# Numerical Algorithms (MU4IN910)

# Lecture 1: Introduction to MATLAB and to floating-point arithmetic

Stef Graillat

Sorbonne Université

# Format du cours

**Lecturer**: Stef Graillat (office 26-00/313)
`stef.graillat@sorbonne-universite.fr`

**Assignments, exams and grading**

- 1 exam (50 %) + practicals (50 %)

**Schedule**

- 5 lectures: wednesday 8h30-10h30
- 5 tutorials: wednesday 10h45-12h45
- 5 practicals as a personal work in pairs
- Practicals due one week after
  → report written in LaTeX and submitted in one pdf file on Moodle

**Website**: `http://www-pequan.lip6.fr/~graillat/teach/anum/`

**Moodle**: `https://moodle-sciences-22.sorbonne-universite.fr/course/view.php?id=4479`

# Aims of the course

**Goals**:

- Mathematical concept: mathematical definition of concepts and quantities
- Algorithm: how to efficiently calculate these quantities on a computer (via the use of MATLAB)?
- Problem solving: use concepts and algorithms to solve real-life problems

# General course outline

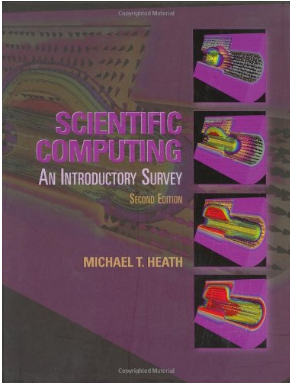
## Numerical Algorithms

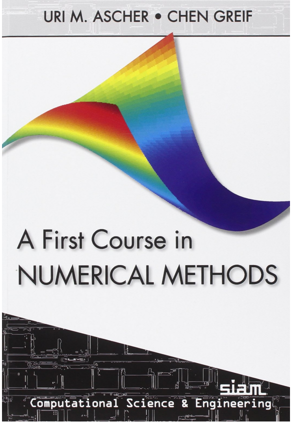
1. Introduction to MATLAB and floating-point arithmetic
2. Matrix computation
3. Introduction to numerical optimization
4. Nonlinear equations

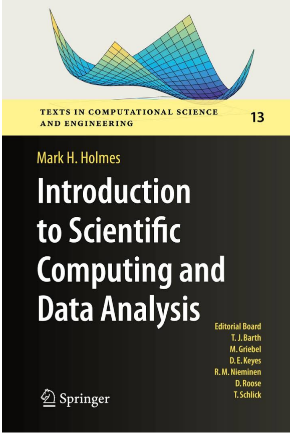
# Main references

- **A First Course in Numerical Methods, Uri M. Ascher, Chen Greif, SIAM, 2011**
- **Scientific Computing, An Introductory Survey, Michael T. Heath, McGraw-Hill, 2002**
- **Scientific Computing with Case Studies, Dianne P. O'Leary, SIAM, 2009**
- **Introduction to Scientific Computing and Data Analysis, Mark H. Holmes, Springer, 2016**
- Mathématiques appliquées L3, sous la direction de Jacques-Arthur Weil et Alain Yger, Pearson, 2009
- Numerical Computing with MATLAB, Cleve Moler, SIAM, 2004
- MATLAB Guide, Desmond J. Higham, Nicholas J. Higham, 3e édition, SIAM, 2017
- Scientific Computing, An Introduction using Maple and MATLAB, Walter Gander, Martin Gander, Felix Kwok, Springer, 2014
- Numerical Recipes. The Art of Scientific Computing, William Press, Saul Teukolsky, William Vetterling et Brian Flannery, 3rd Edition, Cambridge University Press, 2007

# Main references



# Scope

The concepts seen in this course can be applied to:

- robotics
- signal processing
- image processing
- finance
- biology
- etc.

