



**POLITECNICO**  
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE  
E DELL'INFORMAZIONE

# Introduction to MATLAB

Calcoli di Processo dell' Ingegneria Chimica

---

**Timoteo Dinelli, Marco Mehl**

20<sup>th</sup> of September 2024,

27<sup>th</sup> of September 2024,

3<sup>rd</sup> of October 2024.

Department of Chemistry, Materials and Chemical Engineering, G. Natta. Politecnico di Milano.

email: [timoteo.dinelli@polimi.it](mailto:timoteo.dinelli@polimi.it)

email: [marco.mehl@polimi.it](mailto:marco.mehl@polimi.it)

# Contacts and Informations

Professor Marco Mehl

**e-mail:** marco.mehl@polimi.it

**Ufficio:** Campus Leonardo, Edificio 6, DCMIC “G. Natta” (Sezione Ingegneria Chimica)

Ing. Timoteo Dinelli

**e-mail:** timoteo.dinelli@polimi.it

**Ufficio:** Campus Leonardo, Edificio 6, DCMIC “G. Natta” (Sezione Ingegneria Chimica)

# Resources

## Books:

- ▶ G., Buzzi Ferraris; Manenti, Flavio. Fundamentals and Linear Algebra for the Chemical Engineer: Solving Numerical Problems.
- ▶ G., Buzzi Ferraris; Manenti, Flavio. Interpolation and Regression Models for the Chemical Engineer: Solving Numerical Problems.
- ▶ G., Buzzi Ferraris; Manenti, Flavio. Nonlinear Systems and Optimization for the Chemical Engineer: Solving Numerical Problems.
- ▶ G., Buzzi Ferraris; Manenti, Flavio. Differential and Differential-Algebraic Systems for the Chemical Engineer: Solving Numerical Problems.
- ▶ J. Nathan Kutz. Data-Driven Modeling and Scientific Computation.

- ▶ Steven L. Brunton; J. Nathan Kutz. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control.
- ▶ A. Quarteroni; R. Sacco; F. Saleri; P. Gervasio. Matematica Numerica.
- ▶ A. Quarteroni; F. Saleri; P. Gervasio. Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave.
- ▶ D. Manca. Calcolo numerico applicato.

### Online Material:

- ▶ Numerical Methods applied to chemical engineering (MIT).
- ▶ GitHub repository of the practical sessions.
- ▶ Matlab online tutorial and documentation.

# Table of Contents

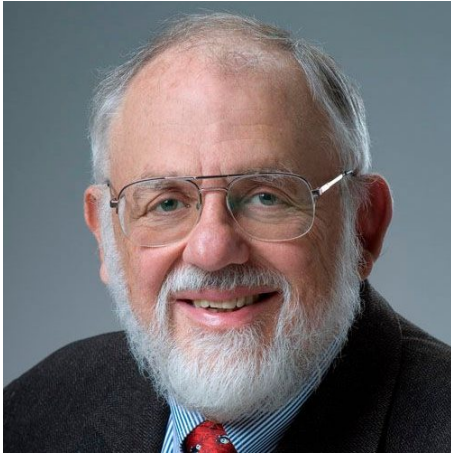
- 1 Introduction
  - What is programming?
  - The Natural Language of Computers
- 2 First Steps with MATLAB
  - Variables
  - Arrays and Matrices
- 3 Loops
  - For
  - While
- 4 Conditional statements
  - If, else if, else
- 5 Functions
- 6 Plot
- 7 Exercises
- 8 Useful material

# What is programming?

- ▶ The most simple answer is: “Programming is the act or activity of writing computer programs.”. In a more general way programming or coding is the act of writing a computer program.
- ▶ A computer program is a sequence of instructions that a computer is able to execute.
- ▶ Computer in the definition above is any device that is capable of processing code. This could be smartphones, ATMs, the Raspberry Pi, Servers to name a few.

Remember, every time we use smart devices, some code is running in the background. Moving a mouse pointer from one part of your computer screen to the other may seem like a simple task, but in reality, so many lines of code just ran. An act as simple as typing letters into Google Docs leads to lines of code being executed in the background. It's all code everywhere.

