Errors in computer simulations

Ivo Roghair, Martin van Sint Annaland

Chemical Process Intensification,
Eindhoven University of Technology

Today's outline

Today's outline

Example 1

Start your spreadsheet program (Excel, ...)

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`
A3	`=(A2*10)-0.9`

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`
A3	`=(A2*10)-0.9`
A4	`=(A3*10)-0.9`

(repeat until A30)

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`
A3	`=(A2*10)-0.9`
A4	`=(A3*10)-0.9`

(repeat until A30)

What's happening?

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`
A3	`=(A2*10)-0.9`
A4	`=(A3*10)-0.9`

Enter:

Cell	Value
A1	2

(repeat until A30)

What's happening?

Example 1

Start your spreadsheet program (Excel, ...)

Enter:

Cell	Value
A1	`0.1`
A2	`=(A1*10)-0.9`
A3	`=(A2*10)-0.9`
A4	`=(A3*10)-0.9`

(repeat until A30)

Enter:

Cell	Value
A1	2
A2	`=(A1*10)-18`
A3	`=(A2*10)-18`
A4	`=(A3*10)-18`

(repeat until A30)

What's happening?

Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

Errors in computer simulations

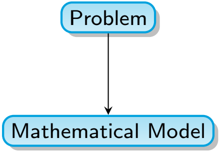
In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)

Verification and validation

Problem

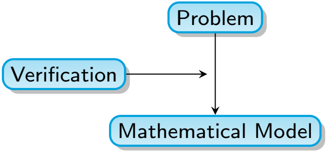
Verification and validation



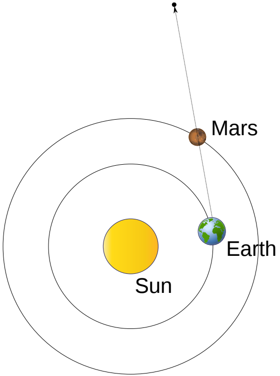
Verification and validation

Verification

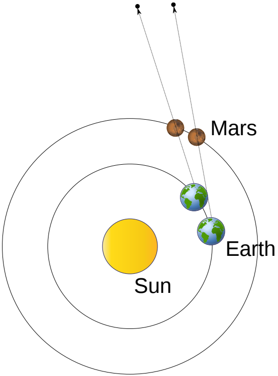
Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