# 1. Floating-point arithmetic

# Can we count to 6 with a computer?

| | | |
|---|---|---|
| $2-1$ | | 1.000000000000000 |
| $\left(\dfrac{1}{\cos(100\pi+\pi/4)}\right)^2$ | | 2.000000000000111 |
| $3\dfrac{\tan(\arctan(10000))}{10000}$ | | 2.999999999997162 |
| $\left(\left(\cdots\left(\sqrt{\sqrt{\cdots\sqrt{4}}}\right)^2\cdots\right)^2\right)^2$ | (20 times) | 4.000000000629434 |
| $5\times\left\{\dfrac{(1+e^{-100})-1)}{(1+e^{-100})-1)}\right\}$ | | `NaN` |
| $\dfrac{\log(e^{6000})}{1000}$ | | `Inf` |

# Floating-point numbers

A normalized floating-point number $x \in \mathbb{F}$ is a number which is written in the form

$$x = \pm \underbrace{x_0.x_1 \dots x_{p-1}}_{mantissa} \times b^{e}, \quad 0 \le x_i \le b-1, \quad x_0 \neq 0$$

$b$: the base, $p$: precision, $e$: exponent satisfying $e_{\min} \le e \le e_{\max}$

Machine precision $\epsilon = b^{1-p}$, $|1^+ - 1| = \epsilon$

Approximation of $\mathbb{R}$ by $\mathbb{F}$, rounding $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$
Let $x \in \mathbb{R}$ then

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le \mathbf{u}.$$

Unit roundoff $\mathbf{u}$ is equal to $\mathbf{u} = \epsilon/2$ for round to nearest

# Standard model of floating-point arithmetic

IEEE 754 standard (1985,2008,2019)

- The arithmetic operations ops $(+, -, \times, /, \sqrt{\ })$ are performed as if they were calculated in infinite precision and then rounded off
- Default: rounded to nearest

| Type | Size | Mantissa | Exponent | Unit roundoff | Interval |
|---|---|---|---|---|---|
| binary32 (simple) | 32 bits | 23+1 bits | 8 bits | $\mathbf{u} = 2^{-24} \approx 5,96 \times 10^{-8}$ | $\approx 10^{\pm 38}$ |
| binary64 (double) | 64 bits | 52+1 bits | 11 bits | $\mathbf{u} = 2^{-53} \approx 1,11 \times 10^{-16}$ | $\approx 10^{\pm 308}$ |

Let $x, y \in \mathbb{F}$,

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \quad |\delta| \leq \mathbf{u}, \quad \circ \in \{+, -, \cdot, /\}$$

# Exceptions

The arithmetic is "closed ": each operation returns a result.

| Exception | Results |
|---|---|
| Invalid operation | NaN (Not a Number) |
| Overflow | $\pm\infty$ |
| Divide by zero | $\pm\infty$ |
| Underflow | Denormalized numbers |
| Inexact | correctly rounded result |

NaN is generated by operations such as $0/0$, $0 \times \infty$, $\infty/\infty$, $(+\infty) + (-\infty)$ and $\sqrt{-1}$.

Infinite symbols satisfy $\infty + \infty = \infty$, $(-1) \times \infty = -\infty$ and $(\text{fini})/\infty = 0$.

# Accuracy of the computation

At each rounding, we lose a priori a bit of accuracy, we talk about rounding error.

Even if an isolated operation returns the best possible result (rounding of the exact result), a series of calculations can lead to large errors due to the accumulation of rounding errors.

The two main sources of rounding error during calculations are cancellation and absorption.

# Example of cancellation

Let $f(x) = (1 - \cos(x))/x^2$, then $0 \leq f(x) < 1/2$ for all $x \neq 0$.
With $x = 1.2 \times 10^{-5}$, the cosine rounded to 10 significant digits is equal to

$$c = 0.9999\,9999\,99,$$

so

$$1 - c = 0.0000\,0000\,01$$

Therefore $(1-c)/x^2 = 10^{-10}/1.44 \times 10^{-10} =$ 0.6944 ... !!!!

However, the subtraction $1 - c$ is exact.