# The Natural Language of Computers

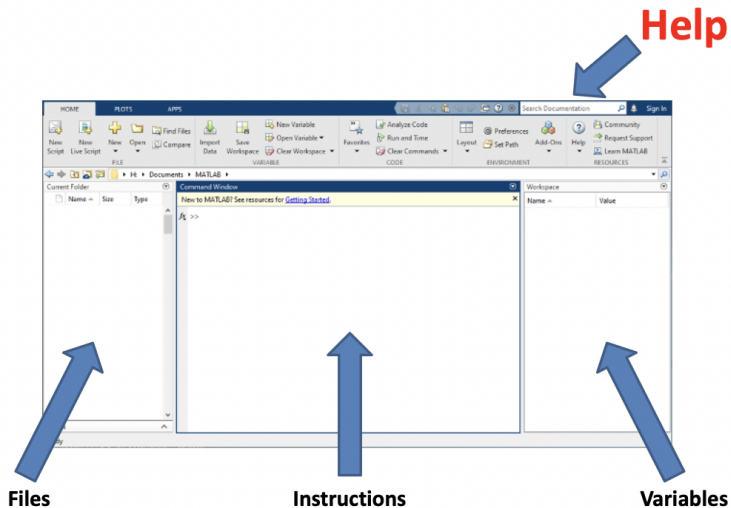


**Figure 1:** Cleve Barry Moler

The only language understood by a computer is the machine language. A very long list of 0 and 1. However it is a little bit inconvenient to write a series of zeros and ones. So (very smart) people, like the one in the picture, invented what are called programming languages (C/C++, Fortran, python, julia, MATLAB, ...).

**N.B.** Computers aren't very smart, the instructions need to be very precise! Telling a computer what you want it to do is sometimes hard because you have to explain things very carefully and precisely.

# MATLAB





# What can a Computer handle?

- ▶ MATLAB deals with numbers, arrays of numbers, characters, strings.
- ▶ Numbers can be of different types: signed and unsigned integers, and single-precision and double-precision floating-point numbers.
- ▶ When we set a variable value it gets stored as a series of 0 and 1 in a memory slot whose size depends on the type of variable (i.e., single-precision floating point numbers take 32 bits, or 4 bytes of memory, double precision 64 bit, or 8 bytes).
- ▶ By default, MATLAB store all values as double-precision floating point.

# Area of a cylinder

Let's compute the area of a cylinder given the diameter  $D = 20$  cm and height  $h = 50$  cm.

## Command window

```
>> D = 0.2 % m  
>> h = 0.5 % m  
>> perimeter = pi * D  
>> Area = perimeter * h
```

## Workspace

- Generate variable D and assign value of 0.2
- Generate variable h and assign value of 0.5
- Generate variable perimeter and assign the result of the operation.
- Generate variable Area and assign the result of the operation.

## Source code: scripts and functions in MATLAB

Scripts are m-files (text format) containing MATLAB statements. MATLAB “functions” are another type of m-file. The biggest difference between scripts and functions is that functions have input and output parameters. Script files can only operate on the variables that are “hard-coded” into their m-file.

```
>> D = 0.2 % m  
>> h = 0.5 % m  
>> perimeter = pi * D  
>> Area = perimeter * h
```

---

```
ans = 0.3142
```

```
function Area = ComputeArea(D, h)  
    perimeter = pi * D;  
    Area = perimeter * h;  
end
```

---

```
>> ComputeArea(0.2, 0.5)  
>> ans = 0.3142
```

# Variables names

- ▶ MATLAB is case sensitive! **Pippo**  $\neq$  **pippo**
- ▶ Variables name should be self explanatory, so prefer **distance, radius, ...** than **a, b, ...**
- ▶ When possible use the camel case notation to make stuff easier to read.  
**PipeLength, GasTemperature, ....**

# Arrays and Matrices

- In MATLAB **arrays** are defined as:

```
>> v = [5 13 97 31 98]
```

```
v =
```

```
5      13      97      31      98
```

- **Matrices** can be defined as a set of stacked arrays separated with ;

```
>> M = [5 13 97; 31 98 36; 11 9 20]
```

```
M =
```

```
5      13      97
31     98     36
11      9     20
```

- Elements can be accessed using their index (indices in MATLAB starts from 1)

```
>> v(1)  
ans = 5
```

```
>> M(2,1) % (row number, column number)  
ans = 31
```

