$ cannot be stored exactly since its significand will contain $24 + 1$ binary digits:

$$2^{24} + 1 = (1.\underbrace{00\ldots0}_{\text{23 zeros}}1)_2 \cdot 2^{24}$$

Therefore, for single precision $M = 2^{24}$. Any integer less or equal to M will be stored exactly.

So $M = 2^{24} = 16777216$
For double precision $M = 2^{53} \approx 9.0 \cdot 10^{15}$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1?

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

and the machine epsilon in IEEE single precision floating point format is $\eta = 2^{-23}$

Machine epsilon

Let y be the smallest number representable in the machine arithmetic that is greater than 1 in the machine.

The machine epsilon is $\eta = y - 1$.

It is a widely used to measure the accuracy possible in representing numbers in the machine.

The number 1 has the simple floating point representation

$$1 = (1.00\ldots0)_2 \cdot 2^0$$

What is the smallest number that is greater than 1? It is

$$1 + 2^{-23} = (1.0\ldots01)_2 \cdot 2^0 > 1$$

and the machine epsilon in IEEE single precision floating point format is $\eta = 2^{-23} \approx 1.19 \cdot 10^{-7}$

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The size of δ is another way of describing the accuracy attainable in the floating point representation of the machine.

The unit round

Consider the smallest number $\delta > 0$ that is representable in the machine and for which

$$1 + \delta > 1$$

in the arithmetic of the machine.

For any number $0 < \alpha < \delta$ the result of $1 + \alpha$ is exactly 1 in the machines arithmetic.

Thus α drops off the end of the floating point representation in the machine.

The size of δ is another way of describing the accuracy attainable in the floating point representation of the machine.

The machine epsilon has been replacing it in recent years.

Chopping and rounding in decimal

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

This is done by either chopping or rounding.

Chopping and rounding in decimal

We write a computer floating point number z as

$$z = \sigma \cdot \overline{z} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e$$

with $a_1 \neq 0$, so that there are n decimal digits in the significand.

Given a general number x

$$x = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n a_{n+1} \ldots)_{10} \cdot 10^e, \quad a_1 \neq 0.$$

we must shorten it to fit within the computer.

This is done by either chopping or rounding.

The floating point chopped version of x is given by

$$fl(x) = \sigma \cdot \overline{x} \cdot 10^e \equiv \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e,$$

where we assume that e fits within the bounds required by the computer or calculator.

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

A simplified formula is

$$fl(x) = \begin{cases} \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e, & \textit{if} \quad a_{n+1} < 5 \\ \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e + (0.00 \ldots 01)_{10}, & \textit{if} \quad a_{n+1} \geq 5 \end{cases}$$

Chopping and rounding in decimal

For the rounded version, we must decide whether to round up or round down.

A simplified formula is

$$fl(x) = \begin{cases} \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e, & \text{if} \quad a_{n+1} < 5 \\ \sigma \cdot (a_1.a_2 \ldots a_n)_{10} \cdot 10^e + (0.00 \ldots 01)_{10}\,, & \text{if} \quad a_{n+1} \geq 5 \end{cases}$$

The term $(0.00 \ldots 01)_{10}$ denotes 10^{-n+1}, giving the ordinary sense of rounding with which you are familiar.

Chopping and rounding in binary

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \ldots a_n a_{n+1} \ldots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \dots a_n a_{n+1} \dots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Then for a **chopped** floating point representation, we have

$$fl(x) = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \dots a_n)_2 \cdot 2^e.$$

Chopping and rounding in binary

Let

$$x = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \dots a_n a_{n+1} \dots)_2 \cdot 2^e,$$

with all a_i equal to 0 or 1.

Then for a **chopped** floating point representation, we have

$$fl(x) = \sigma \cdot \overline{x} \cdot 2^e \equiv \sigma \cdot (1.a_2 \dots a_n)_2 \cdot 2^e.$$

For a **rounded** floating point representation, we have

$$fl(x) = \begin{cases} \sigma \cdot (1.a_2 \dots a_n)_2 \cdot 2^e, & \textit{if} \quad a_{n+1} = 0 \\ \sigma \cdot (1.a_2 \dots a_n)_2 \cdot 2^e + (0.00 \dots 01)_2\,, & \textit{if} \quad a_{n+1} = 1 \end{cases}$$

Errors in floating point representation

The error $x - fl(x)$

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$,

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping,

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

With $x \neq fl(x)$ and rounding

Errors in floating point representation

The error $x - fl(x) = 0$ when x needs no change to be put into the computer or calculator, in other words x is stored exactly.

Of more interest is the case when the error is nonzero.

Consider first the case $x > 0$ (meaning $\sigma = +1$).