To avoid the cancellation, rewrite $f$ in the form

$$f(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2$$

# Example of absorption

We calculate numerically, for large values of $N$, the sum:

$$\sum_{i=1}^{N} \frac{1}{i}$$

Results of a C program (floating-point single precision) on a Pentium 4 processor:

| order | $N$ | | | |
|---|---|---|---|---|
| | $10^5$ | $10^6$ | $10^7$ | $10^8$ |
| exact | 1.209015e+01 | 1.439273e+01 | 1.669531e+01 | 1.899790e+01 |
| $1 \to N$ | 1.209085e+01 | 1.435736e+01 | 1.540368e+01 | 1.540368e+01 |
| $N \to 1$ | 1.209015e+01 | 1.439265e+01 | 1.668603e+01 | 1.880792e+01 |

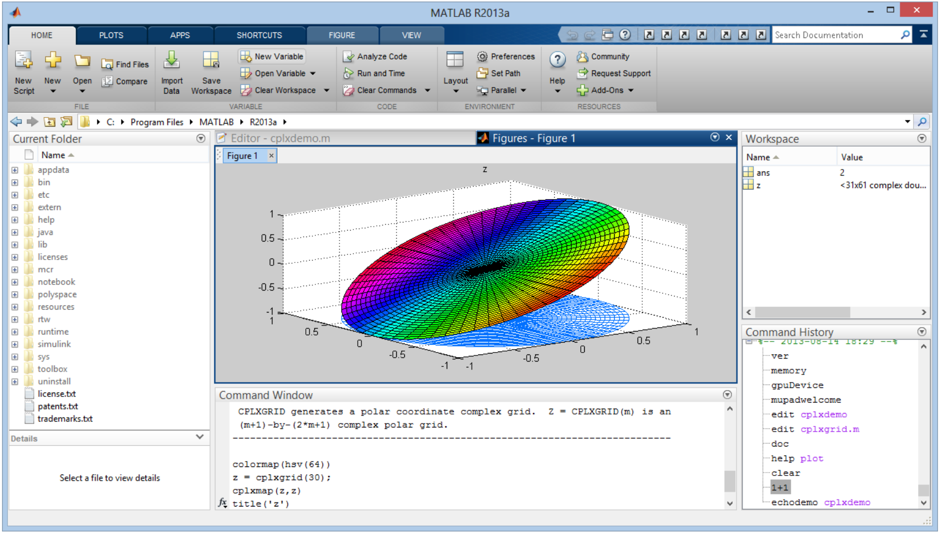
# 2. Introduction to MATLAB

# MATLAB

- MATLAB = MATrix LABoratory
- a programming language and a development environment
- MATLAB was designed by Cleve Moler in the late 1970s
- MATLAB is supplemented by multiple toolboxes
- MATLAB language supports OOP
- Interaction possible with C and Fortran languages
- for help on a `command` command, use `help command` or `doc command`
- to write comments %

Reference: MATLAB Guide, Desmond J. Higham, Nicholas J. Higham, 3rd edition, SIAM, 2017

# MATLAB



MATLAB R2013a
HOME
PLOTS
APPS
SHORTCUTS
FIGURE
VIEW
Search Documentation
New Script
New
Open
Find Files
Compare
Import Data
Save Workspace
New Variable
Open Variable
Clear Workspace
Analyze Code
Run and Time
Clear Commands
Layout
Preferences
Set Path
Parallel
Help
Community
Request Support
Add-Ons
FILE
VARIABLE
CODE
ENVIRONMENT
RESOURCES
C: ▸ Program Files ▸ MATLAB ▸ R2013a ▸
Current Folder
Name
appdata
bin
etc
extern
help
java
lib
licenses
mcr
notebook
polyspace
resources
rtw
runtime
simulink
sys
toolbox
uninstall
license.txt
patents.txt
trademarks.txt
Details
Select a file to view details
Editor - cplxdemo.m
Figures - Figure 1
Figure 1
z
Workspace
Name
Value
ans
2
z
<31x61 complex dou...
Command History
ver
memory
gpuDevice
mupadwelcome
edit cplxdemo
edit cplxgrid.m
doc
help plot
clear
1+1
echodemo cplxdemo
Command Window
CPLXGRID generates a polar coordinate complex grid. Z = CPLXGRID(m) is an
(m+1)-by-(2*m+1) complex polar grid.
colormap(hsv(64))
z = cplxgrid(30);
cplxmap(z,z)
title('z')

