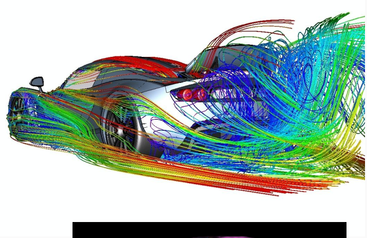
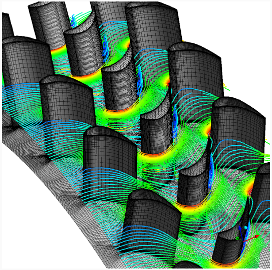
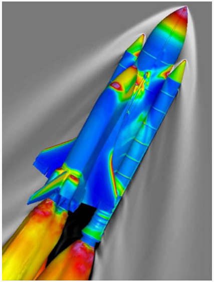
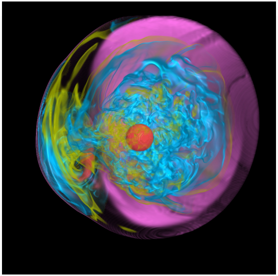
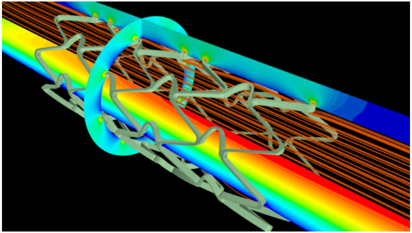
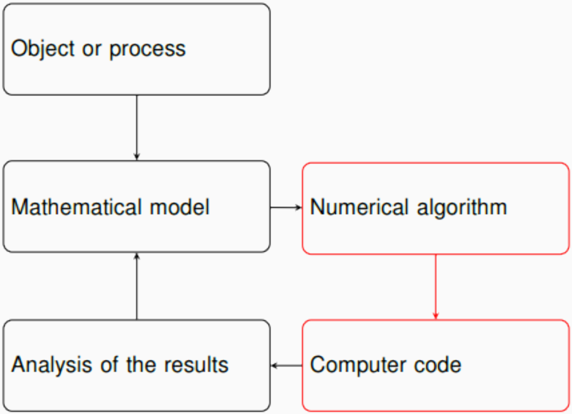
Introduction

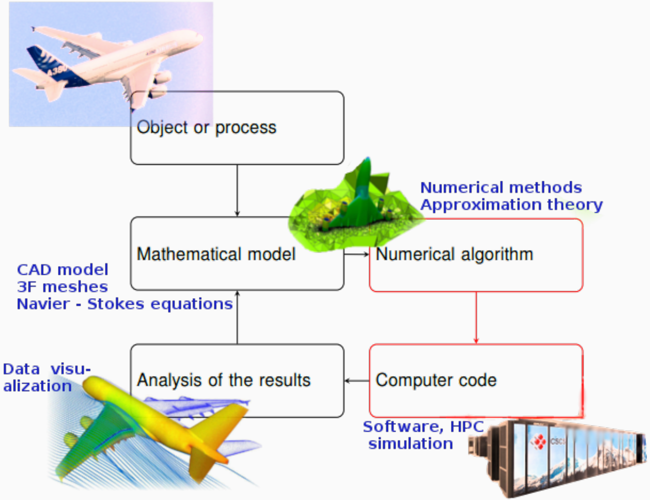
Motivation: numerical simulations

Motivation: mathematical modeling



Motivation: mathematical modeling

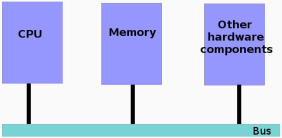


Tool: computers

Computer structure

- **Memory (ROM):** Stores data and instructions
- **CPU:** Elaborates instructions in a machine code
- **Bus:** Data communication system between components

One can use a computer with an operating system and software



Development

Software (including numerical solvers) are developed via **programming languages**, than can be of high or low level