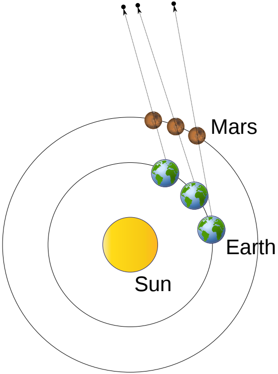
Verification of the physical model



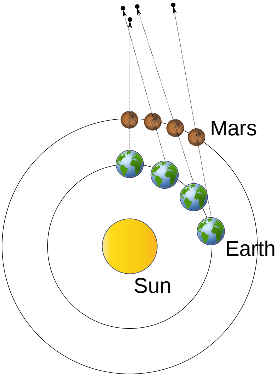
Verification of the physical model



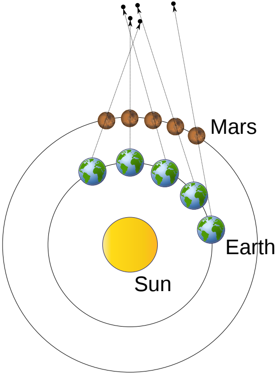
Verification of the physical model



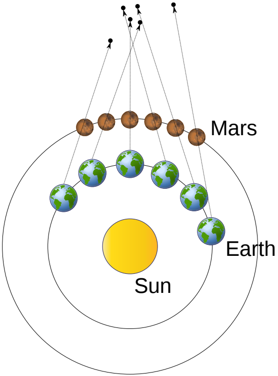
Verification of the physical model



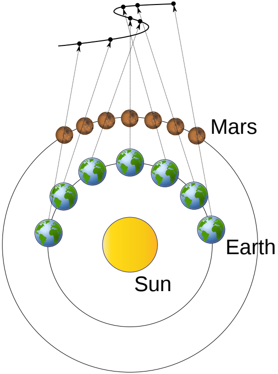
Verification of the physical model

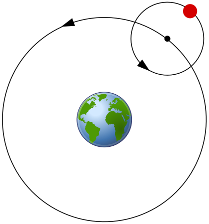


Verification of the physical model



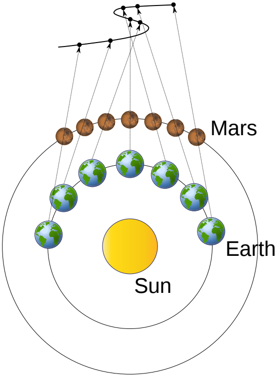
Verification of the physical model

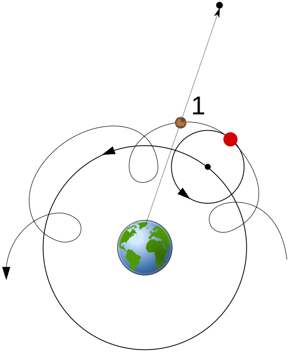




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

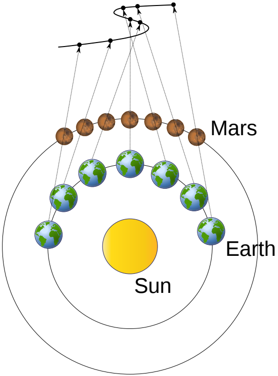
Verification of the physical model

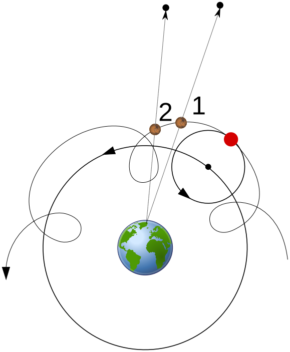




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

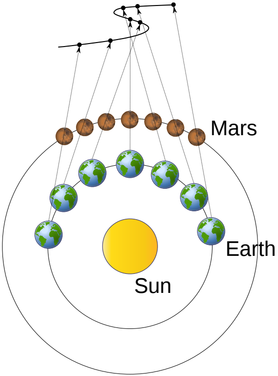
Verification of the physical model

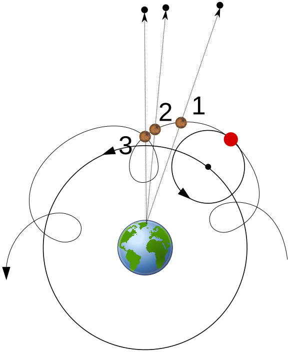




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

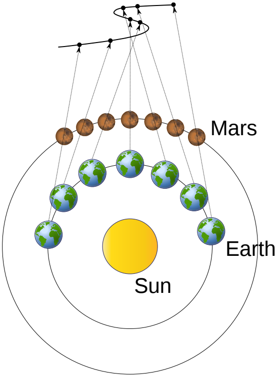
Verification of the physical model

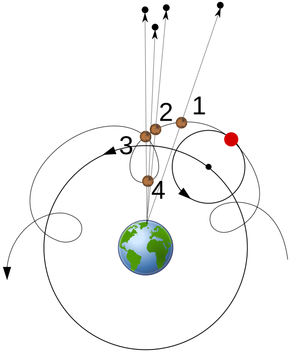




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

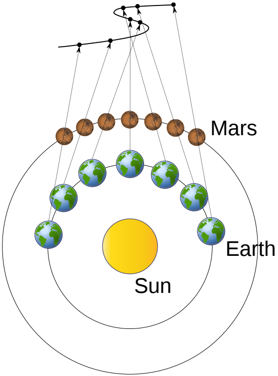
Verification of the physical model

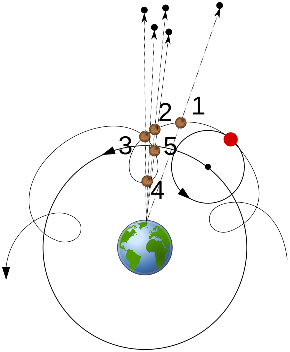




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

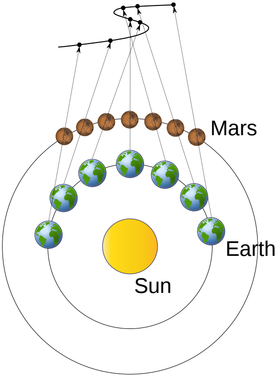
Verification of the physical model

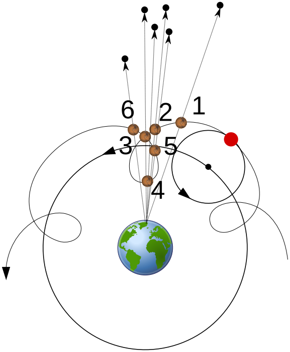




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

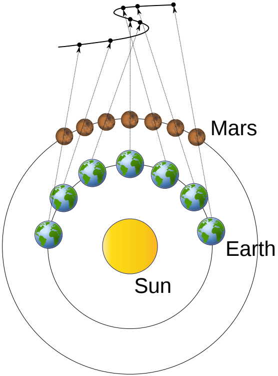
Verification of the physical model

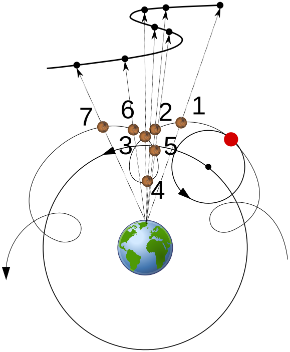




- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)

Verification of the physical model





- The perceived orbit of Mars from Earth shows a zig-zag (in contrast to the Sun, Mercury, Venus)
- Even though they were not 'right', Earth-centered models (Ptolemy) were still valid

Be aware of your uncertainties

Aleatory uncertainty

Uncertainty that arises due to inherent randomness of the system, features that are too complex to measure and take into account

Epistemic uncertainty

Uncertainty that arises due to lack of knowledge of the system, but could in principle be known

Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)

Errors in computer simulations

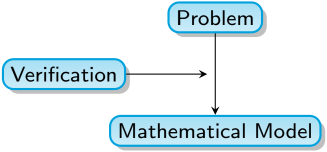
In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)

Verification and validation

Verification

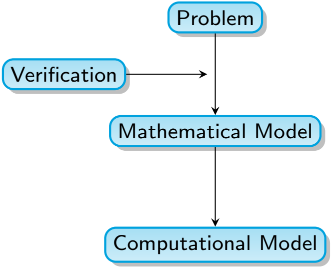
Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



Verification and validation

Verification

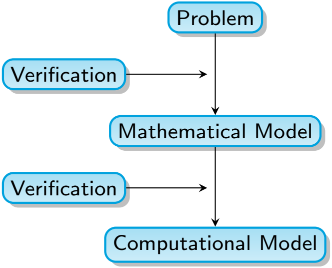
Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



Verification and validation

Verification

Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)

Errors in computer simulations

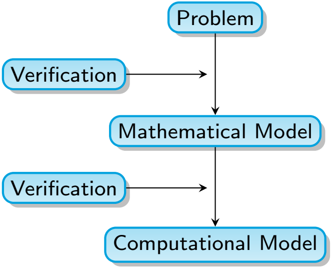
In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)
- Errors in the entered parameters

Verification and validation

Verification

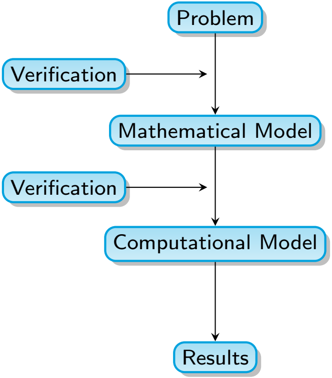
Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



Verification and validation

Verification

Verification is the process of mathematically and computationally assuring that the model computes what you have entered.



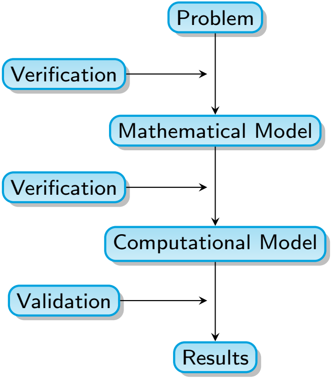
Verification and validation

Verification

Verification is the process of mathematically and computationally assuring that the model computes what you have entered.

Validation

Validation is the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model



Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)
- Errors in the entered parameters

Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)
- Errors in the entered parameters
- Roundoff- and truncation errors

Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)
- Errors in the entered parameters
- Roundoff- and truncation errors
- Break errors

Errors in computer simulations

In this course we will outline different numerical errors that may appear in computer simulations, and how these errors can affect the simulation results.

- Errors in the mathematical model (physics)
- Errors in the program (implementation)
- Errors in the entered parameters
- Roundoff- and truncation errors
- Break errors

Significant digits

A numerical result $\tilde{x}$ is an approximation of the real value x.

- Absolute error

$$\delta = |\tilde{x} - x|, x \neq 0$$

- Relative error

$$\frac{\delta}{\tilde{x}} = |\frac{\tilde{x} - x}{\tilde{x}}|$$

- Error margin

$$\tilde{x} - \delta \leq x \leq \tilde{x} + \delta$$

$$x = \tilde{x} \pm \delta$$

Significant digits

A numerical result $\tilde{x}$ is an approximation of the real value x.

- Absolute error

$$\delta = |\tilde{x} - x|, x \neq 0$$

- Relative error

$$\frac{\delta}{\tilde{x}} = |\frac{\tilde{x} - x}{\tilde{x}}|$$

- Error margin

$$\tilde{x} - \delta \leq x \leq \tilde{x} + \delta$$

$$x = \tilde{x} \pm \delta$$

Significant digits

A numerical result $\tilde{x}$ is an approximation of the real value x.