# MATLAB

- MATLAB was created in the 1970s by Cleve Moler, professor of mathematics at the University of New Mexico
- MATLAB was created from the Fortran LINPACK and EISPACK libraries
- MATLAB then evolved, integrating the LAPACK library in 2000
- There are free alternatives to MATLAB such as GNU Octave, FreeMat and Scilab
- The current version of MATLAB is MATLAB R2022b (version 9.13)

# Vectors and matrices in MATLAB

MATLAB variables are mainly vectors and matrices

- Vectors:

  - row vector

```
>> format compact
>> x = [1.1 10.1 100.1]
x =
   1.1000 10.1000 100.1000
```

  - column vector

```
>> x = [1.1; 10.1; 100.1]
x =
    1.1000
   10.1000
  100.1000
```

# Vectors in MATLAB

- Display and assignment

```
>> x = [1.1; 10.1; 100.1];
>> x
x =
    1.1000
   10.1000
  100.1000
>> x(3) = -1.1
x =
    1.1000
   10.1000
   -1.1000
```

The vectors are indexed from 1 (and not 0 as in C)

# Vectors in MATLAB

- Transpose of a vector

```
>> x = [1.1 10.1 100.1]'
x =
   1.1000
  10.1000
 100.1000
```

- To enter a vector or a command occupying more than one line

```
>> x = [0 .05 .10 .15 .20 .25 .30 .35 .40 .45 .50 ...
   .55 .60 .65 .70 .75 .80 .85 .90 .95 1];
```

- Length of a vector

```
>> length(x)
ans =
   21
```

# Vectors in MATLAB

- Vector of any size (by default vectors are row vectors)

```
n = 20;
h = 1/n;
for k=1:n
   x(k) = k*h;
end
```

- Initialization of a column vector

```
x = zeros(n,1);
```

- colons. The notation `a:b` denotes the row vector going from `a` to `b` in steps of 1 while for `a:s:b` the step is `s`

```
>> x = 1:5
x =
   1 2 3 4 5
>> x = 4:-1:0
x =
   4 3 2 1 0
```

# Vectors in MATLAB

- `>> x = 0:0.05:1;   % vector with 21 components.`
  `>> x = 0.05*(0:20) % another way to generate the same vecto`
- Access parts of a vector
  `x(1:4)` extract the first 4 elements of `x`
- `linspace(a,b,n)` produces a row vector with $n$ components which divides $[a, b]$ in $n-1$ equal intervals.
  `x = linspace(0,1,21);`

# Matrices in MATLAB

The power of MATLAB comes from its matrix operations

- fast
- and accurate

2 ways to enter matrices

```
>> A = [1 2 3
         2 4 7
        -1 0 5]
A =
    1 2 3
    2 4 7
   -1 0 5
>> a = [1 2 3; 1 4 8; 3 -1 0]
a =
    1 2 3
    1 4 8
    3 -1 0
```

# Matrices in MATLAB

Special functions for creating matrices

- `zeros(m,n)` returns a matrix of 0 of size $m \times n$

```
>> A = zeros(2,4)
A =
    0  0  0  0
    0  0  0  0
```

- `ones(m,n)` returns a matrix of 1 of size $m \times n$

```
>> A = ones(3)
A =
    1  1  1
    1  1  1
    1  1  1
```

# Matrices in MATLAB

- `eye(n)` returns the identity matrix of size $n$

```
>> A = eye(3)
A =
    1 0 0
    0 1 0
    0 0 1
```

- Solving a linear system $Ax = b$

```
>> A = [1 2 3; 2 4 7; -1 0 5];
>> b = [1 1 1]';
>> x =A\b
x =
     -6
      5
     -1
```

# Matrices in MATLAB

- Another use of colons:

```
>> C = A([1 3],:)
C =
      1      2      3
     -1      0      5
```

- Vectorized functions

  We want to evaluate a function on a vector of value $x_i$ :

  $a = x_1 < x_2 < \cdots < x_n = b$.