Tool: programming languages

Low-level languages are non-portable, *i.e.* they are specific to each processor architecture, and have very few/low abstraction

- Machine code
- Assembly language

High-level languages are characterised by a strong abstraction

- Compiled: C, C++, C# , Fortran, Delphi, Rust
- Bytecode compiled: Java, Python, Common Lisp
- Interpreted: JavaScript, Mathematica, Python, Matlab®
- Source-to-source translated: Haxe, Dart, TypeScript

About MatLab®

Matlab®, from *MATrix LABoratory*, is a proprietary **software** containing ready-to-use **Toolboxes**. It is commonly used for

- Scientific computing
- Image processing
- Data analysis and post processing
- Symbolic computations and exact arithmetic,

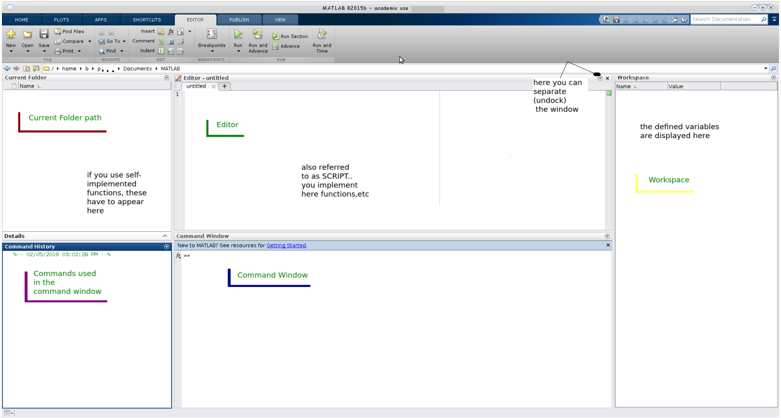
  though Mathematica is preferable for this...

SciLab
Open-source
Octave
GNU

It originates from a FORTRAN program and is notably good with

- Data elaboration with floating points
- Matrix based computations

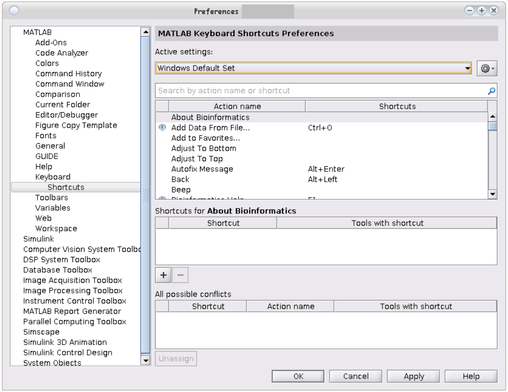
The Matlab® interface



The Matlab® shortcuts

Default Matlab® shortcuts are fairly different from the usual ones

Preferences → *Matlab* → *Keyboard* → *Shortcuts*



The Matlab® precision

Real numbers representation at the computer level

Matlab uses *floating-point numbers* $\mathbb{F}$, which is different from $\mathbb{R}$

```
>> 1/7
ans =
      0.1429
```

Matlab® computes on floats, in single or double precision

Single precision:

`1/3 ≈ 0.33333333`

Double precision:

`1/3 ≈ 0.3333333333333333`

Note that the numbers of decimals is not the same as the one prompted above: **precision** is different from **output format**

The Matlab® precision

Output formats

One can prescribe the output format that Matlab® should use for **showing** the results by using the command `format Type`

```
>> 2.105
ans =
    2.1050
>> format long
>> 2.105
ans =
    2.105000000000000
```

Type	
rat	1/7
short	0.1429
short g	0.14286
short e	1.4286e-01
long	0.142857142857143
long g	0.142857142857143
long e	1.428571428571428e-01

The Matlab® precision

Output formats

The choice of the output format **does not** impact the computer precision, which is either simple or double (float)

```
>> format short
>> 2.01
ans =
    2.0100
>> 2.01 + 0.000000002
ans =
    2.0100
>> format long
>> ans
ans =
    2.010000002000000
```

Type	
`rat`	1/7
`short`	0.1429
`short g`	0.14286
`short e`	1.4286e-01
`long`	0.142857142857143
`long g`	0.142857142857143
`long e`	1.428571428571428e-01

Theory: representation of floating-point numbers

How to represent any decimal number on the computer regardless its order of magnitude?

