ALGERIAN DEMOCRATIC AND POPULAR REPUBLIC
Ministry of Higher Education and Scientific Research

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology

Course Handout

# *Computer Science 3*

Module UEM 2.1
English version: Informatique 3
Lectures / lab. Sessions

Dr. TADJEDDINE Ali Abderrazak;

Dr. BENDELHOUM Mohammed Sofiane;

Dr. BENDJILLALI Ridha Ilyas;

2023-2024

# Preface

In line with the Ministry of Higher Education and Scientific Research's initiatives to enhance the prevalence of the English language within academic spheres, this course handout constitutes the translated version of our preceding material on Computer Science 3.

This course manual / lab material, entitled "**Computer Science 3**," is a methodological subject studied by second-year undergraduate students in the 3rd semester, as part of the common exchange in the fields of science and technology across all specialties. The curriculum of this course/lab is designed to provide students with a fundamental understanding of programming and numerical computation using Matlab.

Although the term "Matlab" stands for Matrix Laboratory in English, Matlab is a powerful computer programming language that can be utilized as a calculator, serving as a valuable tool for experimenting with ideas that could be implemented in your program. However, once you have moved beyond the experimentation stage, you typically rely on MATLAB to create a program that assists you in performing tasks: Regularly, Easily, and Rapidly. While these three characteristics do not encompass the entirety of MATLAB's capabilities, they do provide you with concepts to pursue and leverage to your advantage.

The fields of Science, Technology, Engineering, and Mathematics (STEM) currently emphasize the importance of simulating mathematical models prior to experimentation. Innovation in all these domains necessitates such practices, as do many practical professions. MATLAB offers a rich and extensive toolbox for STEM that encompasses Statistics, Simulation, Image Processing, Symbolic Processing, and Numerical Analysis.

## Approach

Suggestions for enhancing the course will be greatly appreciated. While every effort has been made to eliminate errors, claiming perfection is a challenging feat. I will be very grateful to teachers, students, and users of this course if they identify any errors that may have inadvertently surfaced.

In order to simplify the learning process for students, each course/lab sheet consists of four parts:

1. **TP Objective:**
   This section covers the purpose of the lab session and the main concepts being taught.
2. **Part 01: Theoretical Section (Matlab Interface)**
   This part includes definitions of the commands and instructions used during the session.
3. **Part 02: Simulation Section (MATLAB - SIMULINK)**
   It contains solved exercises with illustrative figures for better understanding.
4. **Part 03: Experimental Section (MATLAB - SIMULINK)**
   This section contains exercises to adapt to programming problems and apply the concepts covered in Parts 1 and 2.

Organization of Lab Sheets:
According to the official framework, this course is divided into eight lab sheets:

| TP No. | Title | Duration |
|---|---|---|
| **TP 1** | **Presentation of MATLAB** | **1 week** |
| **TP 2** | **Reading, Displaying, and Saving Data** | **2 weeks** |
| **TP 3** | **Script Files and Data Types and Variables** | **2 weeks** |
| **TP 4** | **Vectors and Matrices** | **2 weeks** |
| **TP 5** | **Control Statements (Loops, if, and switch)** | **2 weeks** |
| **TP 6** | **Function Files** | **2 weeks** |
| **TP 7** | **Graphics (Window Management, Plotting)** | **2 weeks** |
| **TP 8** | **Toolbox Usage** | **2 weeks** |

# Contents

# Lab Sheet 01
# Presentation of MATLAB

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

# Lab Sheet 01: Presentation of MATLAB

## TP Objective:

The objective of Lab Sheet 01 is to acquaint you with the interface and fundamentals of the MATLAB environment, enabling you to use fundamental functions for data reading, display, and storage.

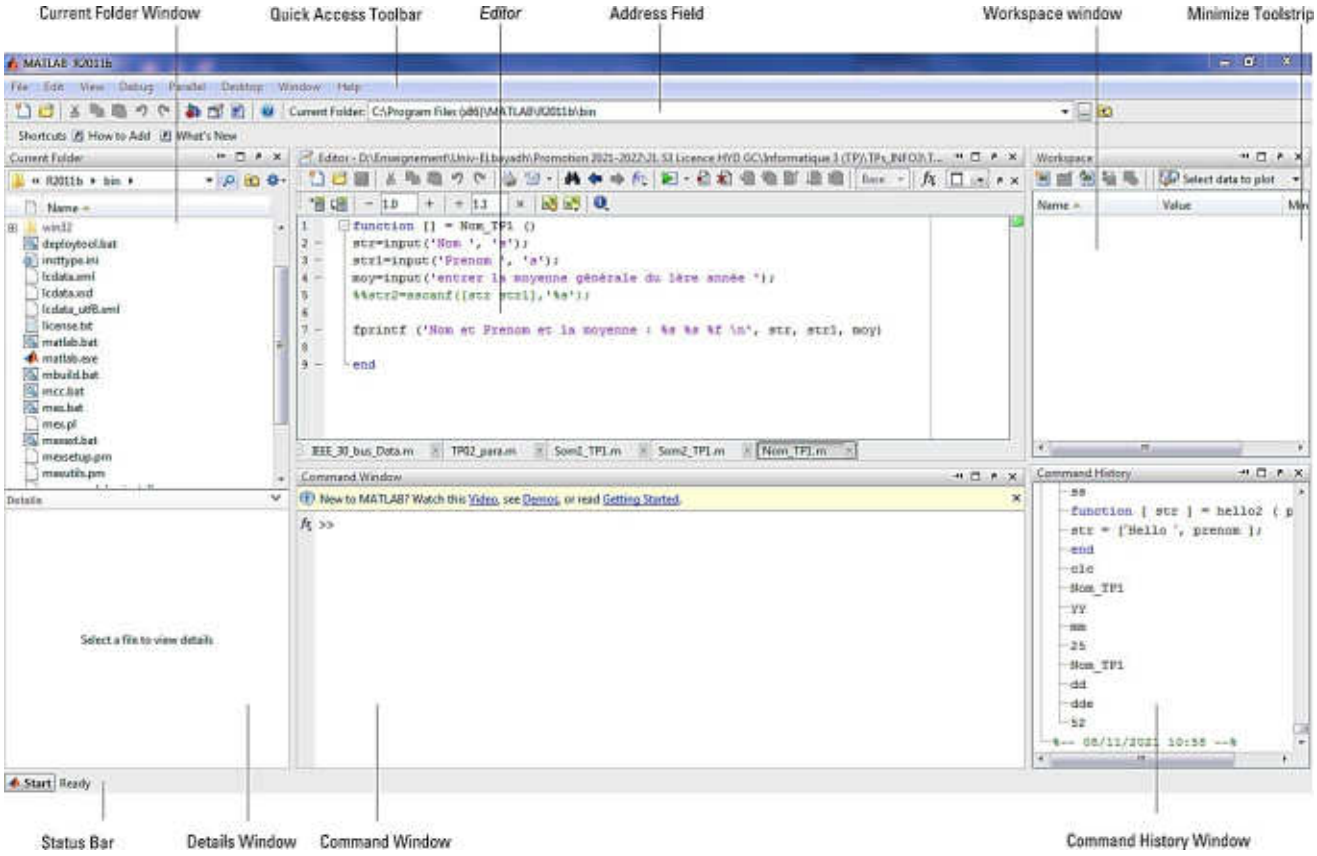



*Figure 1. Matlab Interface*

## Part 01: Theoretical Section (Matlab Interface)

In this section, we will become acquainted with the Matlab interface. Depending on the version being used, the interface might vary slightly, but the core elements will remain consistent.

### Command Window

A scalar variable (x = 5: assigning the value 5 to the variable x) is perceived by MATLAB as a matrix with dimensions 1x1 (1 row by 1 column). This is illustrated in the following example

where we assign the value 5 to the variable x and subsequently inquire about its dimensions using the function size(x) in the **Command Window** space:

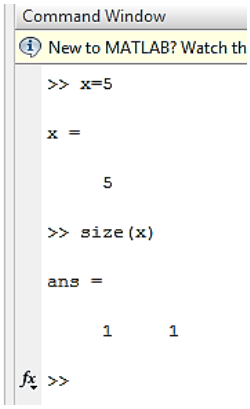



*Figure 2. Command Window*

In MATLAB, the semicolon (;) is used to suppress the display of output or results of an expression. When you include a semicolon at the end of a command or statement, MATLAB will execute the command but will not display the result in the Command Window. This can be particularly useful when you are performing calculations or operations and don't need to see every intermediate result.

*Table 1. Usage of Semicolon*

| Usage of Semicolon | *ans* (answer) |
|---|---|
| To prevent the display of results, simply append a semicolon (;) to the end of the command.<br>`>> x=7;`<br>`>> y=size(x);`<br>`>>` | Matlab defines a variable 'ans', which is a matrix with dimensions 1x1 (one row by one column). |

**For example: with Matlab code**

a = 5; % Variable 'a' is assigned the value 5

b = 3 + 2; % Expression is calculated but not displayed

c = a * b; % Expression is calculated but not displayed

In this case, only the assignment of variable 'a' is shown in the Command Window, while the calculations involving 'b' and 'c' are performed without displaying their values.

A command '**clc**': This command clears the Command Window to provide a clean display.

A command '**clear**': This command removes all variables from the Workspace, enabling a fresh start with a clean workspace.