- Absolute error

$$\delta = |\tilde{x} - x|, x \neq 0$$

- Relative error

$$\frac{\delta}{\tilde{x}} = |\frac{\tilde{x} - x}{\tilde{x}}|$$

- Error margin

$$\tilde{x} - \delta \leq x \leq \tilde{x} + \delta$$

$$x = \tilde{x} \pm \delta$$

Significant digits

- $\tilde{x}$ has m significant digits if the absolute error in x is smaller or equal to 5 at the $(m+1)$-th position:

$$10^{q-1} \leq |\tilde{x}| \leq 10^q$$

$$|x - \tilde{x}| = 0.5 \times 10^{q-m}$$

- For example:

$$x = \frac{1}{3}, \tilde{x} = 0.333 \Rightarrow \delta = 0.00033333\ldots$$

3 significant digits

Significant digits

- $\tilde{x}$ has m significant digits if the absolute error in x is smaller or equal to 5 at the $(m+1)$-th position:

$$10^{q-1} \leq |\tilde{x}| \leq 10^q$$

$$|x - \tilde{x}| = 0.5 \times 10^{q-m}$$

- For example:

$$x = \frac{1}{3}, \tilde{x} = 0.333 \Rightarrow \delta = 0.00033333\ldots$$

3 significant digits

Today's outline

Representation of numbers

- Computers represent a number with a finite number of digits: each number is therefore an approximation due to roundoff and truncation errors.

Representation of numbers

- Computers represent a number with a finite number of digits: each number is therefore an approximation due to roundoff and truncation errors.
- In the decimal system, a digit c at position n has a value of $c \times 10^{n-1}$

Representation of numbers

- Computers represent a number with a finite number of digits: each number is therefore an approximation due to roundoff and truncation errors.
- In the decimal system, a digit c at position n has a value of $c \times 10^{n-1}$

Representation of numbers

- Computers represent a number with a finite number of digits: each number is therefore an approximation due to roundoff and truncation errors.
- In the decimal system, a digit c at position n has a value of $c \times 10^{n-1}$

4 5 2 1

Number of 1's: $1 \times 10^{(0)}$

Number of 10's: $2 \times 10^{(1)}$

Number of 100's: $5 \times 10^{(2)}$

Number of 1000's: $4 \times 10^{(3)}$

Representation of numbers

- Computers represent a number with a finite number of digits: each number is therefore an approximation due to roundoff and truncation errors.
- In the decimal system, a digit c at position n has a value of $c \times 10^{n-1}$

4 5 2 1

$\rightarrow$ Number of 1's: $1 \times 10^{(0)}$

$\rightarrow$ Number of 10's: $2 \times 10^{(1)}$

$\rightarrow$ Number of 100's: $5 \times 10^{(2)}$

$\rightarrow$ Number of 1000's: $4 \times 10^{(3)}$

$$(4521)_{10} = 4 \times 10^3 + 5 \times 10^2 + 2 \times 10^1 + 1 \times 10^0$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$(4521)_{10} =$

$=($

Representation of numbers

- You could use another basis, computers often use the basis 2:

$(4521)_{10} = 1 \times 2^{12} +$

$=(1$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} +$

$=(10$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} +$

$= (100$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} +$

$=(1000$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots$$

$$=(10001$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \dots$$
$$\dots 1 \times 2^{7} +$$

$$=(100011$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \dots$$
$$\dots 1 \times 2^{7} + 0 \times 2^{6} +$$

$$=(1000110$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots$$
$$\ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} +$$

$$=(10001101$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$(4521)_{10} = 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots$$
$$\ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \ldots$$

$$=(100011010$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$
\begin{aligned}
(4521)_{10} = & 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots \\
& \ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \ldots \\
& \ldots 1 \times 2^{3} + \\
& = (1000110101
\end{aligned}
$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$
\begin{aligned}
(4521)_{10} = & 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots \\
& \ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \ldots \\
& \ldots 1 \times 2^{3} + 0 \times 2^{2} + \\
= & (10001101010
\end{aligned}
$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$
\begin{aligned}
(4521)_{10} = & 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots \\
& \ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \ldots \\
& \ldots 1 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + \\
& =(100011010100
\end{aligned}
$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$
\begin{aligned}
(4521)_{10} = & 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \dots \\
& \dots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \dots \\
& \dots 1 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} \\
& = (1000110101001)_2
\end{aligned}
$$

Representation of numbers

- You could use another basis, computers often use the basis 2:

$$
\begin{aligned}
(4521)_{10} = & 1 \times 2^{12} + 0 \times 2^{11} + 0 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + \ldots \\
& \ldots 1 \times 2^{7} + 0 \times 2^{6} + 1 \times 2^{5} + 0 \times 2^{4} + \ldots \\
& \ldots 1 \times 2^{3} + 0 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} \\
& = (1000110101001)_2
\end{aligned}
$$

- In general:

$$(c_m \ldots c_1 c_0)_q = c_0 q^0 + c_1 q^1 + \ldots + c_m q^m, c \in \{0, 1, 2, \ldots, q-1\}$$

Representation of numbers

- Numbers are stored in binary in the memory of a computer, in segments of a specific length (called a *word*).
- We distinguish multiple types of numbers:
  - Integers: $-301, -1, 0, 1, 96, 2293, \ldots$
  - Floating points: $-301.01, 0.01, 3.14159265, 14498.2$
- A binary integer representation looks like the following bit sequence:

$$z = \sigma\left(c_0 2^0 + c_1 2^1 + \ldots + c_{\lambda-1} 2^{\lambda-1}\right)$$

  σ is the sign of z (+ or -), and λ is the length of the word
- Endianness: the order of bits stored by a computer

Representation of numbers

- Numbers are stored in binary in the memory of a computer, in segments of a specific length (called a *word*).
- We distinguish multiple types of numbers:
  - Integers: $-301, -1, 0, 1, 96, 2293, \ldots$
  - Floating points: $-301.01, 0.01, 3.14159265, 14498.2$
- A binary integer representation looks like the following bit sequence:

$$z = \sigma \left(c_0 2^0 + c_1 2^1 + \ldots + c_{\lambda-1} 2^{\lambda-1} \right)$$

  σ is the sign of z (+ or -), and λ is the length of the word
- Endianness: the order of bits stored by a computer

Representation of numbers

- Numbers are stored in binary in the memory of a computer, in segments of a specific length (called a *word*).
- We distinguish multiple types of numbers:
  - Integers: $-301, -1, 0, 1, 96, 2293, \ldots$
  - Floating points: $-301.01, 0.01, 3.14159265, 14498.2$
- A binary integer representation looks like the following bit sequence:

$$z = \sigma\left(c_0 2^0 + c_1 2^1 + \ldots + c_{\lambda-1} 2^{\lambda-1}\right)$$

  σ is the sign of z (+ or -), and λ is the length of the word
- Endianness: the order of bits stored by a computer

Representation of numbers

- Numbers are stored in binary in the memory of a computer, in segments of a specific length (called a *word*).
- We distinguish multiple types of numbers:
  - Integers: $-301, -1, 0, 1, 96, 2293, \ldots$
  - Floating points: $-301.01, 0.01, 3.14159265, 14498.2$
- A binary integer representation looks like the following bit sequence:

$$z = \sigma \left(c_0 2^0 + c_1 2^1 + \ldots + c_{\lambda-1} 2^{\lambda-1} \right)$$

  σ is the sign of z (+ or -), and λ is the length of the word
- Endianness: the order of bits stored by a computer