Theory: representation of floating-point numbers

How to represent any decimal number on the computer regardless its order of magnitude?

→ Using a floating-point representation

In base 2, only 23 or 52 (single or double precision) digits can be stored. Thus, one *shifts* the decimal point according to the order of magnitude.

Needs three digits to be represented accurrately

The exponent expresses the position of the coma (floating point)

$$0.145000 = 0.145 \times 10^{0}$$
$$0.000145 = 0.145 \times 10^{-3}$$

Needs six digits to be represented accurrately

Both need three digits and an exponent to be represented accurately

Assuming one can only store 4 digits, one can represent exactly 0.000001 but not 0.0010001. This is due to **machine precision**.

Floating-point representation in any base

In practice, a computer represents any (signed) floating-point number x on a basis β (usually 2) depending on its architecture

$$\begin{aligned} x &= 0.\underbrace{00\cdots\ 0}_{e \text{ times}}\ \underbrace{a_1 a_2 a_3 \cdots a_t}_{t \text{ significant digits}} \quad \text{in the basis } \beta \\ &= (-1)^s \times a_1 a_2 a_3 \cdots a_t \times \beta^{e-t} \\ &= (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j} \end{aligned}$$

Vocabulary

- The p first *significant digits* are $a_1 a_2, \cdots a_p$, where $a_1 \neq 0$
- The integer $m = a_1 a_2 \ldots a_t$ composed of all the t significant digits of x is the *mantissa*

Further reference: Quarteroni, Saleri, Gervasio - Scientific Computing with Matlab and Octave, IVth ed.

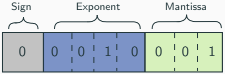
Theory: representation of floating-point numbers

At the computer level

A computer stores in memory any floating point number

$$x = (-1)^s \times a_1 a_2 a_3 \cdots a_t \times \beta^{e-t}$$

by assigning the sign s, each coefficient a_i, and the exponent to a contiguous allocation in memory.



Exemple of the representation of $0.25 = 1 \times 2^{-2}$ on 8 bits in base 2

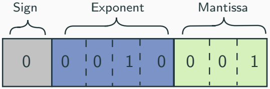
No. of Bits	Sign Bits	Exponent Bits	Mantissa Bits
8	1	4	3
16	1	6	9
32	1	8	23
64	1	11	52
128	1	15	112

The number of allocated storage spots per block type depends on the computer's architecture

Theory: representation of floating-point numbers

Range of representable values

As the number of significant digits, the exponents that can be used in are limited by the number of bits of its storage



Exemple of the representation of $0.25 \; = \; 1 \times 2^{-2}$ on 8 bits in base 2

Practically, the exponent e can take any integer value in $\{L, \cdots, U\}$, where L (*resp.* U) is the smaller (resp. bigger) number that can be expressed in base 2 on the number of allocated exponent's bits

Range of representable values: $x_{min} = \beta^{L-1}$

$$x_{max} = \beta^{U}(1 - \beta^{-t})$$

Theory: representation of floating-point numbers

Exercise

Knowing that a floating-point number is represented as

$$
\begin{aligned}
x &= (-1)^s \times a_1 a_2 a_3 \cdots a_t \times \beta^{e-t} \\
&= (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j}_{,}
\end{aligned}
$$

and assuming that $\beta = 2$, $t = 3$ and $e \in \{-4, \ldots, 3\}$, derive:

- the smallest representable number:
- the biggest representable number:
- the smallest representable absolute value of a number:
- the number of bits you need to store one number:

Theory: representation of floating-point numbers

Exercise

Knowing that a floating-point number is represented as

$$\begin{aligned} x &= (-1)^s \times a_1 a_2 a_3 \cdots a_t \times \beta^{e-t} \\ &= (-1)^s \sum_{j=1}^{t} a_j \beta^{e-j}, \end{aligned}$$

and assuming that $\beta = 2$, $t = 3$ and $e \in \{-4, \dots, 3\}$, derive:

- the smallest representable number:
  -7
- the biggest representable number:
  7
- the smallest representable absolute value of a number:
  $\frac{1}{32}$
- the number of bits you need to store one number:
  $s : 1, a_i : t - 1 = 2, e : 3 \Rightarrow 6$ *bits*

Space of floating-point numbers

The set of a floating-point numbers $\mathbb{F}$ that one can represent on a computer is characterised by $\mathbb{F}(\beta, t, L, U)$,

where β is the basis, t the number of significant digits, and L, U are integers giving the range's bounds of the used exponents

Space used in Matlab®

t is 52, but we can consider 53

Numbers are represented on $\mathbb{F} = \mathbb{F}(2, 53, -1021, 1024)$ and are stored on 8 bytes ($=$ 64 bits, exponent on 11 bits: *double precision*)

```
>> realmin
ans =
    2.2250738585072e-308
```

```
>> realmax
ans =
    1.79769313486232e+308
```

Storing in Matlab

Exercise

Represent the following floating points in base 2, or explain why this is not possible to be done exactly.

a) 15 1111

b) 0.5 0.1

c) 0.1 no, infinit

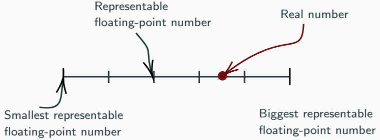
d) $1/3$ no, infinit

e) $1/256$ 0.00000001

Definition

A *round-off* error is generated when a real number is replaced with a floating-point number

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_M$$



with $\epsilon_M = \beta^{1-t}$ the *machine epsilon*

In Matlab®

The machine epsilon is given by $\epsilon_M = 2^{-52} \sim 2.22 \cdot 10^{-16}$

```
>> eps
ans =
     2.22044604925031e-16
```

Good to know

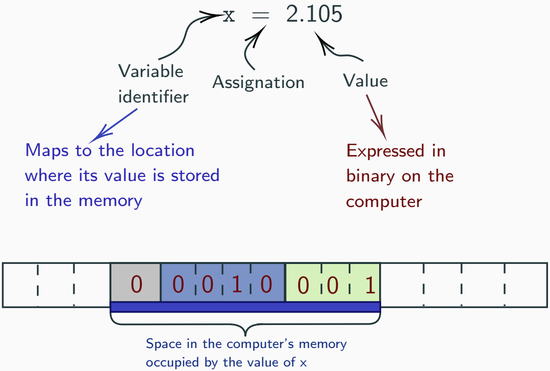
Sources of numerical errors

1. Fixed-size data storage format on computers
   (e.g. round-off errors)
2. Errors in the initial or input data
   (e.g. from experimental measurements)
3. Incompleteness of mathematical model
4. Approximate methods to solve the equations of the mathematical model
5. ...

Getting started

Reusing values

The values that will be reused are stored in *variables*



Here, `x` is a variable, and we *assigned* the *value* 2.105 to it

Scalar assignment

Type in the command line:

```
>> a = 2.45
>> A = 3.1
>> a
```

Note: >> is called a *prompt*

Scalar assignment

Type in the command line:

```
>> a = 2.45
>> A = 3.1
>> a
```

```
ans =
      2.4500
```

→ Distinction between capital and small letters (case sensitivity)