### Workspace (go to Window -> Workspace)

In this window, you will find a list of variables recognized by MATLAB. You can double-click on a variable to display its contents. Right-clicking on variables provides numerous options such as: Copy, Paste, Delete, and more [2.1].

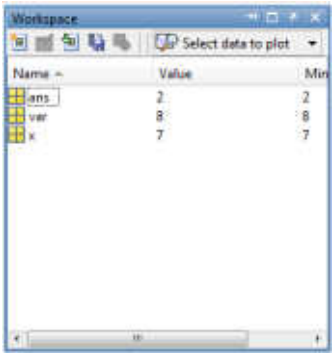



*Figure 3. Workspace*

### Command History (go to Window -> Command History)

The Command History space retains a record of all operations that have been performed in the Command Window. You can also navigate through the list of commands by being in the Command Window and using the arrow keys ↑ and ↓.


*Figure 4. Command History*

### Current Folder (go to Window -> Current Folder)

It's the folder that contains script files, programming, and the work done in Matlab.

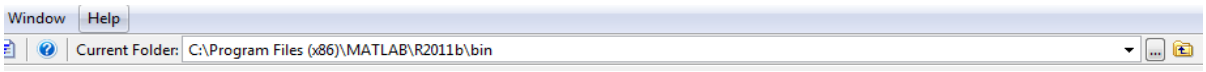



*Figure 5. Current Folder*

### Help

The help space is essential when programming in a high-level language like Matlab, where the number of functions is extensive and syntax can be complex. To access help, you can either select a function or press F1 on the keyboard, or type in the Command Window: "help cos", "help input", etc. It's crucial for you to become acquainted with Matlab's help tools to succeed in this course/lab.

*Table 2. Help command*

| ***Command*** | ***Description*** |
|---|---|
| ***helpwin*** | ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations |
| ***help*** | donne la liste de toutes les commandes par thèmes |
| ***help nom*** | décrit la fonction nom.m |
| ***lookfor nom*** | Recherche une instruction à partir du mot clé nom |

***Use: help → Product Help to display the help Window***

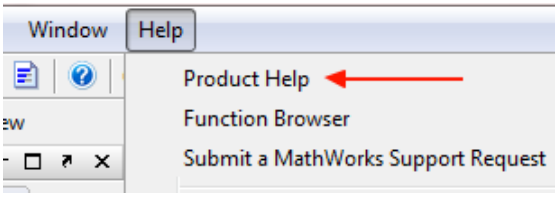



*Figure 6. Help Window*

**Script**

A script is a file with the extension '.m' that contains the program in a more simplified manner. It consists of a list of commands and functions.

**Function**

A function allows you to input arguments and obtain various variables as output.

**Editor**

Most of your work in Matlab will involve creating or modifying files with the ".m" extension, which defines Matlab files. When performing a task in Matlab, it's often possible to accomplish it using just the Command Window. However, when the task becomes more complex (involving several lines of code) or you want to easily share it with someone else, you use the Editor window. You create an .m file that can be either a script or a function [2.2].

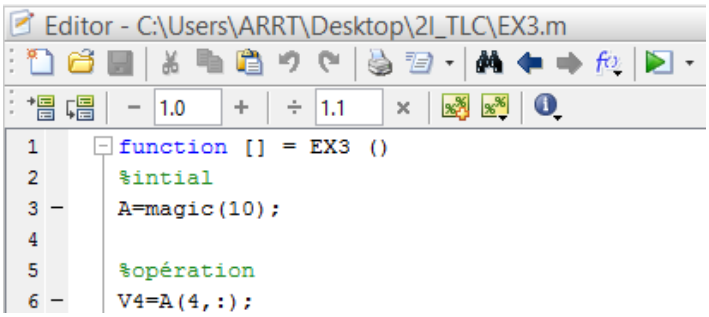



*Figure 7. Editor window, function and Script*

## Part 02: Simulation Section (MATLAB - SIMULINK)

To start, we set the USERPATH, the folder containing your (.m) file.

1. Create a folder on your Desktop named: TP_INFO3.
2. Inside the previous folder, create another folder named: TP01_INFO3.
3. In the Current Folder space as shown in figure 5, select the TP01_INFO3 folder for the first lab session, and similarly for subsequent lab sessions.

### Exercise 1: "Hello World"

In this exercise, you will create a simple program to display the famous "Hello World" message using MATLAB. This exercise serves as an introduction to writing and executing basic code in MATLAB in script mode.

Here's the basic code snippet for displaying "Hello World" in MATLAB:

```
>>disp('Hello World');
```

You can create a new script file (.m) in the designated folder, for example, TP01_INFO3, and add the above code to it.

Then, run the script in MATLAB to see the "Hello World" message displayed in the **Command Window**.

### Exercise 2: "Sum of Two Numbers 2 and 3"

To calculate the sum of the numbers 2 and 3 using the function **Sum_TP1**, you can follow these steps:

1. In the Command Window, use this script:

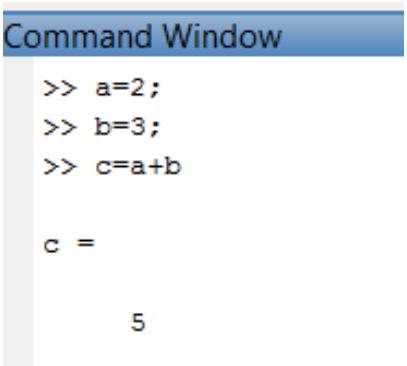



*Figure 8. Script Sum of Two Numbers 2 and 3*

2. In the Editor Window, you can use the function with the numbers 2 and 3 as arguments:

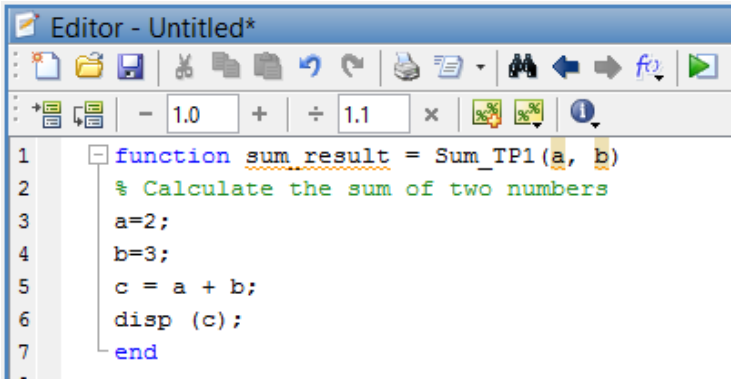



*Figure 9. Function Sum of Two Numbers 2 and 3*

## Exercise 3: "Sum of Any Two Numbers"

Create a new script file in the designated folder, TP01_INFO3, and add the above code. When you run the script in MATLAB, it will prompt you to input two numbers and then display the sum of those numbers in the Command Window.

Let's attempt to create a script and then convert it into a function that takes two numbers as input and returns their sum [2.3].

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program that calculates the sum of two numbers; numbers x and y.
3. Save the file with the name: **Sum_xy_TP1.m** (use underscore "_ ").

Here's how you can structure your script and then convert it into a function:

**With Script Version (Sum_xy_TP1.m):**

```
>>% Calculate the sum of two numbers x and y
>>x = input('Enter the first number: ');
>>y = input('Enter the second number: ');
>>sum_result = num1 + num2;
>>disp(['The sum of ', num2str(x), ' and ', num2str(y), ' is ',
num2str(sum_result)]);
```

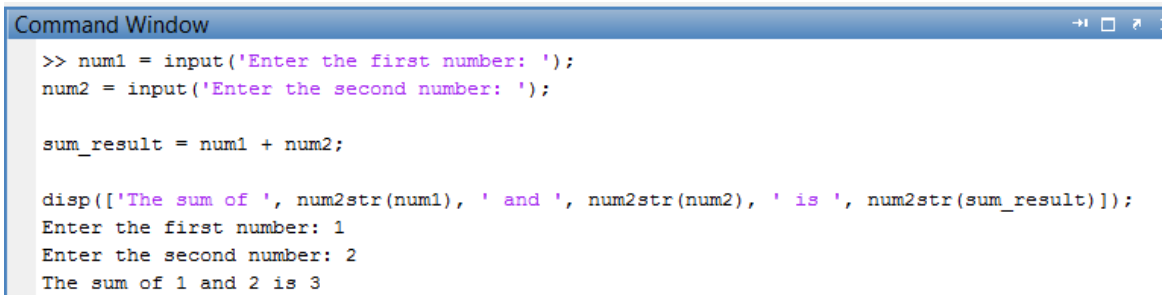



*Figure 10. Exercise 2, Sum of Two Numbers: **Script Version** (Sum_xy_TP1.m)*

**With Function Version (Sum_xy_TP1.m):**

```
1. function sum_result = Sum_xy_TP1(x, y)
2. % Calculate the sum of two numbers
3. sum_result = x + y;
4. disp (sum_result);
5. end
```

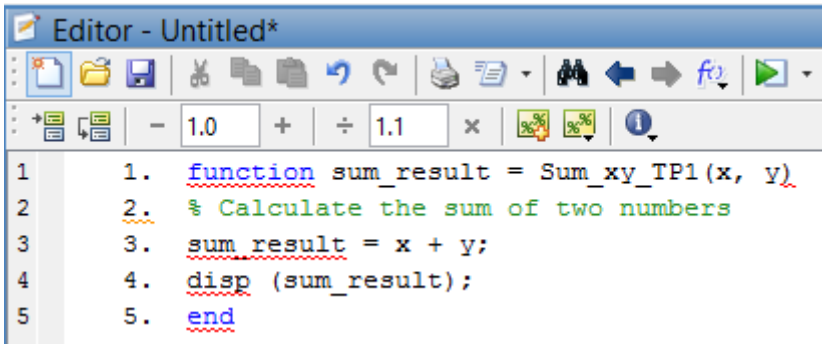



*Figure 11. Exercise 2, Sum of Two Numbers: **Function Version** (Sum_xy_TP1.m)*

In the function version, you define a function named Sum_xy_TP1 that takes two input arguments x and y, calculates their sum, and returns the result. Remember to save the function file in the same folder, and you can then call this function from the Command Window or other scripts to calculate the sum of two numbers when you need.