Excercise

- Convert the following decimal number to base-2: 214

Excercise

- Convert the following decimal number to base-2: 214

$$214_{10} = 11010110_2$$

Excercise

- Convert the following decimal number to base-2: 214

  $$214_{10} = 11010110_2$$

- Excel:
  - Decimal: `=DEC2BIN(214)`
  - Octal: `=DEC2OCT(214)`
  - Hexadecimal: `=DEC2HEX(214)`

Excercise

- Convert the following decimal number to base-2: 214

$$214_{10} = 11010110_2$$

- Excel:
  - Decimal: `=DEC2BIN(214)`
  - Octal: `=DEC2OCT(214)`
  - Hexadecimal: `=DEC2HEX(214)`
- Matlab:
  - Decimal: `dec2bin(214)`
  - Other base: `dec2base(214,<base>)`

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$

(carry one)

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$
\begin{array}{cccc}
 & 1 & 4 & 5 \\
+ & & 2 & 3 \\
\hline
 & 1 & 6 & 8
\end{array}
$$

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$\begin{array}{cccc} & 1 & 4 & 5 \\ + & & 2 & 3 \\ \hline & 1 & 6 & 8 \end{array}$$

$$\begin{array}{ccccccccc} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$
\begin{array}{rrrr}
 & 1 & 4 & 5 \\
+ & & 2 & 3 \\
\hline
 & 1 & 6 & 8
\end{array}
$$

$$
\begin{array}{rrrrrrrrr}
 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
+ & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ + & & 2 & 3 \\ \hline & 1 & 6 & 8 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Subtraction:

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$
(borrow one)

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ + & & 2 & 3 \\ \hline & 1 & 6 & 8 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Subtraction:

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$
(borrow one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ - & & 2 & 3 \\ \hline & 1 & 2 & 2 \end{array}$$

Arithmetic operations with binary numbers

Addition:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$
(carry one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ + & & 2 & 3 \\ \hline & 1 & 6 & 8 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Subtraction:

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = 1$
(borrow one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ - & & 2 & 3 \\ \hline & 1 & 2 & 2 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ - & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline \end{array}$$

Arithmetic operations with binary numbers

Addition:

$0+0=0$
$0+1=1$
$1+0=1$
$1+1=0$
(carry one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ + & & 2 & 3 \\ \hline & 1 & 6 & 8 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ + & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Subtraction:

$0-0=0$
$1-0=1$
$1-1=0$
$0-1=1$
(borrow one)

$$\begin{array}{rrrr} & 1 & 4 & 5 \\ - & & 2 & 3 \\ \hline & 1 & 2 & 2 \end{array}$$

$$\begin{array}{rrrrrrrrr} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ - & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{array}$$

- Multiplication and division are more expensive, and more elaborate

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	`-2147483648`
`intmax`	`2147483647`
`i = int16(intmax)`	`i = 32767`

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	`-2147483648`
`intmax`	`2147483647`
`i = int16(intmax)`	`i = 32767`
`whos i`	`int16` information

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	`i = 32767`
`whos i`	`int16` information
`i = i + 100`	`i = 32767`

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	i = 32767
`whos i`	int16 information
`i = i + 100`	i = 32767
`realmax`	1.7977e+308

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	`i = 32767`
`whos i`	`int16` information
`i = i + 100`	`i = 32767`
`realmax`	1.7977e+308
`f = 0.1`	

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	`i = 32767`
`whos i`	`int16` information
`i = i + 100`	`i = 32767`
`realmax`	1.7977e+308
`f = 0.1`	
`whos f`	double information

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	`i = 32767`
`whos i`	`int16` information
`i = i + 100`	`i = 32767`
`realmax`	`1.7977e+308`
`f = 0.1`	
`whos f`	double information
`format long e`	

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	i = 32767
`whos i`	int16 information
`i = i + 100`	i = 32767
`realmax`	1.7977e+308
`f = 0.1`	
`whos f`	double information
`format long e`	
`realmax`	1.797693134862316e+308

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	i = 32767
`whos i`	int16 information
`i = i + 100`	i = 32767
`realmax`	1.7977e+308
`f = 0.1`	
`whos f`	double information
`format long e`	
`realmax`	1.797693134862316e+308
`f`	

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	i = 32767
`whos i`	int16 information
`i = i + 100`	i = 32767
`realmax`	1.7977e+308
`f = 0.1`	
`whos f`	double information
`format long e`	
`realmax`	1.797693134862316e+308
`f`	
`fprintf("%0.16f",f)`	0.1000000000000000

Excercise

Try the following commands in Matlab:

Command	Result
`intmin`	-2147483648
`intmax`	2147483647
`i = int16(intmax)`	i = 32767
`whos i`	int16 information
`i = i + 100`	i = 32767
`realmax`	1.7977e+308
`f = 0.1`	
`whos f`	double information
`format long e`	
`realmax`	1.797693134862316e+308
`f`	
`fprintf("%0.16f",f)`	0.1000000000000000
`fprintf("%0.20f",f)`	0.10000000000000000555

Representation of integer numbers

- In Matlab, integers of the type `int32` are represented by 32-bit words ($\lambda = 31$).

[1] Matlab does not perform actual integer overflows, it just stops at the maximum.

Representation of integer numbers

- In Matlab, integers of the type `int32` are represented by 32-bit words ($\lambda = 31$).
- The set of numbers that an `int32` z can represent is:

$$-2^{31} \leq z \leq 2^{31} - 1 \approx 2 \times 10^9$$

[1] Matlab does not perform actual integer overflows, it just stops at the maximum.

Representation of integer numbers

- In Matlab, integers of the type `int32` are represented by 32-bit words ($\lambda = 31$).
- The set of numbers that an `int32` z can represent is:

$$-2^{31} \leq z \leq 2^{31} - 1 \approx 2 \times 10^9$$

- If, during a calculation, an integer number becomes larger than $2^\lambda - 1$, the computer reports an overflow[1]

[1] Matlab does not perform actual integer overflows, it just stops at the maximum.

Representation of integer numbers

- In Matlab, integers of the type `int32` are represented by 32-bit words ($\lambda = 31$).
- The set of numbers that an `int32` z can represent is:

$$-2^{31} \leq z \leq 2^{31} - 1 \approx 2 \times 10^9$$

- If, during a calculation, an integer number becomes larger than $2^\lambda - 1$, the computer reports an overflow[1]
- How can a computer identify an overflow?

[1] Matlab does not perform actual integer overflows, it just stops at the maximum.

Representation of real (floating point) numbers

- Formally, a real number is represented by the following bit sequence

$$x = \sigma \left(2^{-1} + c_2 2^{-2} + \ldots + c_m 2^{-m} \right) 2^{e-1023}$$

Here, σ is the sign of x and e is an integer value.