  Standard functions take vectors as arguments and return a vector.

```
n=21;
x = linspace(0,2*pi,n);
y = cos(x);
```

# Save your work

- In general, one write the list of commands in a text file (via the `edit` editor integrated into MATLAB) or via a favorite text editor
- To save variables, use the `save` function. Saving the variables `A` and `b` in the file `svar.mat` can be done by:

  ```
  save svar A b
  ```

  One reload the variables by

  ```
  load svar
  ```

# Help

- type `help command`.

```
>> help length
 LENGTH   Length of vector.
    LENGTH(X) returns the length of vector X.  It is
    equivalent to MAX(SIZE(X)) for non-empty arrays
    and 0 for empty ones.
```

- to have it in graphic mode, type `doc`

```
>> doc length
```

# M-files

2 types of file (which have the extension `.m`):

- script M-files: neither input nor output and uses workspace variables
- function M-files: contains a function that accepts input arguments and returns output arguments and internal variables are local to the function

Example (to save in a file `sumprod.m`):

```
function [s,p]= sumprod(x)
   n = length(x);
   s=0;
   p=1;
   for i=1:n
      s = s + x(i);
      p = p*x(i);
   end
```

# M-files

Structure of an M-files function

1. the keyword `function`
2. list of output arguments (in brackets `[ ]` if there are several ones)
3. the symbol =
4. the name of the function (which must be the same as the name of the `.m` file)
5. the list in parenthesis of the entries
6. the body of the function

To edit the files, type `edit`

Useful commands: `dir`, `ls`, `cd`, `type`, `lookfor`, `path`

# Display

- Numeric format: command `format` to display fixed or floating-point numbers

```
>> format short, pi^4 % fixed, 5 digits
ans =
   97.4091
>> format shortE, pi^4 % float, 5 digits
ans =
   9.7409e+001
>> format long, pi^4 % fixed, 15 digits
ans =
   97.40909103400242
>> format longE, pi^4 % float, 15 digits
ans =
   9.740909103400242e+001
```

# Display

- Display of strings
  - the command `disp` allows to display a string or a variable

```
>> x=3;
>> disp(x);
     3
>> disp('test');
test
```

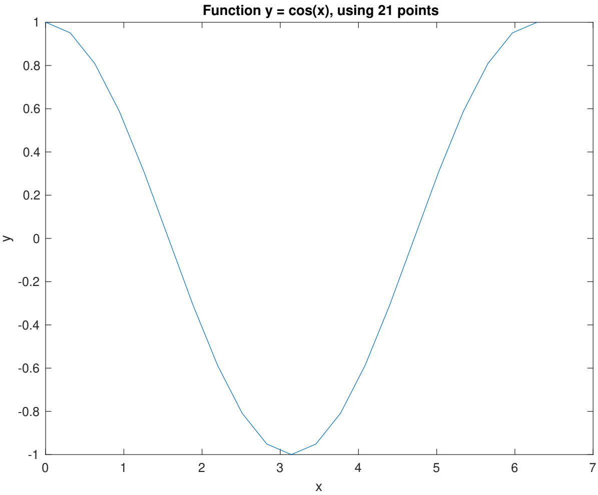
  - the command `fprintf` allows to display strings and numbers (similar to the `printf` command in C)

# Graphics

Plot of a function $y = f(x)$ over an interval $[a, b]$

```
n = 21;
x = linspace(0,2*pi,n);
y = cos(x);
plot(x,y)
title('Function y = cos(x), using 21 points')
xlabel('x')
ylabel('y')
```

# Graphics



Function y = cos(x), using 21 points
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
y
0
1
2
3
4
5
6
7
x

# Control structures and tests

- loop `for`

```
for variable = expression
   instructions
end
```

- loop `while`

```
while expression
   instructions
end
```

- test `if`

```
if expression
   instructions
end
```

```
if expression
   instructions
else
   instructions
end
```