With $x \neq fl(x)$, and using chopping, we have $fl(x) < x$ and the error $x - fl(x)$ is always positive.

This fact has major consequences in extended numerical computations.

With $x \neq fl(x)$ and rounding, the error $x - fl(x)$ is negative for half the values of x, and it is positive for the other half of possible values of x.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$-\frac{1}{2^n} \quad \leq \quad \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding}$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1 + \varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$\begin{aligned} -\frac{1}{2^n} &\leq \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding} \\ -\frac{1}{2^{n-1}} &\leq \varepsilon \leq 0, \quad \textit{chopping} \end{aligned}$$

Errors in floating point representation

We often write the **relative error** as

$$\frac{x - fl(x)}{x} = -\varepsilon$$

This can be expanded to obtain

$$fl(x) = (1+\varepsilon)x$$

Thus $fl(x)$ can be considered as a perturbed value of x.
This formula is used in many analyses of the effects of chopping and rounding errors in numerical computations.

For bounds on ε, we have

$$\begin{aligned} -\frac{1}{2^n} &\leq \varepsilon \leq \frac{1}{2^n}, \quad \textit{rounding} \\ -\frac{1}{2^{n-1}} &\leq \varepsilon \leq 0, \quad \textit{chopping} \end{aligned}$$

There is also an extended representation, having $n = 69$ digits in its significand.

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
>>format hex

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
$>>$format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
$>>$format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
$>>$format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>>$ 401c800000000000

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
$>>$format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>>$ 401c8000000000000

The 16 hexadecimal digits are
$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f\}$

MATLAB floating point representation

MATLAB can be used to generate the binary floating point representation of a number.

Execute in MATLAB the command:
$>>$format hex

This will cause all subsequent numerical output to the screen to be given in hexadecimal format (base 16).

For example, listing the number 7.125 results in an output of
$>>$ 401c800000000000

The 16 hexadecimal digits are
$\{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; a; b; c; d; e; f\}$

To obtain the binary representation, convert each hexadecimal digit to a four digit binary number according to the table on next slide:

MATLAB floating point representation

Format hex	Format binary	Format hex	Format binary
0	0000	8	1000
1	0001	9	1001
2	0010	*a*	1010
3	0011	*b*	1011
4	0100	*c*	1100
5	0101	*d*	1101
6	0110	*e*	1110
7	0111	*f*	1111

MATLAB floating point representation

$$401c800000000000$$

MATLAB floating point representation

$$401c800000000000$$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0001}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

MATLAB floating point representation

$$401c800000000000$$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0001}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

MATLAB floating point representation

$401c800000000000$

$$\overbrace{0100}^{4}\overbrace{0000}^{0}\overbrace{0001}^{1}\overbrace{1100}^{c}\overbrace{1000}^{8}\overbrace{0000}^{0}\overbrace{0000}^{0}\ldots\overbrace{0000}^{0}$$

σ	E			$\overline{x}$		
b_1	b_2	$\ldots$	b_{12}	b_{13}	$\ldots$	b_{64}

$$\underbrace{0}_{\sigma}\ \underbrace{10000000001}_{E}\underbrace{1100100000000000\ldots0000}_{1.b_{13}b_{14}\ldots b_{64}=\overline{x}}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Example:

$$x_T = e, \quad x_A = \frac{19}{7}$$

Errors

Let x_T denote the true value of some number, usually unknown in practice and let x_A denote an approximation of x_T.

The **error** in x_A is

$$err(x_A) = x_T - x_A$$

The **relative error** in x_A is

$$rel(x_A) = \frac{err(x_A)}{x_T} = \frac{x_T - x_A}{x_T}$$

Example:

$$x_T = e, \quad x_A = \frac{19}{7}$$

$$\begin{aligned} err(x_A) &= e - \frac{19}{7} \approx 0.003996 \\ rel(x_A) &\approx \frac{0.003996}{e} \approx 0.00147 \end{aligned}$$

Errors in floating point representation

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$err(D_A) \quad = \quad D_T - D_A = 1km,$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$err(d_A) = d_T - d_A = 1km,$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

In both cases the error is the same.

Errors in floating point representation

Example: Suppose the distance between two cities is $D_T = 100km$ and let this distance be approximated with $D_A = 99km$.

In this case,

$$\begin{aligned} err(D_A) &= D_T - D_A = 1km, \\ rel(D_A) &= \frac{err(D_A)}{D_T} = \frac{1}{100} = 0.01 = 1\% \end{aligned}$$

Now, suppose that distance is $d_T = 2km$ and estimate it with $d_A = 1km$. Then

$$\begin{aligned} err(d_A) &= d_T - d_A = 1km, \\ rel(d_A) &= \frac{err(d_A)}{d_T} = \frac{1}{2} = 0.5 = 50\% \end{aligned}$$

In both cases the error is the same.