- A floating point number hence contains sections that contain the sign, the exponent and the mantissa

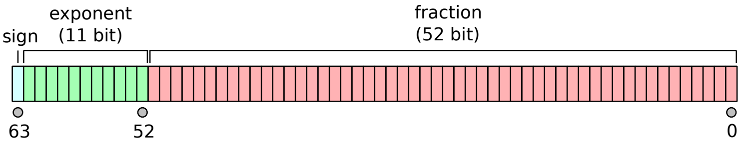


Image: Wikimedia Commons CC by-SA

Representation of real (floating point) numbers

- Example: $\lambda = 3,\ m = 2,\ x = \frac{2}{3}$

$$x = \pm\left(2^{-1} + c_2 2^{-2}\right) 2^e$$

  - $c_0 \in \{0, 1\}$
  - $e = \pm a_0 2^0$
  - $a_0 \in \{0, 1\}$
- Truncation: $fl(x) = 2^{-1} = 0.5$
- Round off: $fl(x) = 2^{-1} + 2^{-2} = 0.75$

Today's outline

1. Introduction
2. Roundoff and truncation errors
3. Break errors
4. Loss of digits
5. (Un)stable methods
6. Symbolic math
7. Summary

Trigonometric, Logarithmic, and Exponential computations

- Processors can do logic and arithmetic instructions
- Trigonometric, logarithmic and exponential calculations are "higher-level" functions:
  exp, sin, cos, tan, sec, arcsin, arccos, arctan, log, ln, ...
- Such functions can be performed using these "low level" instructions, for instance using a Taylor series:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Trigonometric, Logarithmic, and Exponential computations

- Processors can do logic and arithmetic instructions
- Trigonometric, logarithmic and exponential calculations are "higher-level" functions:
  exp, sin, cos, tan, sec, arcsin, arccos, arctan, log, ln, ...
- Such functions can be performed using these "low level" instructions, for instance using a Taylor series:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Trigonometric, Logarithmic, and Exponential computations

- Processors can do logic and arithmetic instructions
- Trigonometric, logarithmic and exponential calculations are "higher-level" functions:
  exp, sin, cos, tan, sec, arcsin, arccos, arctan, log, ln, ...
- Such functions can be performed using these "low level" instructions, for instance using a Taylor series:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Trigonometric, Logarithmic, and Exponential computations

- These operations involve many multiplications and additions, and are therefore *expensive*
- Computations can only take finite time, for infinite series, calculations are interrupted at N

$$\sin(x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^N}{(2N+1)!} x^{2N+1}$$

$$e^x = \sum_{n=0}^{N} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^N}{N!}$$

- This results in a *break error*

Trigonometric, Logarithmic, and Exponential computations

- These operations involve many multiplications and additions, and are therefore *expensive*
- Computations can only take finite time, for infinite series, calculations are interrupted at N

$$\sin(x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^N}{(2N+1)!} x^{2N+1}$$

$$e^x = \sum_{n=0}^{N} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^N}{N!}$$

- This results in a *break error*

Trigonometric, Logarithmic, and Exponential computations

- These operations involve many multiplications and additions, and are therefore *expensive*
- Computations can only take finite time, for infinite series, calculations are interrupted at N

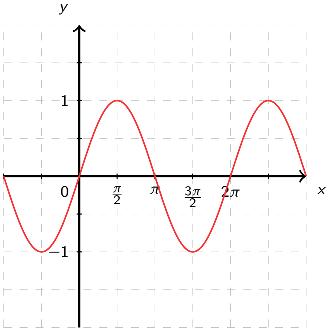
$$\sin(x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^N}{(2N+1)!} x^{2N+1}$$

$$e^x = \sum_{n=0}^{N} \frac{x^n}{x!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots + \frac{x^N}{N!}$$

- This results in a *break error*

Algorithm for sine-computation

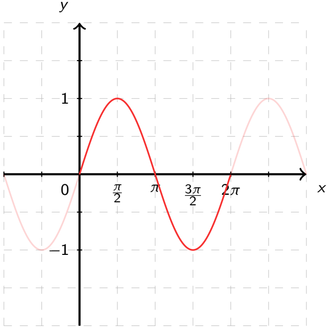
A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

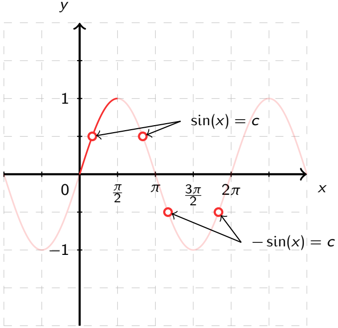
1. Use periodicity so that $0 \leq x \leq 2\pi$



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

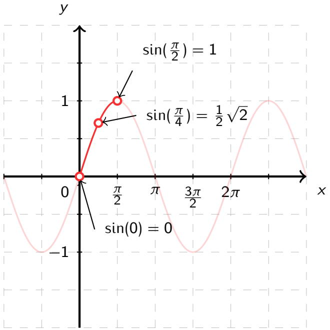
1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry ($0 \leq x \leq \frac{\pi}{2}$)



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

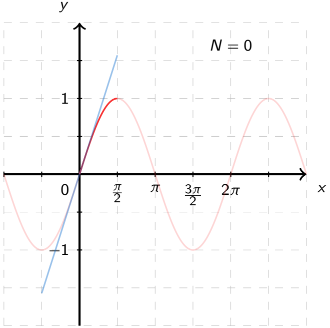
1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry $(0 \leq x \leq \frac{\pi}{2})$
3. Use lookup tables for known values



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

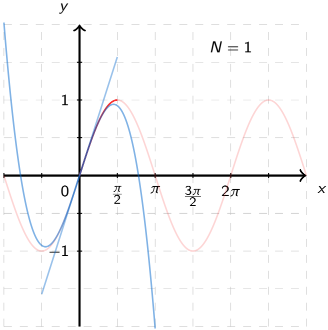
1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry $(0 \leq x \leq \frac{\pi}{2})$
3. Use lookup tables for known values
4. Perform taylor expansion



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

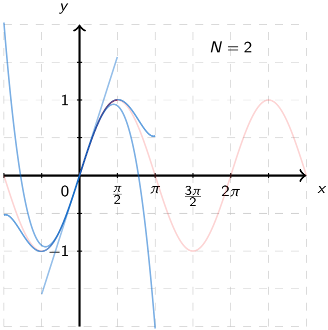
1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry $(0 \leq x \leq \frac{\pi}{2})$
3. Use lookup tables for known values
4. Perform taylor expansion



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

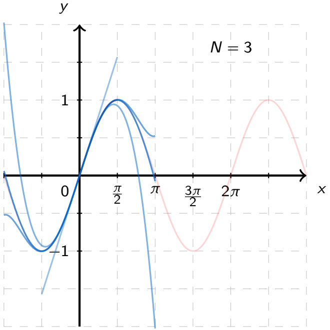
1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry ($0 \leq x \leq \frac{\pi}{2}$)
3. Use lookup tables for known values
4. Perform taylor expansion



Algorithm for sine-computation

A computer may use a clever algorithm to limit the number of operations required to perform a higher-level function. A (fictional!) example for the computation of $\sin(x)$:

1. Use periodicity so that $0 \leq x \leq 2\pi$
2. Use symmetry ($0 \leq x \leq \frac{\pi}{2}$)
3. Use lookup tables for known values
4. Perform taylor expansion



Today's outline

1. Introduction
2. Roundoff and truncation errors
3. Break errors
4. Loss of digits
5. (Un)stable methods
6. Symbolic math
7. Summary