### Exercise 4: "Testing the function Sum_xy_TP1.m"

In this exercise, you'll create a MATLAB function that calculates the sum of the two numbers using the function **Som_xy_TP1** that you've created, this exercise will reinforce your understanding of creating functions in MATLAB, you can follow these steps.

1. Make sure you have the Sum_xy_TP1 function defined in a file named "Sum_xy_TP1.m" in your MATLAB working directory (**Current Folder).**
2. In the Command Window, you can call the function with any two numbers as arguments:

```
>>result = Sum_xy_TP1(2, 3);
>>disp(['The sum of 2 and 3 is: ', num2str(result)]);
```

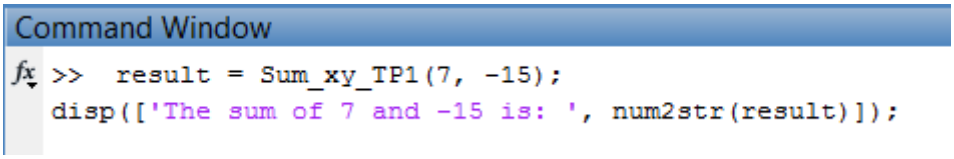



*Figure 12. Call the function Sum_xy_TP1.m*

This demonstrates how you can use the custom function Sum_xy_TP1 to calculate the sum of any two numbers.

### Exercise 5: "How to Find Desired Information"

In this exercise, you'll practice using MATLAB's **help** and documentation resources to find the information you need. These skills are crucial for effectively using MATLAB's capabilities and functions.

Follow these steps:

1. Go to the **Command Window** in MATLAB.
2. Write **help** command or the **doc** command to search for information about a specific function or topic.

For example, let's say you want to find information about the **sqrt** function, which calculates the square root of a number:

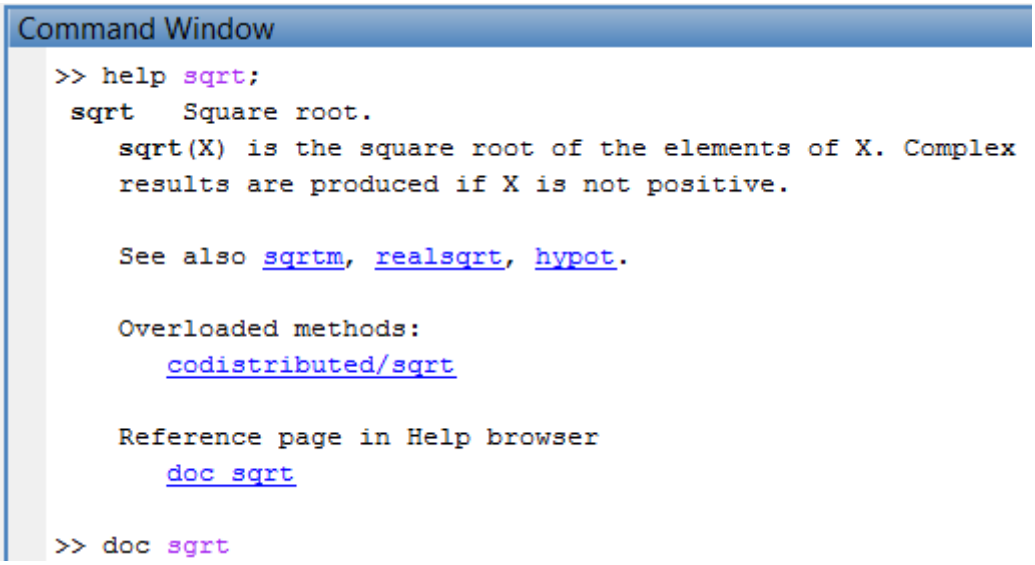



*Figure 13. Find information about the sqrt function*

This command (doc sqrt) will display the help documentation for the sqrt function, including its syntax, description, and usage examples. Similarly, you can use the help or doc command to explore other functions and topics within MATLAB to gain a deeper understanding of their functionalities and how to use them.

Remember, being able to effectively search for and use documentation is a valuable skill for programming in MATLAB.

## Part 03: Experimental Section (MATLAB - SIMULINK)

### Exercise 6: "Calculate operations"

In this exercise, you'll create a MATLAB program that takes three numbers as input and displays the product of the first two numbers and the square root of the product of the last two numbers. This exercise will reinforce your understanding of basic calculations and functions in MATLAB.

Follow these steps:

1. Open a new script file (Ctrl + N).
2. Write a program that takes three numbers as input, calculates the desired values, and displays the results.
3. Save the file with an appropriate name, such as "Sum_Product_Root.m".

Here's how you can structure your script for this exercise:

```
>>% Input three numbers
>>num1 = input('Enter the first number: ');
>>num2 = input('Enter the second number: ');
>>num3 = input('Enter the third number: ');
>>% Calculate product of the first two numbers
>>product_result = num1 * num2;
>>% Calculate square root of product of the last two numbers
>>root_result = sqrt(num2 * num3);
>>% Display the results
>>disp(['The product of ', num2str(num1), ' and ', num2str(num2), ' is ', num2str(product_result)]);
>>disp(['The square root of the product of ', num2str(num2), ' and ', num2str(num3), ' is ', num2str(root_result)]);
```

Create a new script file, add the above code, and run the script in MATLAB. It will prompt you to input three numbers and then display the calculated values as described.

Using this function in Editor Window :

```
function [] = Som2_TP1()
x=input('entrer le 1er nombre ')
y=input('entrer le 2e nombre ');
z=input('entrer le 3e nombre ');
a=x*y;
disp('^_^');
fprintf('le produit des 1er nombres est: %f \n', a)
b=sqrt(y*z);
fprintf ('la racine des deux dernier nombres est: %.2f \n', b);
disp('^_^');
end
```

*Figure 14. Calculate operations*

In this script we used 4 functions "**input(), sqrt(), dips() and fprintf()**",

- Use the command "help" to discover these functions.
- **What is the purpose of these functions?**

## Exercise 7: "Enter First and Last Name"

In this exercise, you'll create a MATLAB function that takes your first name and last name as input arguments and returns them on the same line. This exercise will help you practice creating and using functions with input and fprintf command in MATLAB.

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program in Editor that defines a function to concatenate your first name and last name and display them.
3. Save the file with an appropriate name, such as "Name_TP1.m".

Here's how you can structure your script to create the function:

**Script Version (FullNameConcatenation.m):**

```
1. % Concatenate first name and last name
2. F=input('enter your first name');
3. L= input('enter your last name');
4. fprintf(' full_name is : %s, %s ', F, L);
```

**Function Version (FullNameConcatenation.m):**

```
5. function Name_TP1 = Name_TP1(first_name, last_name)
6. % Concatenate first name and last name
7. full_name = [first_name, ' ', last_name];
8. end
```

- In this function, you define a function named *Name_TP1* that takes two input arguments first_name and last_name, concatenates them with a space in between, and returns the full name.
- Now, save the function file in the same folder and call this function with your first and last name as arguments to see your full name displayed on the same line.

For example, in the Command Window:

```
>> result = Name_TP1(Abdelkader, Amir);
>> disp(['Your full name is: ', result]);
result =
      Abdelkader Amir
```

## Exercise 8: "Enter First and Last Name + First-Year Overall Average"

In this exercise, you'll create a MATLAB function that takes your first name, last name, and your first-year overall average as input arguments and returns them on the same line. This exercise will help you practice creating functions with multiple input arguments in MATLAB.

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program that defines a function to concatenate your full name and your first-year overall average and then display them.
3. Save the file with the name: "Name_Moy_TP1.m".

# Lab Sheet 02
# Reading, Displaying, and Saving Data

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

# Lab Sheet 02: Reading, Displaying, and Saving Data

## TP Objective:

The objective of Lab Sheet 02 is to acquaint you with the interface and fundamentals of the MATLAB environment, enabling you to use fundamental functions for data reading, display, and storage.