But, obviously $D_A$is a better approximation of D_T , then d_A of d_T.

Sources of Error

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile;

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile; and the final term in the equation represents friction force in air.

Sources of Error

The sources of error in the computation of the solution of a mathematical model for some physical situation can be roughly characterised as follows:

1. Modelling Error.

Consider the example of a projectile of mass m that is travelling thrugh the earth's atmosphere. A simple and oftenly used description of projectile motion is given by

$$m\frac{d^2\overrightarrow{r}}{dt^2}(t) = -mg\overrightarrow{k} - b\frac{d\overrightarrow{r}}{dt}$$

with $b \geq 0$. In this, $\overrightarrow{r}(t)$ is the vector position of the projectile; and the final term in the equation represents friction force in air. If there is an error in this a model of a physical situation, then the numerical solution of this equation is not going to improve the results.

Sources of Error

2. **Physical / Observational / Measurement Error**.

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

This error cannot be removed, and it must affect the accuracy of any computation in which it is used.

Sources of Error

2. Physical / Observational / Measurement Error.
The radius of an electron is given by

$$(2.81777 + \varepsilon) \times 10^{-13} cm, \quad |\varepsilon| \leq 0.00011$$

This error cannot be removed, and it must affect the accuracy of any computation in which it is used.

We need to be aware of these effects and to so arrange the computation as to minimize the effects.

Sources of Error

3. **Approximation Error**.

Sources of Error

3. Approximation Error.

This is also called **“discretization error”** and **“truncation error”**;

Sources of Error

3. Approximation Error.

This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course.

Sources of Error

3. Approximation Error.
This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.

Sources of Error

3. Approximation Error.

This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.

For example, the Taylor polynomial approximation

$$e^x \approx 1 + x + \frac{1}{2}x^2$$

contains an "approximation error".

Sources of Error

3. Approximation Error.

This is also called **"discretization error"** and **"truncation error"**; and it is the main source of error with which we deal in this course. Such errors generally occur when we replace a computationally unsolvable problem with a nearby problem that is more tractable computationally.

For example, the Taylor polynomial approximation

$$e^x \approx 1 + x + \frac{1}{2}x^2$$

contains an "approximation error".

The numerical integration

$$\int_0^1 f(x)dx \approx \frac{1}{N}\sum_{j=1}^{N} f\left(\frac{j}{N}\right)$$

contains an approximation error.

Sources of Error

4. Finiteness of Algorithm Error

Sources of Error

4. Finiteness of Algorithm Error

This is an error due to stopping an algorithm after a finite number of iterations.

4. Finiteness of Algorithm Error
This is an error due to stopping an algorithm after a finite number of iterations.

Even if theoretically an algorithm can run for indefinite time, after a finite (usually specified) number of iterations the algorithm will be stopped.

Sources of Error

5. **Blunders.**

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Sources of Error

5. Blunders.

In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.

Some simple rules to decrease the risk of having a bug in the code:

Sources of Error

5. Blunders.
In the pre-computer era, blunders were mostly arithmetic errors. In the earlier years of the computer era, the typical blunder was a programming bug. Present day "blunders" are still often programming errors. But now they are often much more difficult to find, as they are often embedded in very large codes which may mask their effect.
Some simple rules to decrease the risk of having a bug in the code:

- Break programs into small testable subprograms;
- Run test cases for which you know the outcome;
- When running the full code, maintain a skeptical eye on the output, checking whether the output is reasonable or not.

Sources of Error

6. **Rounding/chopping Error**.

Sources of Error

6. Rounding/chopping Error.
This is the main source of many problems, especially problems in solving systems of linear equations.

Sources of Error

6. Rounding/chopping Error.

This is the main source of many problems, especially problems in solving systems of linear equations.

We later look at the effects of such errors.

Sources of Error

7. Finitness of precision errors

Sources of Error

7. Finitness of precision errors

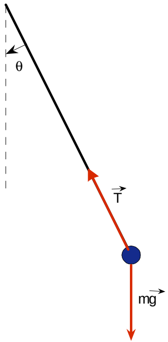
All the numbers stored in computer memory are subject to the finiteness of allocated space for storage.

Pendulum Example

Original problem in engineering or in science to be solved:

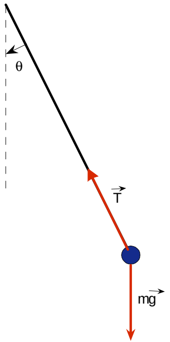
Pendulum Example

Original problem in engineering or in science to be solved:



Pendulum Example

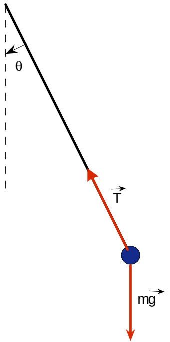
Original problem in engineering or in science to be solved:



Model this physical problem mathematically.