# Control structures and tests

- relational operations

| | |
|---|---|
| `==` | equal |
| `~=` | different |
| `<` | strictly lower |
| `>` | strictly greater |
| `<=` | less or equal |
| `>=` | greater or equal |

- logical operations

| | |
|---|---|
| `&` | and |
| `\|` | or |
| `~` | not |

# Symbolic Math Toolbox

- The toolbox contains the kernel of MuPAD and MATLAB functions which communicate with this kernel
- New type: objects `sym` created by the `sym` and `syms` commands
- Create a symbolic variable `x`

```
>> syms x
```

# Symbolic Math Toolbox

- Manipulation of symbolic expression in x

```
>> f = 1/(1+x^2)
f =
   1/(1+x^2)

>> g = int(f) % integration
g =
   atan(x)

>> diff(g) % differentiation
ans =
   1/(1+x^2)

>> syms a
>> y = solve(f-a) % solves f(x)-a=0
y = [ 1/a*(-a*(-1+a))^(1/2)]
    [ -1/a*(-a*(-1+a))^(1/2)]
```

# Symbolic Math Toolbox

- Multiprecision arithmetic: function vpa

```
>> digits % by default
Digits = 32
>> vpa('sqrt(2)')
ans =
1.4142135623730950488016887242097
>> digits(50)
>> vpa('sqrt(2)')
ans =
1.4142135623730950488016887242096980785696718753769
```

- Exact arithmetic: one manipulate symbolic expressions

```
>> z = sym('sqrt(2)')
z =
sqrt(2)
>> z^2-2
ans =
0
```

# Matrix storage

For programming efficiently algorithms on matrices, it is very important to know how matrices are stored in memory!

Example: matrix-vector product

Given a matrix $A$ of size $m \times n$ and a vector $x$ of length $n$, one wants to compute $y = Ax$

- The vector $y$ is defined by dot products between the lines of $A$ and $x$

$$y_i = A(i,:)x$$

```
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    for j=1:n,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

- We can express $Ax$ using the columns of $A$ instead

$$Ax = x_1 A(:,1) + x_2 A(:,2) + \cdots + x_n A(:,n)$$

which is equivalent to compute the transpose of $(Ax)^T$

```
[m,n]=size(A);
y = zeros(m,1);
for j=1:n,
    for i=1:m,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

# Matrix storage

Both algorithms perform $mn$ multiplications and $mn$ additions!

- The computer stores the information in memory pages
- Part of the information is stored in caches
- hat does not fit in the cache is stored in the main memory (slower to access)
- Finally, what does not fit in main memory is stored on the hard disk

To use data, the page containing the data must be moved to cache (therefore, other data must be cleared from the cache), these pages being at best in the RAM (most of the time when the calculations are reasonable) or at worst in the hard disk (for example for the multiplication of square matrices of several million lines).
For an algorithm to be effective, it is necessary to limit the number times a page is moved to cache!

# Matrix storage

Question to ask when one wants to multiply a matrix and a vector:
What is the algorithm to use to make the algorithm efficient?

The right question to ask is in fact:

Are the matrices ordered by row or by column?

| Language | Storage scheme |
|---|---|
| C, C++ | by row |
| Fortran | by column |
| Java | by row |
| MATLAB | by column |

# Basic tools for matrix manipulations: the BLAS

There are tasks that are used in almost all matrix problems!

Libraries have been developed to prevent programmers do not have to reprogram them each time

The Basic Linear Algebra Subroutines or BLAS are now available for almost all programming languages

The BLAS are divided into levels. Level-$k$ BLAS need $\mathcal{O}(n^k)$ operations (here $n$ is the size of the vectors or matrices)

# Basic tools for matrix manipulations: the BLAS

## Example 1

- *Level-1 BLAS: vector operations*
  1. *sscal computes $ax$ where $a$ is a scalar and $x$ is a vector*
  2. *saxpy computes $ax + y$*
  3. *sdot computes $x^* y$*
- *Level-2 BLAS: matrix-vector operations*
  1. *matrix-vector product*
  2. *solution of linear systems involving triangular matrix*
- *Level-3 BLAS: matrix-matrix operations*
  1. *matrix-matrix product*
  2. *solution of multiple linear systems involving triangular matrix*

When a BLAS exist for a task that is needed, it is a good idea to use it on the algorithm

MATLAB automatically uses the BLAS. In other languages, one need to call specific subroutines to do it.