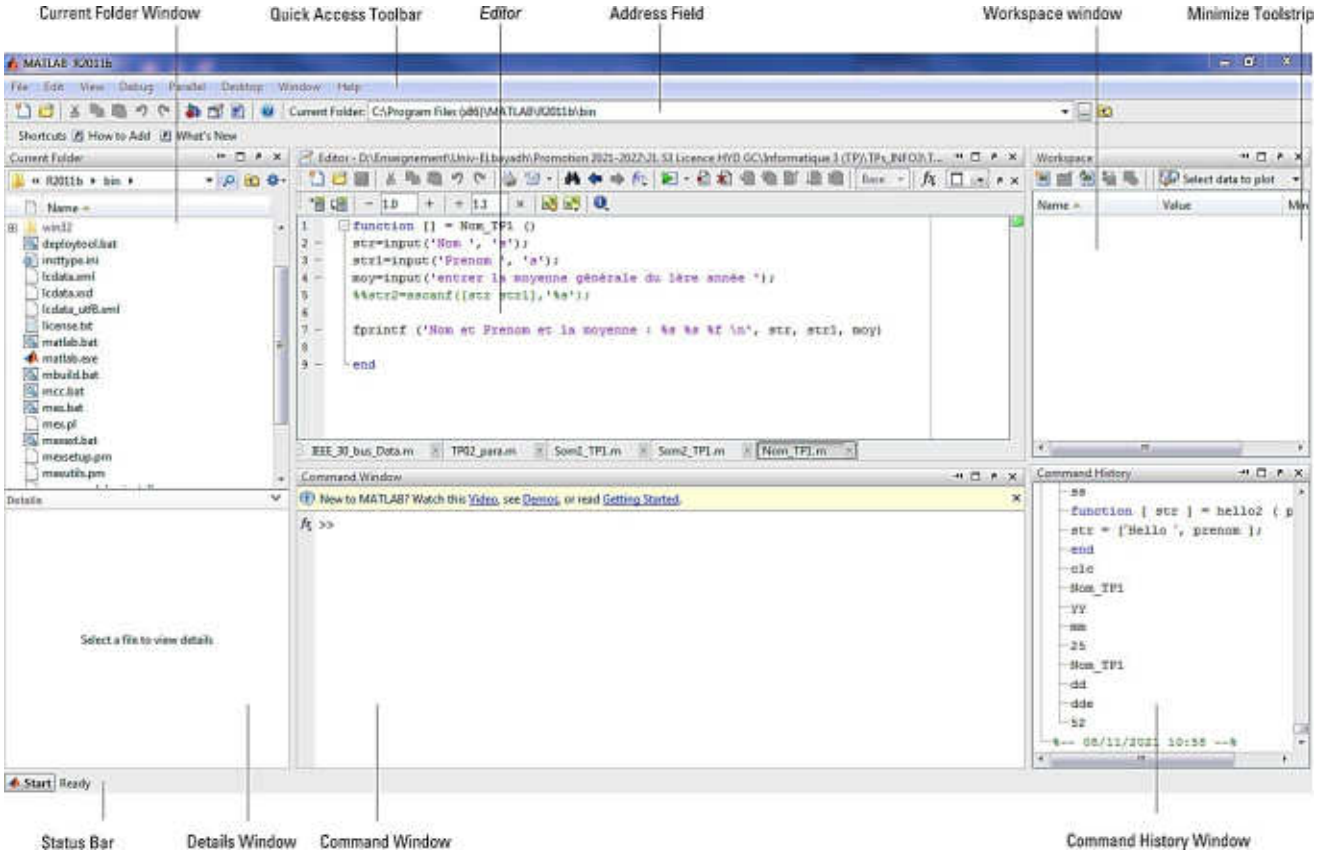



*Figure 1. Matlab Interface*

## Part 01: Theoretical Section (Matlab Interface)

In this section, we will become acquainted with the Matlab interface. Depending on the version being used, the interface might vary slightly, but the core elements will remain consistent.

### Command Window

A scalar variable (x = 5: assigning the value 5 to the variable x) is perceived by MATLAB as a matrix with dimensions 1x1 (1 row by 1 column). This is illustrated in the following example

where we assign the value 5 to the variable x and subsequently inquire about its dimensions using the function size(x) in the **Command Window** space:

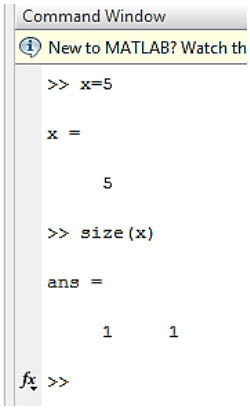



*Figure 2. Command Window*

In MATLAB, the semicolon (;) is used to suppress the display of output or results of an expression. When you include a semicolon at the end of a command or statement, MATLAB will execute the command but will not display the result in the Command Window. This can be particularly useful when you are performing calculations or operations and don't need to see every intermediate result.

*Table 1. Usage of Semicolon*

| Usage of Semicolon | *ans* (answer) |
|---|---|
| To prevent the display of results, simply append a semicolon (;) to the end of the command.<br>`>> x=7;`<br>`>> y=size(x);`<br>`>>` | Matlab defines a variable 'ans', which is a matrix with dimensions 1x1 (one row by one column). |

**For example: with Matlab code**

a = 5; % Variable 'a' is assigned the value 5

b = 3 + 2; % Expression is calculated but not displayed

c = a * b; % Expression is calculated but not displayed

In this case, only the assignment of variable 'a' is shown in the Command Window, while the calculations involving 'b' and 'c' are performed without displaying their values.

A command '**clc**': This command clears the Command Window to provide a clean display.

A command '**clear**': This command removes all variables from the Workspace, enabling a fresh start with a clean workspace.

### Workspace (go to Window -> Workspace)

In this window, you will find a list of variables recognized by MATLAB. You can double-click on a variable to display its contents. Right-clicking on variables provides numerous options such as: Copy, Paste, Delete, and more.

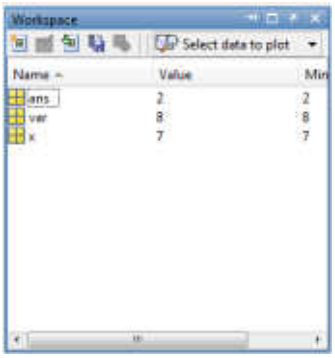



*Figure 3. Workspace*

### Command History (go to Window -> Command History)

The Command History space retains a record of all operations that have been performed in the Command Window. You can also navigate through the list of commands by being in the Command Window and using the arrow keys ↑ and ↓.


*Figure 4. Command History*

### Current Folder (go to Window -> Current Folder)

It's the folder that contains script files, programming, and the work done in Matlab.

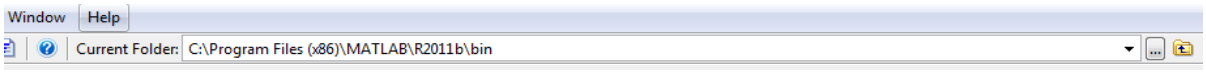



*Figure 5. Current Folder*

### Help

The help space is essential when programming in a high-level language like Matlab, where the number of functions is extensive and syntax can be complex. To access help, you can either select a function or press F1 on the keyboard, or type in the Command Window: "help cos", "help input", etc. It's crucial for you to become acquainted with Matlab's help tools to succeed in this course/lab.

*Table 2. Help command*

| ***Command*** | ***Description*** |
|---|---|
| ***helpwin*** | ouvre une fenêtre contenant la liste des commandes Matlab ainsi que leurs documentations |
| ***help*** | donne la liste de toutes les commandes par thèmes |
| ***help nom*** | décrit la fonction nom.m |
| ***lookfor nom*** | Recherche une instruction à partir du mot clé nom |

***Use: help → Product Help to display the help Window***

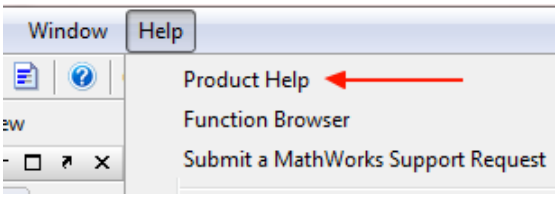



*Figure 6. Help Window*

### Script

A script is a file with the extension '.m' that contains the program in a more simplified manner. It consists of a list of commands and functions.

### Function

A function allows you to input arguments and obtain various variables as output.

### Editor

Most of your work in Matlab will involve creating or modifying files with the ".m" extension, which defines Matlab files. When performing a task in Matlab, it's often possible to accomplish it using just the Command Window. However, when the task becomes more complex (involving several lines of code) or you want to easily share it with someone else, you use the Editor window. You create an .m file that can be either a script or a function.

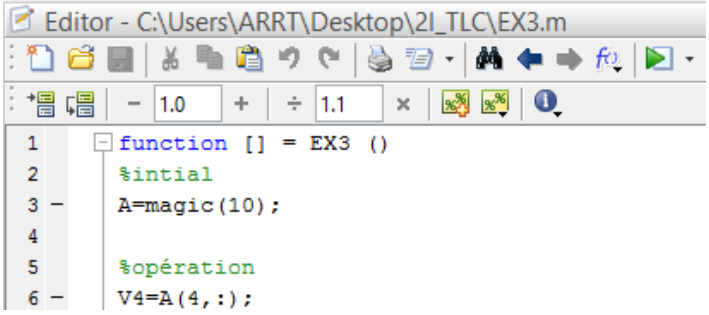



*Figure 7. Editor window, function and Script*

## Part 02: Simulation Section (MATLAB - SIMULINK)

To start, we set the USERPATH, the folder containing your (.m) file.

1. Create a folder on your Desktop named: TP_INFO3.
2. Inside the previous folder, create another folder named: TP01_INFO3.
3. In the Current Folder space as shown in figure 5, select the TP01_INFO3 folder for the first lab session, and similarly for subsequent lab sessions.

### Exercise 1: "Hello World"

In this exercise, you will create a simple program to display the famous "Hello World" message using MATLAB. This exercise serves as an introduction to writing and executing basic code in MATLAB in script mode.