Loss of digits

- During operations such as $+$, $-$, $\times$, $\div$, an error can add up
- Consider the summation of x and y

$$\tilde{x} - \delta \leq x \leq \tilde{x} + \delta \quad \text{and} \quad \tilde{y} - \varepsilon \leq y \leq \tilde{y} + \varepsilon$$

$$(\tilde{x} + \tilde{y}) - (\delta + \varepsilon) \leq x + y \leq (\tilde{x} + \tilde{y}) + (\delta + \varepsilon)$$

Loss of digits: Example 1

$x = \pi, \tilde{x} = 3.1416$

$y = 22/7, \tilde{y} = 3.1429$

Loss of digits: Example 1

$$\left.\begin{aligned} x = \pi, \tilde{x} = 3.1416 \\ y = 22/7, \tilde{y} = 3.1429 \end{aligned}\right\} \Rightarrow \begin{aligned} \delta = \tilde{x} - x = 7.35 \times 10^{-6} \\ \varepsilon = \tilde{y} - y = 4.29 \times 10^{-5} \end{aligned}\Bigg\}$$

Loss of digits: Example 1

$$\left.\begin{array}{l} x = \pi, \tilde{x} = 3.1416 \\ y = 22/7, \tilde{y} = 3.1429 \end{array}\right\} \Rightarrow \left.\begin{array}{l} \delta = \tilde{x} - x = 7.35 \times 10^{-6} \\ \varepsilon = \tilde{y} - y = 4.29 \times 10^{-5} \end{array}\right\}$$

$$x + y = \tilde{x} + \tilde{y} \pm (\delta + \varepsilon) \approx 6.2845 - 5.025 \times 10^{-5}$$

$$x - y = \tilde{x} - \tilde{y} \pm (\delta + \varepsilon) \approx -0.0013 + 3.55 \times 10^{-5}$$

- The absolute error is small ($\approx 10^{-5}$), but the relative error is much bigger (0.028).
- Adding up the errors results in a loss of significant digits!

Loss of digits: Example 2

- Calculate e^{-5}
  - Use the Taylor series
  - Calculate the first 26 terms ($N = 26$)
- Now repeat the calculation, but use for each calculation only 4 digits. What do you find?

- Without errors you would find: $e^{-5} = 0.006738$
- If you only use 4 digits in the calculations, you'll find 0.00998

Loss of digits: Example 2

- Calculate e^{-5}
  - Use the Taylor series
  - Calculate the first 26 terms ($N = 26$)
- Now repeat the calculation, but use for each calculation only 4 digits. What do you find?
  Use: `str2double(sprintf('%.4g', term))`
- Without errors you would find: $e^{-5} = 0.006738$
- If you only use 4 digits in the calculations, you'll find 0.00998

Loss of digits: Example 2

- Calculate e^{-5}
  - Use the Taylor series
  - Calculate the first 26 terms ($N = 26$)
- Now repeat the calculation, but use for each calculation only 4 digits. What do you find?
  Use: `str2double(sprintf('%.4g', term))`
- Without errors you would find: $e^{-5} = 0.006738$
- If you only use 4 digits in the calculations, you'll find 0.00998

Loss of digits: Example 2

- Calculate e^{-5}
  - Use the Taylor series
  - Calculate the first 26 terms ($N = 26$)
- Now repeat the calculation, but use for each calculation only 4 digits. What do you find?
  Use: `str2double(sprintf('%.4g', term))`
- Without errors you would find: $e^{-5} = 0.006738$
- If you only use 4 digits in the calculations, you'll find 0.00998

Badly (ill) conditioned problems

We consider a system $F(x, y)$ that computes a solution from input data. The input data may have errors:

$$\xrightarrow{\;x\;} \boxed{F(x, y) \; = \; 0} \xrightarrow{\;y\;} \text{(Without errors)}$$

Badly (ill) conditioned problems

We consider a system $F(x, y)$ that computes a solution from input data. The input data may have errors:

$$\xrightarrow{\quad x \quad} \boxed{F(x,y) \; = \; 0} \xrightarrow{\quad y \quad} \text{(Without errors)}$$

$$\xrightarrow{x + \delta x} \boxed{F(x + \delta x, y + \delta y) = 0} \xrightarrow{y + \delta y} \text{(With errors)}$$

Badly (ill) conditioned problems

We consider a system $F(x, y)$ that computes a solution from input data. The input data may have errors:

$$\xrightarrow{\;x\;} \boxed{F(x,y) \;=\; 0} \xrightarrow{\;y\;} \text{(Without errors)}$$

$$\xrightarrow{\;x+\delta x\;} \boxed{F(x+\delta x, y+\delta y) = 0} \xrightarrow{\;y+\delta y\;} \text{(With errors)}$$

$$y(x+\delta x) - y(x) \approx y'(x)\delta x$$

Propagated error on the basis of Taylor expansion

$$C = \max_{\delta x}\left(\left|\frac{\delta y/y}{\delta x/x}\right|\right)$$

Condition criterion, $C < 10$ error development small

Badly (ill) conditioned problems: Example

Solve the following linear system in Matlab using double and single precision:

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix}, \quad x = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix}$$

Badly (ill) conditioned problems: Example

Solve the following linear system in Matlab using double and single precision:

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix}, \quad x = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix}, \quad y = \begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix}$$

Badly (ill) conditioned problems: Example

Solve the following linear system in Matlab using double and single precision:

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix}, \quad x = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix}, \quad y = \begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix}$$

Double precision

```
>> clear;clc;format long e;
>> A = [[1.2969 0.8648];
    [0.2161 0.1441]];
>> x = [0.8642; 0.1440];
>> y = A\x
y =
     2.000000002400302e+00
    -2.000000003599621e+00
```

Badly (ill) conditioned problems: Example

Solve the following linear system in Matlab using double and single precision:

$$A = \begin{bmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{bmatrix}, \quad x = \begin{bmatrix} 0.8642 \\ 0.1440 \end{bmatrix}, \quad y = \begin{bmatrix} 2.0 \\ 2.0 \end{bmatrix}$$

Double precision

```
>> clear;clc;format long e;
>> A = [[1.2969 0.8648];
    [0.2161 0.1441]];
>> x = [0.8642; 0.1440];
>> y = A\x
y =
      2.000000002400302e+00
     -2.000000003599621e+00
```

Single precision

```
>> clear;clc;format long e;
>> A = single(
   [[1.2969 0.8648];
   [0.2161 0.1441]] );
>> x = single(
   [0.8642; 0.1440] );
>> y = A\x
y =
     1.3331791e+00
    -1.0000000e+00
```

Badly (ill) conditioned problems: Example

- Matlab already warned us about the bad condition number:

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.148983e-08.
```

- The `RCOND` is the reciprocal condition number
- A small error in x results in a big error in y. This is called an ill conditioned problem.

Badly (ill) conditioned problems: Example

- Matlab already warned us about the bad condition number:

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.148983e-08.
```

- The `RCOND` is the reciprocal condition number
- A small error in x results in a big error in y. This is called an ill conditioned problem.