# MATLAB and Maple at PPTI

- MATLAB is available at PPTI. It is MATLAB R2019b (version 9.7) with the Symbolic Math Toolbox.

  To run MATLAB, type in a terminal
  `/usr/local/Matlab-2019/R2019b/bin/matlab`

- Maple is also available at PPTI. It is Maple 2020.

  To run Maple, type `/usr/local/maple2020/bin/xmaple`

# Agreement with MATLAB and Maple at SU

- SU now makes available to you (for staff and students) MATLAB
  http://logiciels.upmc.fr/fr/marches_conclus_par_l_upmc/matlab.html
- Whether you are a teacher, researcher, student, you can have a Maple license on your personal commuter
  http://logiciels.upmc.fr/fr/marches_conclus_par_l_upmc/maple.html

# Conclusion

To get an overview of MATLAB functionality, type `demo`

# Representation of real numbers

## Examples

**Integer part - fractional part** In base $\beta$ a number can be represented by $A = \sum_{-\infty}^{+\infty} a_i \beta^i$

**Rational numbers** The fractional part $A_f = \sum_{-\infty}^{-1} a_i \beta^i$ is periodic from a certain rank.

$\frac{237}{315} = 0.75238095238095\overline{23809}$

**Real numbers** The fractional part can be aperiodic.

$\sqrt{2} = 1.4142135623730950488016887242 09\ldots$

$\sqrt{2} = 1 + \frac{1}{2} - \frac{1}{8} + \frac{1}{16} - \frac{5}{128} \ldots + (-1)^{n+1} \frac{\binom{2n}{n}}{(2n-1)2^{2n}}$

$e = 1 + \sum_{n=1}^{\infty} \frac{1}{n!} = 2.718281828459045 23536\ldots$

$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 3.14159265358979323 8\ldots$

# Scientific notation

- Representation of normalized numbers

$$X = (-1)^s \times x_0, x_1 x_2 \ldots x_n \times \beta^e = (-1)^s \times \sum_{i=0}^{n} x_i \times \beta^{-i+e}$$

  with $x_0 \neq 0$ (except if $X = 0$).
- The number of digits is fixed: notion of approximate value.
- $\frac{-237}{315} = (-1)^1 \times 7.523809523 \times 10^{-1}$
  $\sqrt{2} = (-1)^0 \times 1.414213562 \times 10^0$

# IEEE 754 standard

- A number is represented in single precision (32 bits) or in double precision (64 bits) as follows:

| | | |
|---|---|---|
| s(1) | e =exp biased (8 or 11) | ... mantissa (23 or 52) |

- If $e \neq 0$ and $e \neq 255(11111111)$ or $e \neq 2047(11111111111)$

$$(-1)^s \times 2^{(\mathrm{e})-\mathrm{b}} \times 1,\ldots\text{mantissa}\ldots$$

  The bias is $b = 127$ (01111111) in single precision, and $b = 1023$ (01111111111) in double precision.

- if, exp biased = 0:

$$(-1)^s \times 2^{1-\mathrm{bias}} \times 0,\ldots\text{mantissa}\ldots$$

- If $e = 255(11111111)$ (or $e = 2047(11111111111)$ ):
  then the number reprensents infinity if the mantissa is 0 or a NaN (not a number) if not.