# Creating arrays and matrices

- Create a matrix of zeros or ones:

```
>> A = zeros(3,2)
```

```
A =
```

0	0
0	0
0	0

```
>> A = ones(2,3)
```

```
A =
```

1	1	1
1	1	1

- Create a vector of n equally spaced elements:

```
>> v1 = [1:2:11] % Parenthesis can be omitted
```

```
v1 =      1      3      5      7      9     11
```

```
>> v2 = [1:3:11] % Parenthesis can be omitted
```

```
v2 =      1      4      7     10 % be careful!
```

```
>> v3 = linspace(1,10,6) % Parenthesis can NOT be omitted!
```

```
v3 =      1.0      2.8      4.6      6.4      8.2     10.0
```



# Operations with arrays and matrices

- Size of a matrix:

```
>> M = [1 2 3; 4 5 6]
```

```
>> size(M)
```

```
ans = 2 3
```

- Copying a matrix:

```
>> A = M
```

```
A =
```

1	2	3
4	5	6

- Copying a line or a column of a matrix:

```
>> v = M(1,:)
```

```
v =
```

1	2	3
---	---	---

- Size of an array:

```
>> size(v) ans = 1 3
```

```
>> length(v)
```

```
ans = 3
```

► Matrix transposition:

```
>> C = B'
```

```
C =
```

1	4
2	5
3	6

► Element wise multiplication:

```
>> M .* M
```

```
ans =
```

1	4	9
16	25	36

► Matrix multiplication:

```
>> M * B
```

Error using \* Inner matrices dimensions must agree.

```
>> M * C
```

```
ans =
```

14	32
32	77

```
size(M) = 2 3; size(C) = 3 2
```

## Loops and conditional statements

## for loop

- The **for** loop repeats a series of instructions a **fixed number of times**.

```
>> for i = 1:10
```

```
>>   paperino(i) = i
```

```
>> end
```

---

```
paperino = 1
```

```
paperino = 1 2
```

```
paperino = 1 2 3
```

```
...
```

```
paperino = 1 2 3 4 5 6 7 8 9
```

```
paperino = 1 2 3 4 5 6 7 8 9 10
```

- The index of the **for** loop can be changed by an arbitrary increment:

```
>> for i = 10:-1:1  
>>   pluto(i) = i  
>> end
```

---

```
pluto = 0 0 0 0 0 0 0 0 0 10
```

```
pluto = 0 0 0 0 0 0 0 0 9 10
```

```
pluto = 0 0 0 0 0 0 0 8 9 10
```

```
...
```

```
pluto = 0 2 3 4 5 6 7 8 9 10
```

```
pluto = 1 2 3 4 5 6 7 8 9 10
```

# Examples

- Sum the first hundred natural numbers.

$$\sum_{i=1}^{100} i = ?$$

- Sum hundred times one.

$$\sum_{i=1}^{100} 1 = ?$$

```
>> sum = 0;  
>> for i = 1:100  
>>     sum = sum + i;  
>> end  
>> disp(sum);
```

```
>> sum = 0;  
>> for i = 1:100  
>>     sum += 1;  
>> end  
>> disp(sum);
```

## while loop

- ▶ The **while** loop repeats a series of instructions **until a condition is TRUE**.
- ▶ N.B. pay attention with the while loop it can run till infinite, **handle with care**.

```
>> sum = 0;  
>> while (sum < 325)  
>>     token = token + 1  
>>     sum = sum + token;  
>> end  
disp(["Iteration number: ", num2str(token)]);  
disp(["Sum is equal to: ", num2str(sum)]);
```

---

```
Iteration number: 25  
Sum is equal to: 325
```

## IF Statement

- The **IF** statement executes a series of instructions only **IF** a condition is TRUE:

```
if (condition)
... instructions
elseif (condition)
... instructions
elseif (condition)
... instructions
else
... instructions
end
```

Example: Write a simple script to compute the absolute value of a number.

```
>> x = 35
>> if (x ≥ 0)
>>     abs = x;
>> else
>>     abs = -x
>> end
```



# Functions and Plot

# Functions

In MATLAB, a **function** is a block of code that takes inputs, performs a set of operations, and returns outputs. Functions allow you to organize code in a clearer and more reusable way, especially when you need to perform the same operation in different parts of your program.