Badly (ill) conditioned problems: Example

- Matlab already warned us about the bad condition number:

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.148983e-08.
```

- The `RCOND` is the reciprocal condition number
- A small error in x results in a big error in y. This is called an ill conditioned problem.

Today's outline

(Un)stable methods

- The condition criterion does not tell you anything about the quality of a numerical solution method!
- It is very well possible that a certain solution method is more sensitive for one problem than another
- If the method propagates the error, we call it an *unstable method*. Let's look at an example.

The Golden mean

- Let's evaluate the following recurrent relationship:

$$y_{n+1} = y_{n-1} - y_n$$

$$y_0 = 1, \quad y_1 = \frac{2}{1+\sqrt{5}}$$

- You can prove (by substitution) that:

$$y_n = x^{-n}, \quad n = 0, 1, 2, \ldots, \quad x = \frac{1+\sqrt{5}}{2}$$

The Golden mean

- Let's evaluate the following recurrent relationship:

$$y_{n+1} = y_{n-1} - y_n$$

$$y_0 = 1, \quad y_1 = \frac{2}{1+\sqrt{5}}$$

- You can prove (by substitution) that:

$$y_n = x^{-n}, \quad n = 0, 1, 2, \ldots, \quad x = \frac{1+\sqrt{5}}{2}$$

The Golden mean

Recurrent version

```
% initialise
y(1) = 1;
y(2) = 2 / (1 + sqrt(5));

% Perform recurrent
    approach
for n = 2:39
     y(n+1) = y(n-1)-y(n);
end
```

The Golden mean

Recurrent version

```
% initialise
y(1) = 1;
y(2) = 2 / (1 + sqrt(5));

% Perform recurrent
    approach
for n = 2:39
    y(n+1) = y(n-1)-y(n);
end
```

Powerlaw version

```
% initialise
x = (1 + sqrt(5))/2;
y2(1) = x^0; % n = 1

% Perform powerlaw apprach
for n = 0:39
    y2(n+1) = x^-n
end
```

The Golden mean

n	Recurrent	Powerlaw
1	1.0000	1.0000
1	0.6180	0.6180
2	0.3820	0.3820
3	0.2361	0.2361
...	...	...
37	$3.080 \cdot 10^{-08}$	$2.995 \cdot 10^{-08}$
38	$1.714 \cdot 10^{-08}$	$1.851 \cdot 10^{-08}$
39	$1.366 \cdot 10^{-08}$	$1.144 \cdot 10^{-08}$
40	$3.485 \cdot 10^{-08}$	$7.071 \cdot 10^{-08}$

- The recurrent approach enlarges errors from earlier calculations!

The Golden mean

n	Recurrent	Powerlaw
1	1.0000	1.0000
1	0.6180	0.6180
2	0.3820	0.3820
3	0.2361	0.2361
...	...	...
37	$3.080 \cdot 10^{-08}$	$2.995 \cdot 10^{-08}$
38	$1.714 \cdot 10^{-08}$	$1.851 \cdot 10^{-08}$
39	$1.366 \cdot 10^{-08}$	$1.144 \cdot 10^{-08}$
40	$3.485 \cdot 10^{-08}$	$7.071 \cdot 10^{-08}$

- The recurrent approach enlarges errors from earlier calculations!

Example 1: Explanation

Recall example 1, where the errors blew up our computation of 0.1, whereas they did not for 2. Why did we see these results?

Example 1: Explanation

Recall example 1, where the errors blew up our computation of 0.1, whereas they did not for 2. Why did we see these results?

- The number 0.1 is not exactly represented in binary
  - A tiny error can accumulate up to catastrophic proportions!
- The number 2 does have an exact binary representation

Example 1: Explanation

Recall example 1, where the errors blew up our computation of 0.1, whereas they did not for 2. Why did we see these results?

- The number 0.1 is not exactly represented in binary
  - A tiny error can accumulate up to catastrophic proportions!
- The number 2 does have an exact binary representation

Example 2

Start your calculation program of choice (Excel, Matlab, ...)

Example 2

Start your calculation program of choice (Excel, Matlab, ...)

Calculate the result of y:

$$y = e^{\pi} - \pi$$

Example 2

Start your calculation program of choice (Excel, Matlab, ...)

Calculate the result of y:

$$y = e^{\pi} - \pi = 19.999099979$$

Example 2

Start your calculation program of choice (Excel, Matlab, ...)

Calculate the result of y:

$$y = e^{\pi} - \pi = 19.999099979 \neq 20$$

Example 2

Start your calculation program of choice (Excel, Matlab, ...)

Calculate the result of y:

$$y = e^{\pi} - \pi = 19.999099979 \neq 20$$

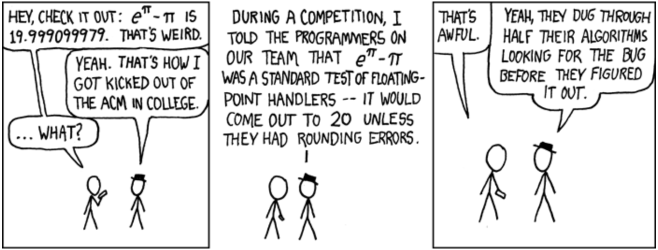


Image: xkcd

Today's outline

Symbolic math packages

Definition

The use of computers to manipulate mathematical equations and expressions in symbolic form, as opposed to manipulating the numerical quantities represented by those symbols.

- Symbolic integration or differentiation, substitution of one expression into another
- Simplification of an expression, change of subject etc.
- Packages and toolboxes:

Symbolic math packages

Mathematica	Well known software package, license available via TU/e
Maple	Well known, license available via TU/e
Wolfram|Alpha	Web-based interface by Mathematica developer. Less powerful in mathematical respect, but more accessible and has a broad application range (unit conversion, semantic commands).
Sage	Open-source alternative to Maple, Mathematica, Magma, and MATLAB.
Matlab	Symbolic math toolbox

Symbolic math: simplify

$$f(x) = (x-1)(x+1)(x^2+1)+1$$

Symbolic math: simplify

$$f(x) = (x-1)(x+1)(x^2+1)+1$$

```
>> syms x
>> f = (x - 1)*(x + 1)*(x^2 + 1) + 1
```

Symbolic math: simplify

$$f(x) = (x-1)(x+1)(x^2+1)+1$$

```
>> syms x
>> f = (x - 1)*(x + 1)*(x^2 + 1) + 1

f =

(x^2 + 1)*(x - 1)*(x + 1) + 1
```

Symbolic math: simplify

$$f(x) = (x - 1)(x + 1)(x^2 + 1) + 1$$

```
>> syms x
>> f = (x - 1)*(x + 1)*(x^2 + 1) + 1

f =

(x^2 + 1)*(x - 1)*(x + 1) + 1

>> f2 = simplify(f)
```

Symbolic math: simplify

$$f(x) = (x-1)(x+1)(x^2+1)+1$$

```
>> syms x
>> f = (x - 1)*(x + 1)*(x^2 + 1) + 1

f =

(x^2 + 1)*(x - 1)*(x + 1) + 1

>> f2 = simplify(f)

f2 =

x^4
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)

my_f_int = log(x + 1)/3 - log((x - 1/2)^2 + 3/4)/6 +
    (3^(1/2)*atan((2*3^(1/2)*(x - 1/2))/3))/3
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)

my_f_int = log(x + 1)/3 - log((x - 1/2)^2 + 3/4)/6 +
    (3^(1/2)*atan((2*3^(1/2)*(x - 1/2))/3))/3