Pendulum Example

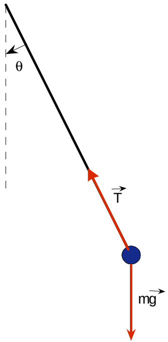
Original problem in engineering or in science to be solved:



Model this physical problem mathematically.
Second Newton law provides us with:

Pendulum Example

Original problem in engineering or in science to be solved:

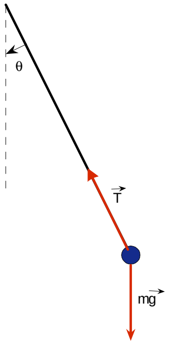


Model this physical problem mathematically.
Second Newton law provides us with:

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

Pendulum Example

Original problem in engineering or in science to be solved:



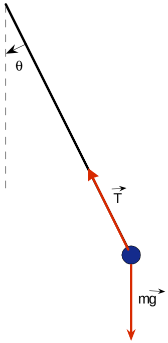
Model this physical problem mathematically.
Second Newton law provides us with:

$$\ddot{\theta} = -\frac{g}{l}\sin\theta$$

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\frac{g}{l}\sin\theta \end{cases}$$

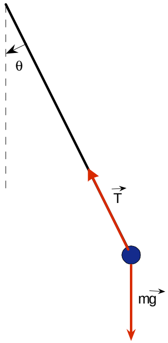
Pendulum Example

Problem of continuous mathematics:



Pendulum Example

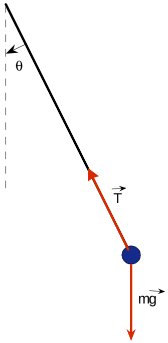
Problem of continuous mathematics:



$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\frac{g}{l}\sin\theta \end{cases}$$

Pendulum Example

Problem of continuous mathematics:

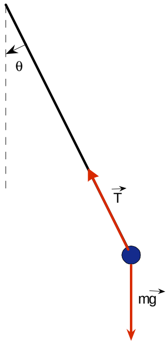


$$\begin{cases} \dot{\theta} &= \omega \\ \dot{\omega} &= -\frac{g}{l} \sin\theta \end{cases}$$

- Modeling Errors
- Physical Errors

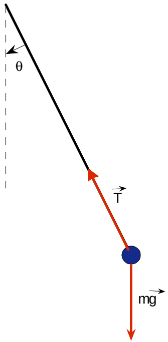
Pendulum Example

Mathematical Algorithms:



Pendulum Example

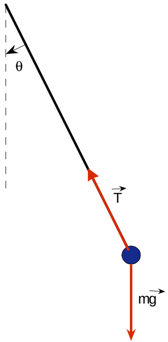
Mathematical Algorithms:



$$\begin{cases} \theta_{n+1} = \theta_n + h\omega_{n+1} \\ \omega_{n+1} = \omega_n - h\frac{g}{l}\sin(\theta_n) \end{cases}$$

Pendulum Example

Mathematical Algorithms:

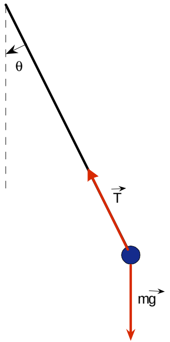


$$\begin{cases} \theta_{n+1} = \theta_n + h\omega_{n+1} \\ \omega_{n+1} = \omega_n - h\frac{g}{l}\sin(\theta_n) \end{cases}$$

- Discretisation Errors
- Finiteness of Algorithm Errors

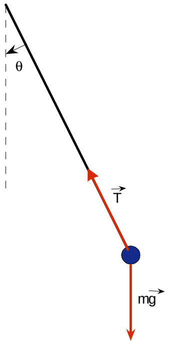
Pendulum Example

Computer Implementation:



Pendulum Example

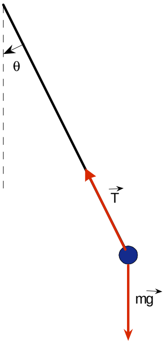
Computer Implementation:




```
for i=1:Nmax
        Omega = Omega - H*g/L*sin(Theta);
        Theta = Theta + H*Omega
end
```

Pendulum Example

Computer Implementation:




```
for i=1:Nmax
        Omega = Omega - H*g/L*sin(Theta);
        Theta = Theta + H*Omega
end
```

- Rounding / Chopping Errors
- Bugs in the Code
- Finite Precision Errors