Here's the basic code snippet for displaying "Hello World" in MATLAB:

```
>>disp('Hello World');
```

You can create a new script file (.m) in the designated folder, for example, TP01_INFO3, and add the above code to it.

Then, run  the script in MATLAB to see the "Hello World" message displayed in the **Command Window**.

### Exercise 2: "Sum of Two Numbers 2 and 3"

To calculate the sum of the numbers 2 and 3 using the function **Sum_TP1**, you can follow these steps:

1. In the Command Window, use this script:

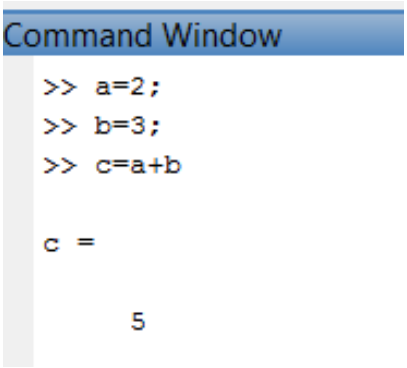



*Figure 8. Script Sum of Two Numbers 2 and 3*

2. In the Editor Window, you can use the function with the numbers 2 and 3 as arguments:

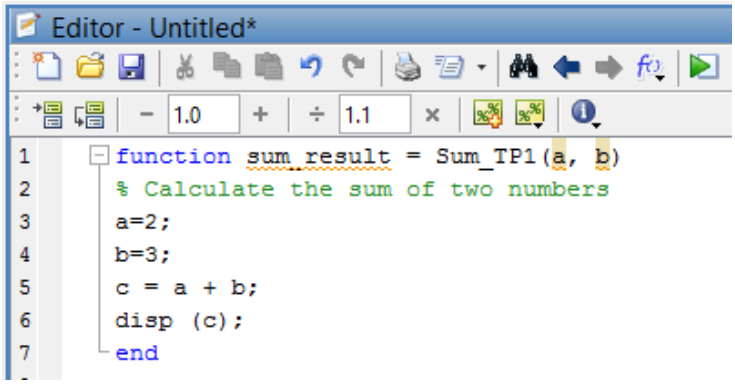



*Figure 9. Function Sum of Two Numbers 2 and 3*

## Exercise 3: "Sum of Any Two Numbers"

Create a new script file in the designated folder, TP01_INFO3, and add the above code. When you run the script in MATLAB, it will prompt you to input two numbers and then display the sum of those numbers in the Command Window.

Let's attempt to create a script and then convert it into a function that takes two numbers as input and returns their sum.

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program that calculates the sum of two numbers; numbers x and y.
3. Save the file with the name: **Sum_xy_TP1.m** (use underscore "_ ").

Here's how you can structure your script and then convert it into a function:

**With Script Version (Sum_xy_TP1.m):**

```
>>% Calculate the sum of two numbers x and y
>>x = input('Enter the first number: ');
>>y = input('Enter the second number: ');
>>sum_result = num1 + num2;
>>disp(['The sum of ', num2str(x), ' and ', num2str(y), ' is ',
num2str(sum_result)]);
```

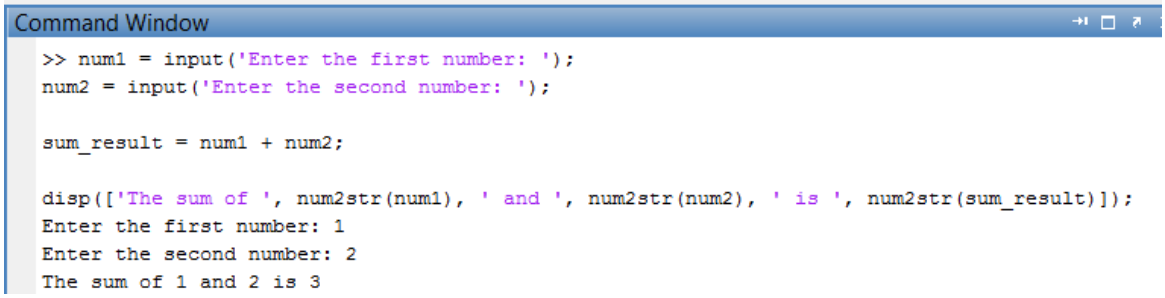



*Figure 10. Exercise 2, Sum of Two Numbers: **Script Version** (Sum_xy_TP1.m)*

**With Function Version (Sum_xy_TP1.m):**

```
1. function sum_result = Sum_xy_TP1(x, y)
2. % Calculate the sum of two numbers
3. sum_result = x + y;
4. disp (sum_result);
5. end
```

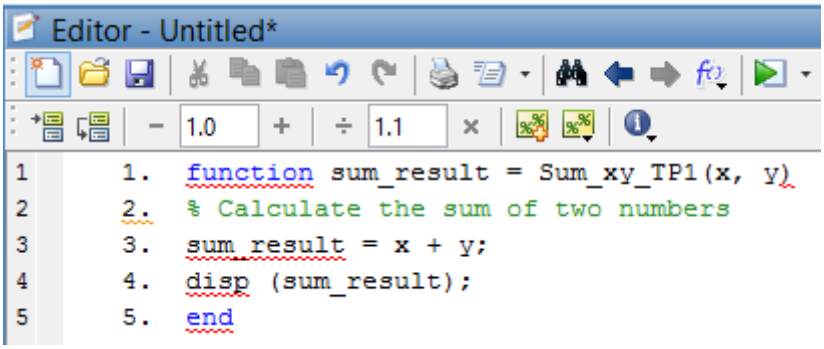



*Figure 11. Exercise 2, Sum of Two Numbers: **Function Version** (Sum_xy_TP1.m)*

In the function version, you define a function named Sum_xy_TP1 that takes two input arguments x and y, calculates their sum, and returns the result. Remember to save the function file in the same folder, and you can then call this function from the Command Window or other scripts to calculate the sum of two numbers when you need.

### Exercise 4: "Testing the function Sum_xy_TP1.m"

In this exercise, you'll create a MATLAB function that calculates the sum of the two numbers using the function **Som_xy_TP1** that you've created, this exercise will reinforce your understanding of creating functions in MATLAB, you can follow these steps.

1. Make sure you have the Sum_xy_TP1 function defined in a file named "Sum_xy_TP1.m" in your MATLAB working directory (**Current Folder).**
2. In the Command Window, you can call the function with any two numbers as arguments:

```
>>result = Sum_xy_TP1(2, 3);
>>disp(['The sum of 2 and 3 is: ', num2str(result)]);
```

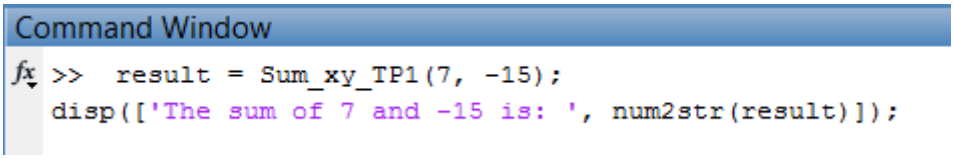



*Figure 12. Call the function Sum_xy_TP1.m*

This demonstrates how you can use the custom function Sum_xy_TP1 to calculate the sum of any two numbers.

## Exercise 5: "How to Find Desired Information"

In this exercise, you'll practice using MATLAB's **help** and documentation resources to find the information you need. These skills are crucial for effectively using MATLAB's capabilities and functions.

Follow these steps:

1. Go to the **Command Window** in MATLAB.
2. Write **help** command or the **doc** command to search for information about a specific function or topic.

For example, let's say you want to find information about the **sqrt** function, which calculates the square root of a number:

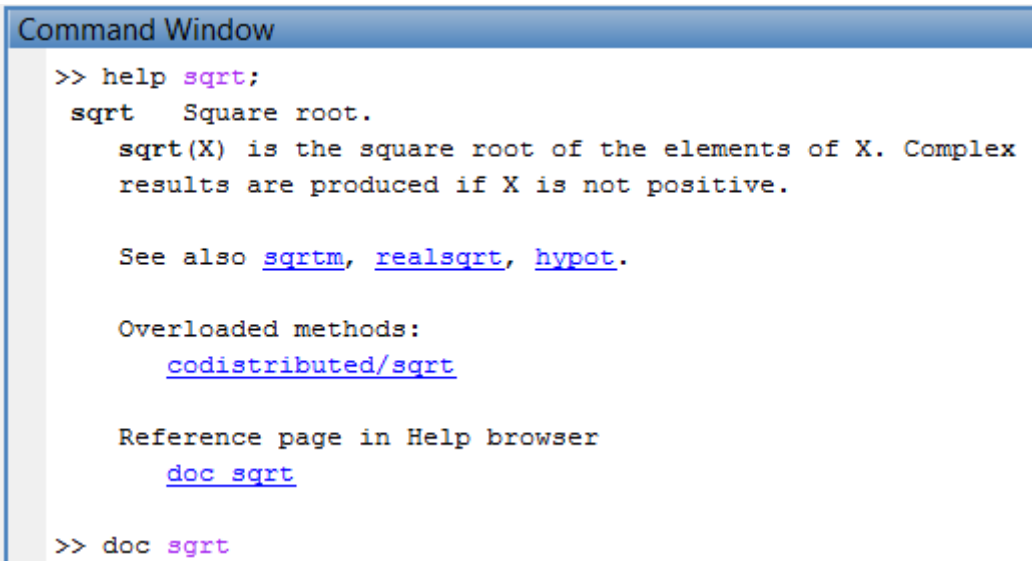



*Figure 13. Find information about the sqrt function*

This command (doc sqrt) will display the help documentation for the sqrt function, including its syntax, description, and usage examples. Similarly, you can use the help or doc command to explore other functions and topics within MATLAB to gain a deeper understanding of their functionalities and how to use them.