# Remarks

- If $e \neq 0$ and $e \neq 255(11111111)$ or $e \neq 2047(11111111111)$ then the number is said tobe normalized and the significand has an implicit 1 at the beginning
- Otherwise, the number is denormalized. This makes it possible to represent very small numbers
- Remark: 0 has a sign

# Examples

- 1 = 0 01111111 00000000000000000000000
- 2 = 0 10000000 00000000000000000000000
- 3 = 0 10000000 10000000000000000000000
- 9 = 0 10000010 00100000000000000000000
- 1/2 = 0 01111110 00000000000000000000000
- 1/3 = 0 01111101 01010101010101010101010 101010101…
- 1/10 = 0 01111011 10110011001100110011001 100110011…

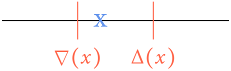
# Special values

| Number | Representation in double precision (hexadecimale) | Decimal value |
|---|---|---|
| $+0$ | 00000000 00000000 | 0.0 |
| $-0$ | 80000000 00000000 | $-0.0$ |
| 1 | 3FF00000 00000000 | 1.0 |
| 2 | 40000000 00000000 | 2.0 |
| Normalized maximum | 7FEFFFFF FFFFFFFF | $1.7976931348623157e+308$ |
| Normalized positive minimum | 00100000 00000000 | $2.2250738585072014e-308$ |
| Denormalized maximum | 000FFFFF FFFFFFFF | $2.2250738585072009e-308$ |
| Denormalized positive minimum | 00000000 00000001 | $4.9406564584124654e-324$ |
| $-\infty$<br>$+\infty$ | FFF00000 00000000<br>7FF00000 00000000 | $-\infty$<br>$+\infty$ |
| $NaN$ | 7FF xxxx...xxxxxxxx | Not A Number |

# IEEE 754 standard: rounding

- If $x$ and $y$ are 2 representable numbers, then the result of an operation $res = x \odot y$ is not, in general, a representable number.
- For example, in base $B = 10$, the number $1/3$ is not representable with a finite number of digits.
- It is necessary to round the result, that is to say to return one of the closest representable numbers.

# IEEE 754 standard: rounding modes




The norm proposes 4 rounding modes:

- rounding toward $+\infty$ denoted $\Delta(x)$: return the smallest floating-point number greater or equal the exact result $x$
- rounding toward $-\infty$ denoted $\nabla(x)$: return the largest floating-point number less or equal the exact result $x$
- rounding toward 0, denoted $\mathcal{Z}(x)$: return $\Delta(x)$ for negative numbers and $\nabla(x)$ for positive numbers
- rounding to the nearest, denoted $\circ(x)$: return the nearest floating-point number of the exact result $x$ (breaks ties by rounding to the nearest even floating-point number)

The 3 first rounding modes are called directed rounding modes.

# IEEE 754 standard: correct rounding

Let $x$ and $y$ be two representable number, $\odot$ be on operations $+, -, \times, /$ and $\diamond$ a rounding mode.

The standard requires that the result of the computation $x \odot y$ be equal to $\diamond(x \odot_{exact} y)$. The result must be similar to the one obtain by computing with infinite precision and then rounding this result.

Similar for square root.

This property is called correctly rounding.

*The standard describes an algorithm for addition, subtraction, multiplication, division and square root and requires that the implementation produces the same result as those algorithms.*

# IEEE 754 standard: comparisons

The standard requires the comparison to be exact and not to overflow.

The specified comparison in the standard are:

- equality
- greater than
- less than

The sign of zero is not taken into account.

In case of a comparison with a NaN, the comparison returns False.
More precisely in case of equality: if $x$ = NaN then $x = x$ returns False and $x \neq x$ returns True (equality is not reflexive but it is a way to detect a NaN)

# IEEE 754 standard: flags

No calculation should hinder the proper functioning of the machine. A mechanism with 5 flags makes it possible to inform the system about the behavior of operations:

INVALID operation — the result by default is a NaN

DIVIDE by ZERO — the result is $\pm\infty$

OVERFLOW — overflow toward $\infty$: the result is either $\pm\infty$ or the greatest floating-point number (in absolute value) depending on the sign of the exact result and the rounding mode

UNDERFLOW — overflow toward 0: the result is either $\pm 0$ or a subnormal

INEXACT — inexact result: raised when the exact result is not representable exactly. Returns the correctly rounded result by default.

A flag, when raised, stay raised until there is a reset by the user (sticky flags). They can be read and write by the user.