>> my_f_diff = diff(my_f_int)
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)

my_f_int = log(x + 1)/3 - log((x - 1/2)^2 + 3/4)/6 +
    (3^(1/2)*atan((2*3^(1/2)*(x - 1/2))/3))/3

>> my_f_diff = diff(my_f_int)

my_f_diff = 1/(3*(x + 1)) + 2/(3*((4*(x - 1/2)^2)/3 +
    1)) - (2*x - 1)/(6*((x - 1/2)^2 + 3/4))
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)

my_f_int = log(x + 1)/3 - log((x - 1/2)^2 + 3/4)/6 +
    (3^(1/2)*atan((2*3^(1/2)*(x - 1/2))/3))/3

>> my_f_diff = diff(my_f_int)

my_f_diff = 1/(3*(x + 1)) + 2/(3*((4*(x - 1/2)^2)/3 +
    1)) - (2*x - 1)/(6*((x - 1/2)^2 + 3/4))

>> simplify(my_f_diff)
```

Symbolic math: integration and differentiation

$$f(x) = \frac{1}{x^3 + 1}$$

```
>> syms x
>> f = 1/(x^3+1);
>> my_f_int = int(f)

my_f_int = log(x + 1)/3 - log((x - 1/2)^2 + 3/4)/6 +
    (3^(1/2)*atan((2*3^(1/2)*(x - 1/2))/3))/3

>> my_f_diff = diff(my_f_int)

my_f_diff = 1/(3*(x + 1)) + 2/(3*((4*(x - 1/2)^2)/3 +
    1)) - (2*x - 1)/(6*((x - 1/2)^2 + 3/4))

>> simplify(my_f_diff)

ans = 1/(x^3 + 1)
```

Symbolic math: exercises

Exercise 1

Simplify the following expression:

$$f(x) = \frac{2\tan x}{(1+\tan^2 x)}$$

Symbolic math: exercises

Exercise 1

Simplify the following expression:

$$f(x) = \frac{2\tan x}{(1+\tan^2 x)} = \sin 2x$$

```
>> simplify(2*tan(x)/(1 + tan(x)^2))
```

Symbolic math: exercises

Exercise 1

Simplify the following expression:

$$f(x) = \frac{2\tan x}{(1+\tan^2 x)} = \sin 2x$$

```
>> simplify(2*tan(x)/(1 + tan(x)^2))
```

Exercise 2

Calculate the *value* of p:

$$p = \int_0^{10} \frac{e^x - e^{-x}}{\sinh x} dx$$

Symbolic math: exercises

Exercise 1

Simplify the following expression:

$$f(x) = \frac{2\tan x}{(1+\tan^2 x)} = \sin 2x$$

```
>> simplify(2*tan(x)/(1 + tan(x)^2))
```

Exercise 2

Calculate the *value* of p:

$$p = \int_0^{10} \frac{e^x - e^{-x}}{\sinh x} dx$$

```
>> f = ((exp(x)- exp(-x))/sinh(x));
>> p = int(f,0,10)
p = 20
```

Symbolic math: root finding

A root finding method searches for the values where a function reaches zero. We will cover the numerical methods later, here we show how to use root finding with symbolic math in Matlab.

Symbolic math function

$$f(x) = \frac{3}{x^2 + 3x} - 2$$

Symbolic math: root finding

A root finding method searches for the values where a function reaches zero. We will cover the numerical methods later, here we show how to use root finding with symbolic math in Matlab.

Symbolic math function

$$f(x) = \frac{3}{x^2 + 3x} - 2$$

```
>> syms x
>> f = 3 / (x^2 + 3*x) - 2;
>> solve(f)
ans =
   15^(1/2)/2 - 3/2
 - 15^(1/2)/2 - 3/2
```

Symbolic math: root finding

A root finding method searches for the values where a function reaches zero. We will cover the numerical methods later, here we show how to use root finding with symbolic math in Matlab.

Symbolic math function

$$f(x) = \frac{3}{x^2 + 3x} - 2$$

```
>> syms x
>> f = 3 / (x^2 + 3*x) - 2;
>> solve(f)
ans =
   15^(1/2)/2 - 3/2
 - 15^(1/2)/2 - 3/2
```

Function as a string

$$f(x) = x^2 - 4x + 2$$

Symbolic math: root finding

A root finding method searches for the values where a function reaches zero. We will cover the numerical methods later, here we show how to use root finding with symbolic math in Matlab.

Symbolic math function

$$f(x) = \frac{3}{x^2 + 3x} - 2$$

```
>> syms x
>> f = 3 / (x^2 + 3*x) - 2;
>> solve(f)
ans =
   15^(1/2)/2 - 3/2
 - 15^(1/2)/2 - 3/2
```

Function as a string

$$f(x) = x^2 - 4x + 2$$

```
>> solve('x^2 - 4*x + 2')
ans =
 2^(1/2) + 2
 2 - 2^(1/2)
```

Symbolic math toolbox: variable precision arithmetic

Variable precision can be used to specify the number of significant digits.

```
>> p = vpa(1/3,16)
p = 0.3333333333333333
>> p = vpa(1/3,4)
p = 0.3333
>> a = vpa(0.1, 30)
a = 0.1
>> b = vpa(0.1, 5);
b = 0.1
>> a-b

ans = 0.0000000000000056843418860808014869689938467514
```

Today's outline

Summary

- Numerical errors mar arise due to truncation, roundoff and break errors, which may seriously affect the accuracy of your solution
- Errors may propagate and accumulate, leading to smaller accuracy
- Ill-conditioned problems and unstable methods have to be identified so that proper measures can be taken
- Symbolic math computations may be performed to solve certain equations algebraically, bypassing numerical errors, but this is not always possible.

Summary

- Numerical errors mar arise due to truncation, roundoff and break errors, which may seriously affect the accuracy of your solution
- Errors may propagate and accumulate, leading to smaller accuracy
- Ill-conditioned problems and unstable methods have to be identified so that proper measures can be taken
- Symbolic math computations may be performed to solve certain equations algebraically, bypassing numerical errors, but this is not always possible.

Summary

- Numerical errors mar arise due to truncation, roundoff and break errors, which may seriously affect the accuracy of your solution
- Errors may propagate and accumulate, leading to smaller accuracy
- Ill-conditioned problems and unstable methods have to be identified so that proper measures can be taken
- Symbolic math computations may be performed to solve certain equations algebraically, bypassing numerical errors, but this is not always possible.

Summary

- Numerical errors mar arise due to truncation, roundoff and break errors, which may seriously affect the accuracy of your solution
- Errors may propagate and accumulate, leading to smaller accuracy
- Ill-conditioned problems and unstable methods have to be identified so that proper measures can be taken
- Symbolic math computations may be performed to solve certain equations algebraically, bypassing numerical errors, but this is not always possible.