Remember, being able to effectively search for and use documentation is a valuable skill for programming in MATLAB.

## Part 03: Experimental Section (MATLAB - SIMULINK)

## Exercise 6: "Calculate operations"

In this exercise, you'll create a MATLAB program that takes three numbers as input and displays the product of the first two numbers and the square root of the product of the last two numbers. This exercise will reinforce your understanding of basic calculations and functions in MATLAB.

Follow these steps:

1. Open a new script file (Ctrl + N).
2. Write a program that takes three numbers as input, calculates the desired values, and displays the results.
3. Save the file with an appropriate name, such as "Sum_Product_Root.m".

Here's how you can structure your script for this exercise:

```
>>% Input three numbers
>>num1 = input('Enter the first number: ');
>>num2 = input('Enter the second number: ');
>>num3 = input('Enter the third number: ');
>>% Calculate product of the first two numbers
>>product_result = num1 * num2;
>>% Calculate square root of product of the last two numbers
>>root_result = sqrt(num2 * num3);
>>% Display the results
>>disp(['The product of ', num2str(num1), ' and ', num2str(num2), ' is ', num2str(product_result)]);
>>disp(['The square root of the product of ', num2str(num2), ' and ', num2str(num3), ' is ', num2str(root_result)]);
```

Create a new script file, add the above code, and run the script in MATLAB. It will prompt you to input three numbers and then display the calculated values as described.

Using this function in Editor Window :

```
function [] = Som2_TP1()
x=input('entrer le 1er nombre ')
y=input('entrer le 2e nombre ');
z=input('entrer le 3e nombre ');
a=x*y;
disp('^_^');
fprintf('le produit des 1er nombres est: %f \n', a)
b=sqrt(y*z);
fprintf ('la racine des deux dernier nombres est: %.2f \n', b);
disp('^_^');
end
```

*Figure 14. Calculate operations*

In this script we used 4 functions "**input(), sqrt(), dips() and fprintf()**",

- Use the command "help" to discover these functions.
- **What is the purpose of these functions?**

## Exercise 7: "Enter First and Last Name"

In this exercise, you'll create a MATLAB function that takes your first name and last name as input arguments and returns them on the same line. This exercise will help you practice creating and using functions with input and fprintf command in MATLAB.

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program in Editor that defines a function to concatenate your first name and last name and display them.
3. Save the file with an appropriate name, such as "Name_TP1.m".

Here's how you can structure your script to create the function:

**Script Version (FullNameConcatenation.m):**

```
1. % Concatenate first name and last name
2. F=input('enter your first name');
3. L= input('enter your last name');
4. fprintf(' full_name is : %s, %s ', F, L);
```

**Function Version (FullNameConcatenation.m):**

```
5. function Name_TP1 = Name_TP1(first_name, last_name)
6. % Concatenate first name and last name
7. full_name = [first_name, ' ', last_name];
8. end
```

- In this function, you define a function named *Name_TP1* that takes two input arguments first_name and last_name, concatenates them with a space in between, and returns the full name.
- Now, save the function file in the same folder and call this function with your first and last name as arguments to see your full name displayed on the same line.

For example, in the Command Window:

```
>> result = Name_TP1(Abdelkader, Amir);
>> disp(['Your full name is: ', result]);
result =
      Abdelkader Amir
```

## Exercise 8: "Enter First and Last Name + First-Year Overall Average"

In this exercise, you'll create a MATLAB function that takes your first name, last name, and your first-year overall average as input arguments and returns them on the same line. This exercise will help you practice creating functions with multiple input arguments in MATLAB.

Follow these steps:

1. Create a new script file (Ctrl + N).
2. Write a program that defines a function to concatenate your full name and your first-year overall average and then display them.
3. Save the file with the name: "Name_Moy_TP1.m".

# Lab Sheet 03
# Script Files and Data Types and Variables

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

# Lab Sheet 03: Script Files and Data Types and Variables

## TP Objective:

The objective of TP03 is to familiarize you with basic operations and variable types, as well as their use in solving exercises.

## Part 01: Theoretical Section (Data Types and Variables)

### Basic Tools

As mentioned before, the fundamental principle of MATLAB is to treat most objects as matrices. Therefore, common operations +, -, *, / should be understood as matrix operations.
In the following section, we will focus on these operations. Let's start by examining what happens with 1x1 matrices (which essentially represent single elements) [2.1].

### Data Types of Variables

The main variable types in MATLAB are:

1. Numeric Types:
    - **Double**: Represents floating-point numbers (default data type for most calculations).
    - **Single**: Represents single-precision floating-point numbers.
    - **Integers**: Represents whole numbers, including signed and unsigned variants of different bit depths.
2. Character and String Types:
    - **Char**: Represents individual characters.
    - **String**: Represents sequences of characters.
3. Logical Type:
    - **Logical**: Represents true (1) or false (0) values, used for logical operations.
4. Complex Numbers:
    - **Complex**: Combinations of real and imaginary parts.
5. Cell Arrays and Structures:
    - **Cell Array**: Stores data of different types and sizes.
    - **Structure**: Stores data using named fields.
6. Function Handles:
    - **Function Handle**: Stores references to functions.

To focus on a simplified overview of MATLAB's variable types, we can limit it to these three main types:

1. **NUMBERS:** This includes integers, real numbers, and complex numbers. Integers represent whole numbers, real numbers can have decimal parts, and complex numbers have both real and imaginary parts.
2. **CHARACTERS**: This refers to strings of characters, such as text. Strings are used to represent sequences of characters like names, sentences, etc.
3. **LOGICAL**: This involves the logical data type, which represents true (1) or false (0) values. Logical values are used in logical operations and decision-making processes.

As a result of focusing on these three fundamental variable types, we can simplify the introduction to MATLAB's capabilities for beginners [2.2].

Let's define a variable of each type:

- Variable 'a' represents a real number.
- Variable 'b' represents a complex number.
- Variable 'c' is a string (sequence of characters).
- Variables 'd1' and 'd2' are two ways of defining a logical variable (both set to TRUE in this case).
- Variable 'e' is an integer coded with 8 bits.

```
1    a = 3.14; b = 1+i; s = 'TP-InFo3';
2    d1 = true(1==1); d2 = logical(1);
3    e = int8(2);
```

*Figure 1. Variable of each type*

We can then check the type of these different variables using the whos function and verify their types using the functions ischar(variable), islogical(variable), and isreal(variable).

```
>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  b         1x1                16  double    complex
  d1        1x1                 1  logical
  d2        1x1                 1  logical
  e         1x1                 1  int8
  s         1x8                16  char
```

*Figure 2. Variables using the whos function*

Here's how you could implement this in MATLAB:

```
a = 3.14;          % Real number
b = 2 + 3i;        % Complex number
c = 'Hello';       % String
d1 = true;         % Logical variable (TRUE)
d2 = logical(1); % Another way to define a logical variable (TRUE)
e = int8(5);       % 8-bit integer
```

```
% Check variable types
type_a = isreal(a);
type_b = isreal(b);
type_c = ischar(c);
type_d1 = islogical(d1);
type_d2 = islogical(d2);
type_e = isinteger(e);
% Display results
fprintf('Variable a is real: %d\n', type_a);
fprintf('Variable b is real: %d\n', type_b);
fprintf('Variable c is a string: %d\n', type_c);
fprintf('Variable d1 is logical: %d\n', type_d1);
fprintf('Variable d2 is logical: %d\n', type_d2);
fprintf('Variable e is an integer: %d\n', type_e);
```

The output of this script will indicate the types of each variable, and you can see if they match your expectations.

## MATLAB Variables

Variable and function names in MATLAB are composed of letters and digits. MATLAB is case-sensitive, meaning that it distinguishes between uppercase and lowercase letters. For example, **INFO3**, **info3**, and **Info3** are considered different variables. If a variable already exists, its content will be overwritten by a new value assigned to that variable [2.3.1].

### Variable:

**In MATLAB, there exists a singular data type: the matrix data type.**

| Matrix | Vector | Scalar |
|---|---|---|
| m x n, m, n >1, | 1 x n, n >1, | 1 x 1 |

### Numbers or strings of characters:

| Variable name | Numbers | '1, 2, 3, ...' |
|---|---|---|
| Strings of characters | 1, 2, (1+i).... | Strings of characters |

## Classes and Format

By the syntax introduced above, MATLAB defines variables that belong to the double array class, which corresponds to arrays of real numbers that can represent scalars, vectors, or matrices. Apart from this fundamental class, it's worth mentioning that there are other predefined MATLAB classes. The most significant is undoubtedly the char array class, to which strings of characters belong, defined using '...'.

## Format and Codage

The following table lists the characters and available conversion subtypes in MATLAB.