Note: >> is called a *prompt*

Scalar assignment

Type in the command line:

```
>> A = 7.2
>> A
```

Scalar assignment

Type in the command line:

```
>> A = 7.2
>> A
```

```
ans =
      7.2000
```

→ Watch out! The variable A has been *overwritten*
(the previous value 3.1 has been replaced by 7.2)

Comma and semicolon

Use of the comma

```
>> a= 1.2,
a =
    1.2000
>> a= 1.7, b=2.45
a =
    1.7000
b =
    2.4500
```

- Useful to **separate multiple commands** on a same line
- No difference with/without the comma at the end of the line

Comma and semicolon

Use of the semicolon

```
>> a= 1.2;
>> a= 1.7; b=2.45
b =
    2.4500
>> a;
>> a
a =
    1.7000
```

- Useful to **separate multiple commands** on a same line
- Suppresses the **output visualisation**

Exploring memory

To check what variables are stored in the memory:

```
>> who
Your variables are:
a  A  ans
```

To know the stored variables, their dimension, the dimension of their memory storage and their typology:

```
>> whos
Name       Size            Bytes  Class     Attributes

A          1x1                 8  double
a          1x1                 8  double
ans        1x1                 8  double
```

Memory management

Clearing memory

Delete variable A from workspace:

```
>> clear A
```

Delete **all variables** from workspace:

```
>> clear
>> clear all
```

Clean up the visualization window:

```
>> clc
>> home
```

Best practice

Always clean your workspace and visualization window before starting a new exercise

Getting help about instructions

To see how to use an instruction, ex. called `instruction`:

`help instruction`

Example:

```
>> help format
format Set output format.
    format with no inputs sets the output format to the
    default appropriate for the class of the variable.
    For float variables, the default is format SHORT.

    format does not affect how MATLAB computations
    are done. Computations on float variables, ...
```

Getting started: trying the command line

Exercise

Type in the command line what follows and see what happens

```
>> doc sin
>> sin <F1>
>> si <TAB>
```

Exercise

Type in the command line what follows and explain the results

```
>> 32 * 3 + 5
>> 32 * (3 + 5)
>> sin (3 * pi)
>> pi
```

Expressions and predefined variables

Arithmetic operators

Arithmetical operators	`^ * /`	power, multiplication, division
	`+ -`	addition, subtraction
Equivalence operators	`== ~=`	equivalent to, unequal to
	`< <= > >=`	less, greater than
Logic operators	`& | ~`	and, or, not

The order in which the operations are done is called *precedence*, whose rule is given by the previous table (decreasing order)

- Two operations with a same priority rank are done from left to right
- One can specify the desired precedence by using brackets ()

```
>> 32 * 3 + 5
>> 32/2*3
>> 32 * (3 + 5)
```

To get more information: `doc 'operator precedence'`

Numerical expressions

Expressing integer and decimal numbers

```
>> 1
```

```
>> 0.23
```

```
>> .23
```

Expressing decimal numbers in their exponential form

```
>> 23*10^(-2)
```

```
>> 23e-2
```

Expressing complex numbers

```
>> 5+4i
```

```
>> 5+4j
```

Built-in functions

Predefined functions exist in Matlab®

sin	sqrt	log	floor
cos	abs	log2	ceil
tan	exp	log10	round

Call them with `functionName(parameter1, parameter2, ...)`
and know their arguments' type by typing `help functionName`

Example:

```
>> help sin    % You learn that the angle is in radians
>> sin(3 * pi)
```

Expressions

Predefined variables

Some commonly used values are stored in *predefined variables*

`ans`	result of the last computation
`pi`	$\pi = 3.14..$
`eps`	machine precision
`i,j`	$\sqrt{-1}$
`nan`	not a number (ex. as result of $0/0$)
`inf`	infinity (ex. as result of $1/0$)
`realmin, realmax`	smallest and biggest floating-point numbers

Best practice

Do not overwrite any of the predefined functions or variables!

Further *keywords* should not be used as variable names. The list of keywords is accessible with the command `>> iskeyword`