## Why using functions?

- ▶ **Modularity:** Separating code into functions makes the program more readable and easier to maintain.
- ▶ **Reusability:** A function can be reused in different parts of the program without rewriting the code.

# Syntax

```
function [y1, ..., yN] = function_name (x1, ..., xM)  
% code  
% code  
% code  
end
```

Input variables.

Name of the function (user defined).

Output variables (parenthesis can be omitted).

The diagram illustrates the syntax of a MATLAB function definition. The code is enclosed in a blue rounded rectangle. The word 'function' is in blue. The output variables '[y<sub>1</sub>, ..., y<sub>N</sub>]' are highlighted in a light green box, with a green arrow pointing to the label 'Output variables (parenthesis can be omitted)'. The function name 'function\_name' is highlighted in a light pink box, with a pink arrow pointing to the label 'Name of the function (user defined)'. The input variables '(x<sub>1</sub>, ..., x<sub>M</sub>)' are highlighted in a light purple box, with a purple arrow pointing to the label 'Input variables.'. The code continues with three lines of green text: '% code', '% code', and '% code', followed by the blue keyword 'end'.

# Plot

Let's plot on a Cartesian plane the following function:

$$y(x) = 3x + 2$$

```
x = 0:1:20  
for i = 1:length(x)  
    y(i) = 3 * x(i) + 2;  
end  
plot(x, y)
```

Now let's add to the previous plot the following function:

$$y(x) = -3x + 4$$

```
x = 0:1:20  
for i = 1:length(x)  
    y1(i) = 3 * x(i) + 2;  
    y2(i) = -3 * x(i) + 4;  
end  
hold on  
plot(x, y)  
plot(x, y2)
```

# Exercises

## Example

Let's define a square matrix populated with random numbers, using the function **`magic()`** and loop over its rows and columns to match specific conditions.

```
>> clear, clc
>> M = magic(300);
>> for (i = 1:300)
>>     for (j = 1:300)
>>         if(M(i,j) == 569)
>>             disp('Found!')
>>         end
>>     end
>> end
```

# The babylonian method

Write a function that implements the Babylonian method to compute the square root of a number with a precision of four decimal figures.

1. **Make an Initial guess.** Guess any positive number  $x_0$ .
2. **Improve the first guess.** Apply the formula  $x_1 = \frac{x_0 + \frac{S}{x_0}}{2}$ . The number  $x_1$  is a better approximation to  $\sqrt{S}$ .
3. **Iterate until convergence.** Apply the formula  $x_{n+1} = \frac{x_n + \frac{S}{x_n}}{2}$  until the convergence is reached.

**Convergence** is reached when the digits of  $x_{n+1}$  and  $x_n$  agree to as many decimal places as you desire.



```
>> function SquareRoot = ComputeSquareRoot(S)
>>     iter = 0;
>>     x0 = S;
>>     y = 0.5*(x0+S/x0);
>>     while abs(x0-y)>1e-5 && iter<50
>>         x0 = y;
>>         y = 0.5*(x0+S/x0);
>>         iter = iter+1;
>>     end
>>     disp(['Number of iteration to reach convergence: ', num2str(iter)])
>>     SquareRoot = y;
>> end
```

Useful material

# Useful Functions

- ▶ **clc**: clear the screen of the command window.
- ▶ **clear / clear all**: erase all the variables contained inside the workspace.
- ▶ **close / close all**: close all the pop-up windows opened after plotting something.
- ▶ **disp(variable name)**: print inside the command window the value of a certain variable.
- ▶ **fprintf(...)**: same function of “disp” with a slight different behaviour.
- ▶ **magic(n)**: is an  $n$ -by- $n$  matrix constructed from the integers 1 through  $n^2$  with equal row, column, and diagonal sums. Produces valid magic squares for all  $n > 0$  except  $n = 2$ .
- ▶ **rand(n)**: returns an  $n$ -by- $n$  matrix of uniformly distributed random numbers.

Line Style	
'_'	Solid Line
'--'	Dashed Line
'.'	Dotted Line
'-.'	Dashed Dotted Line
Marker	
'o'	Circle
'+'	Plus
'*'	Asterisc
'.'	Point
'x'	Cross
'-'	Horizontal line

' '	Vertical line
's'	Square
'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'p'	Pentagram
'h'	Hexagram

Thank you for the attention!