*Table 1. Format and Codage*

| Value Type | Conversion | Details |
|---|---|---|
| Integer, signed | `%d` or `%i` | Base 10 |
| Integer, unsigned | `%u` | Base 10 |
| | `%o` | Base 8 (octal) |
| | `%x` | Base 16 (hexadecimal), lowercase letters `a–f` |
| | `%X` | Same as `%x`, uppercase letters `A–F` |
| Floating-point number | `%f` | Fixed-point notation |
| | `%e` | Exponential notation, such as `3.141593e+00` |
| | `%E` | Same as `%e`, but uppercase, such as `3.141593E+00` |
| | `%g` | The more compact of `%e` or `%f`, with no trailing zeros |
| | `%G` | The more compact of `%E` or `%f`, with no trailing zeros |
| | `%bx` or `%bX`<br>`%bo`<br>`%bu` | Double-precision hexadecimal, octal, or decimal value<br>Example: `%bx` prints `pi` as `400921fb54442d18` |
| | `%tx` or `%tX`<br>`%to`<br>`%tu` | Single-precision hexadecimal, octal, or decimal value<br>Example: `%tx` prints `pi` as `40490fdb` |
| Characters | `%c` | Single character |
| | `%s` | String of characters |

**We will use the following format codes for our practical exercises:**

***Table 2. Format codes for our practical exercises***

| Numbers | Format Conversion | Declaration de variable | Character | Format Conversion | Declaration de variable |
|---|---|---|---|---|---|
| les entiers | %d | int variable | Single | %c | char variable |
| les réels | %f | float variable | String | %s | |
| Classes | | | | | |
| Double / single | | | char | | |

### Display Format

By default, MATLAB displays results in decimal form. This format can be changed at any time using the **format** function [2.3.2].

*Table 3. Display Format*

| Command | Display | Example |
|---|---|---|
| **format short** | Decimal with 5 digits | *31.416* |
| **format long** | Decimal with 16 digits | *31.41592653535879* |
| **format bank** | Fixed-point with 2 decimal places | *31.41* |
| **format rat** | Fractional | *3550/113* |

### Arithmetic and Operations on Scalars

We have already performed basic operations using variables and functions in Lab sheet 01 and 02. You can test the following examples:

```
>> x = 2; y = 1.5 ;
somme = x+y
difference = x-y
produit = x*y
division = x/y

somme = 3.5000
difference = 0.5000
produit =  3
division = 1.3333
```

*Figure 3. Arithmetic and Operations on Scalars*

You can also work with MATLAB as a numerical calculator using trigonometric, power, logarithmic functions, etc.

*Table 4. Main mathematical function used in Matlab*

| | | | | | |
|---|---|---|---|---|---|
| `exp(x)` | : | exponentielle de x | `conj(z)` | : | conjugué de z |
| `log(x)` | : | logarithme néperien de x | `abs(z)` | : | module de z |
| `log10(x)` | : | logarithme en base 10 de x | `angle(z)` | : | argument de z |
| `x^n` | : | x à la puissance n | `real(z)` | : | partie réelle de z |
| `sqrt(x)` | : | racine carrée de x | `imag(z)` | : | partie imaginaire de z |
| `abs(x)` | : | valeur absolue de x | `rem(m,n)` | : | reste de la division entière de $m$ par $n$ |
| `sign(x)` | : | 1 si $x > 0$ et 0 si $x \leq 0$ | `lcm(m,n)` | : | plus petit commun multiple de $m$ et $n$ |
| `sin(x)` | : | sinus de x | `gcd(m,n)` | : | plus grand commun diviseur de $m$ et $n$ |
| `cos(x)` | : | cosinus de x | `factor(n)` | : | décomposition en facteurs premiers de $n$ |
| `tan(x)` | : | tangente de x | `round(x)` | : | entier le plus proche de x |
| `asin(x)` | : | sinus inverse de x (arcsin de x) | `floor(x)` | : | arrondi par défaut de x |
| `sinh(x)` | : | sinus hyperbolique de x | `ceil(x)` | : | arrondi par excés de x |
| `asinh(x)` | : | sinus hyperbolique inverse de x | | | |

## Part 02: Simulation Section (MATLAB – SIMULINK)

- Create a new folder inside the TP_INFO3 folder named: TP03_INFO3.
- Change the current folder (path) to TP03_INFO3 in MATLAB.

### Exercise 01 "Calculate S = 1 + 2 + ... + 10"

Create a program that calculates the sum from 1 to 10. (S = 1 + 2 + ... + 10)

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code below provided above into the script.
3. Save the script with the name "Sum_TP3.m".
4. Run the script by typing "Sum_TP3" in the MATLAB Command Window.

This program calculates the sum of numbers from 1 to 10 and displays the result:

```
% Exercise 01: Calculate S = 1 + 2 + ... + 10
% Initialize the sum variable
S = 0;
% Calculate the sum using a loop
for i = 1:10
    S = S + i;
end
% Display the result
fprintf('The sum of 1 to 10 is: %d\n', S);
```

### Exercise 02: "Calculate P = 3 * 4 * ... * 7"

Search for a program that calculates the product from 3 to 7. (P = 3 * 4 * ... * 7)

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script.
3. Save the script with the name "Prod_TP3.m".
4. Run the script by typing "Prod_TP3" in the MATLAB Command Window.

This program calculates the product of numbers from 3 to 7 and displays the result:

```
% Exercise 02: Calculate P = 3 * 4 * ... * 7
% Initialize the product variable
P = 1;
% Calculate the product using a loop
for i = 3:7
    P = P * i;
end
% Display the result
fprintf('The product of 3 to 7 is: %d\n', P);
```

## Exercise 03: "Calculate Square Root of a Number"

Create a program that calculates the square root of a number.

Now, let's implement this exercise in MATLAB:

```
% Exercise 03: Calculate Square Root of a Number
% Input a number
num = input('Enter a number: ');
% Calculate the square root
sqrt_num = sqrt(num);
% Display the result
fprintf('The square root of %.2f is %.2f\n', num, sqrt_num);
```

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script.
3. Save the script with the name "Rac_TP3.m".
4. Run the script by typing "Rac_TP3" in the MATLAB Command Window.

This program prompts the user to enter a number, calculates its square root, and then displays the result.

## Exercise 04: "Calculate the Determinant of a Quadratic Polynomial"

Create a program that calculates the determinant Δ = b^2 - 4ac of a quadratic polynomial.

Here's how you could implement this exercise in MATLAB:

```
% Exercise 04: Calculate the Determinant of a Quadratic Polynomial
% Input coefficients a, b, and c
a = input('Enter coefficient a: ');
b = input('Enter coefficient b: ');
c = input('Enter coefficient c: ');
% Calculate the determinant
delta = b^2 - 4*a*c;
% Display the result
fprintf('The determinant is: %.2f\n', delta);
```

Follow these steps:

2. Create a new script file using Ctrl + N.
3. Copy and paste the code provided above into the script.
4. Save the script with the name "Det_TP3.m".
5. Run the script by typing "Det_TP3" in the MATLAB Command Window.

This program prompts the user to input coefficients of a quadratic polynomial, calculates the determinant, and displays the result.

## Exercise 05: "Calculate the Solutions of a Quadratic Polynomial Equation"

Create a function that calculates the two solutions of a quadratic polynomial equation given the three coefficients (a, b, c).

Here's how you could implement this exercise in MATLAB:

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the function code provided above into the script (slvordr_TP3.m).
3. Save the script with the name " slvordr_TP3.m ".
4. Create another script or use the Command Window to input coefficients and call the function.
5. Run the script by typing the name of the script (e.g., " slvordr_TP3.m ") in the MATLAB Command Window.

**The function script "slvordr_TP3.m":**

```
% Exercise 05: Calculate the Solutions of a Quadratic Polynomial Equation
function [x1, x2] = solve_quadratic(a, b, c)
    % Calculate the discriminant
    delta = b^2 - 4*a*c;
    % Calculate the solutions
    x1 = (-b + sqrt(delta)) / (2*a);
    x2 = (-b - sqrt(delta)) / (2*a);
end
```

**We can use another script or directly in the Command Window:**

```
% Usage example
a = input('Enter coefficient a: ');
b = input('Enter coefficient b: ');
c = input('Enter coefficient c: ');
[x1, x2] = solve_quadratic(a, b, c);
fprintf('Solutions: x1 = %.2f, x2 = %.2f\n', x1, x2);
```

This program defines a function to calculate the solutions of a quadratic polynomial equation and then demonstrates its usage by taking coefficients as input and displaying the solutions.

## Exercise 06: "Solving a Quadratic Polynomial Equation"

Consider the quadratic polynomial: $P(x) = x^2 + 8x + 16$, where the polynomial coefficients are: $P = [1, 8, 16]$. In the general case, a quadratic polynomial can be written as $P(x) = ax^2 + bx + c$ with $P = [a, b, c]$, and the roots are given by $P(x) = (x - r_1)(x - r_2)$ with $r_{1,2} = \frac{(-b \pm \sqrt{b^2 - 4ac})}{2a}$.

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script (rootordr_TP3.m).
3. Save the script with the name «rootordr_TP3.m ".
4. Run the script by typing «rootordr_TP3.m " in the MATLAB Command Window.

This program calculates and displays the roots of the quadratic polynomial using the ***`roots()`*** function, constructs the coefficients using the ***`poly()`*** function, and compares them with the given polynomial coefficients.

We aim to use a MATLAB function to solve the quadratic polynomial:

1. Create a new script (Ctrl + N).
2. Save the script with the file name: rootordr_TP3.m.
3. Use the same coefficients proposed in the previous exercise.
4. Use the function **roots()** to solve the polynomial equation.
5. Compare the results obtained with your program's results. Draw conclusions.
6. Use the function **poly()** to construct the coefficients of a polynomial from these roots.

Here's how you could implement this exercise in MATLAB:

```
% Exercise 06: Solving a Quadratic Polynomial Equation
% Given polynomial coefficients
P = [1, 8, 16];
% Solve the polynomial equation using roots()
roots_P = roots(P);
% Display the roots
fprintf('Roots of the polynomial: %.2f, %.2f\n', roots_P(1), roots_P(2));
% Construct polynomial coefficients using poly()
coefficients = poly(roots_P);
disp('Coefficients of the polynomial:');
disp(coefficients);
```

## Exercise 07: "Solving and Evaluating a Polynomial"

Consider the polynomial:

$$P(x) = x^7 - 37x^6 + 555x^5 - 4295x^4 + 17924x^3 - 37668x^2 + 30240x$$

1. Create a new script (Ctrl + N).
2. Save the script with the file name: Pol_TP3.m.
3. Using MATLAB:
   - Solve P(x) = 0.
   - Calculate the polynomial P(x) for x = 2 and x = 3, Use the **polyval(p, x)** function.

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script (Pol_TP3.m).
3. Save the script with the name "Pol_TP3.m".
4. Run the script by typing "Pol_TP3" in the MATLAB Command Window.

This program solves the polynomial equation $P(x) = 0$ to find its roots, calculates the polynomial values for given x values using `***polyval***()`, and displays the results.

Here's how you could implement this exercise in MATLAB:

```
% Exercise 07: Solving and Evaluating a Polynomial
% Given polynomial coefficients
P = [1, -37, 555, -4295, 17924, -37668, 30240, 0];
% Solve the polynomial equation P(x) = 0
roots_P = roots(P);
% Display the roots
fprintf('Roots of the polynomial:\n');
disp(roots_P);
% Calculate the polynomial values for x = 2 and x = 3 using polyval()
x_values = [2, 3];
P_values = polyval(P, x_values);
% Display the calculated polynomial values
for i = 1:length(x_values)
    fprintf('P(%d) = %.2f\n', x_values(i), P_values(i));
end
```

## Exercise 08: "Polynomial Analysis"

Consider the roots of a polynomial: $R = [7, 8, 9, 5, 0, 2, 6]$

1. Create a new script (Ctrl + N).
2. Save the script with the file name: RacPol_TP3.m.
3. Using MATLAB:
   - Find the coefficients of the polynomial.
   - Determine the order of the polynomial.
   - Calculate the derivative of the polynomial function (using the polyder(p) function).
   - Calculate the integral of the polynomial function (using the polyint(p) function).

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script (RacPol_TP3.m).

3. Save the script with the name "RacPol_TP3.m".
4. Run the script by typing "RacPol_TP3" in the MATLAB Command Window.

This program analyzes the given polynomial roots to find its coefficients, determine its order, calculate its derivative and integral, and displays the results. Here's how you could implement this exercise in MATLAB:

```
% Exercise 08: Polynomial Analysis
% Given roots of the polynomial
R = [7, 8, 9, 5, 0, 2, 6];
% Find the coefficients of the polynomial using poly()
P_coefficients = poly(R);
% Display the coefficients
fprintf('Coefficients of the polynomial:\n');
disp(P_coefficients);
% Determine the order of the polynomial
order = length(P_coefficients) - 1;
fprintf('Order of the polynomial: %d\n', order);
% Calculate the derivative of the polynomial using polyder()
P_derivative = polyder(P_coefficients);
% Display the derivative coefficients
fprintf('Derivative coefficients of the polynomial:\n');
disp(P_derivative);
% Calculate the integral of the polynomial using polyint()
P_integral = polyint(P_coefficients);
% Display the integral coefficients
fprintf('Integral coefficients of the polynomial:\n');
disp(P_integral);
```

## Exercise 09: "Polynomial Operations"

Consider the following polynomials:

$$P_1(x) = x^3 + 3x^2 - 24x - 80$$

$$P_2(x) = x^2 - x - 20$$

$$P_3(x) = x + 4$$

1. Create a new script (Ctrl + N).
2. Save the script with the file name: divPol_TP3.m.
3. Find the coefficients of the given polynomials.
4. Using MATLAB:

- Calculate the roots of P_1(x) and P_2(x).
- Calculate the convolution product h(x) = P_2(x) * P_3(x) (using the conv(P2, P3) function).

- Calculate the deconvolution product h(x) = P_2(x) / P_3(x) (using the deconv(P1, P3) function).
- Draw conclusions.

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script (divPol_TP3.m).
3. Save the script with the name "divPol_TP3.m".
4. Run the script by typing "divPol_TP3" in the MATLAB Command Window.

This program performs polynomial operations, including finding roots, convolution, and deconvolution, and displays the results along with the coefficients of the given polynomials.

Here's how you could implement this exercise in MATLAB:

```
% Exercise 09: Polynomial Operations
% Given polynomial coefficients
P1 = [1, 3, -24, -80];
P2 = [1, -1, -20];
P3 = [1, 4];
% Find the coefficients of the polynomials
coeff_P1 = P1;
coeff_P2 = P2;
coeff_P3 = P3;
% Calculate the roots of P1(x) and P2(x) using roots()
roots_P1 = roots(coeff_P1);
roots_P2 = roots(coeff_P2);
% Display the roots
fprintf('Roots of P1(x): ');
disp(roots_P1);
fprintf('Roots of P2(x): ');
disp(roots_P2);
% Calculate the convolution product using conv()
convolution = conv(coeff_P2, coeff_P3);
fprintf('Convolution product coefficients:\n');
disp(convolution);
% Calculate the deconvolution product using deconv()
deconvolution = deconv(coeff_P2, coeff_P3);
fprintf('Deconvolution product coefficients:\n');
disp(deconvolution);
```

## Exercise 10: "Complex Numbers Calculation"

This program performs calculations on complex numbers, including calculating real and imaginary parts, conjugates, magnitudes, and arguments, and displays the results.

1. Enter the following complex numbers:

   $z_1 = 1 + i$ , $z_2 = z^2$ , $z_3 = e^{i\pi/4}$ ;

   Note: "$i$" *and* "$j$" are reserved MATLAB variables for complex numbers.
2. Using MATLAB:
   - Calculate the real part of each complex number and assign each part to a variable.
   - Calculate the imaginary part of each complex number and assign each part to a variable.
   - Calculate the conjugate of Z_1 and Z_3.
   - Calculate the magnitude (absolute value) of each complex number and assign it to another variable.
   - Calculate the argument (angle) of each complex number.
3. Utilize the complex functions mentioned in the Table 4: Main mathematical function used in MATLAB.

Follow these steps:

1. Create a new script file using Ctrl + N.
2. Copy and paste the code provided above into the script (nbrcmplx_TP3.m).
3. Save the script with the name "nbrcmplx_TP3.m".
4. Run the script by typing "nbrcmplx_TP3" in the MATLAB Command Window.

Here's how you could implement this exercise in MATLAB:

```
% Exercise 10: Complex Numbers Calculation
% Given complex numbers
z_1 = 1 + 1i;
z_2 = sym('z')^2;
z_3 = exp(1i * pi / 4);
% Calculate the real and imaginary parts
real_z_1 = real(z_1);
imag_z_1 = imag(z_1);
% Display the real and imaginary parts of z_1
fprintf('Real part of z_1: %f\n', real_z_1);
fprintf('Imaginary part of z_1: %f\n', imag_z_1);
% Calculate the conjugate of z_1 and z_3
conjugate_z_1 = conj(z_1);
conjugate_z_3 = conj(z_3);
% Display the conjugates
fprintf('Conjugate of z_1: %f + %fi\n', real(conjugate_z_1),
imag(conjugate_z_1));
```

```
fprintf('Conjugate of z_3: %f + %fi\n', real(conjugate_z_3),
imag(conjugate_z_3));
% Calculate the magnitudes of the complex numbers
magnitude_z_1 = abs(z_1);
magnitude_z_2 = abs(z_2);
magnitude_z_3 = abs(z_3);
% Display the magnitudes
fprintf('Magnitude of z_1: %f\n', magnitude_z_1);
fprintf('Magnitude of z_2: %s\n', char(magnitude_z_2));
fprintf('Magnitude of z_3: %f\n', magnitude_z_3);
% Calculate the arguments of the complex numbers
argument_z_1 = angle(z_1);
argument_z_2 = angle(z_2);
argument_z_3 = angle(z_3);
% Display the arguments
fprintf('Argument of z_1: %f radians\n', argument_z_1);
fprintf('Argument of z_2: %s radians\n', char(argument_z_2));
fprintf('Argument of z_3: %f radians\n', argument_z_3);
```

## Part 03: Experimental Section (MATLAB - SIMULINK)

### Exercise 11: Online Teaching Preparation

This script will help you perform the specified calculations and analyze the polynomial using MATLAB, Given the polynomial:

$$P(x) = x^5 - 2x^4 - 8x^3 + 16x^2 + 16x - 32$$

Perform the following calculations in MATLAB:

1. Evaluate the values of P(x) for the points x = 0, x = 1, and x = 2.
2. Calculate the derivative of P(x).
3. Calculate the integral (primitive) of P(x).
4. Calculate the polynomial of order 2, G(x), such that $G(x) = \frac{P(x)}{x^3-6x^2+12x-8}$

Save the script and execute it to verify the results of the calculations for the given polynomial P(x).

# Lab Sheet 04
# Vectors and Matrices

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

## Lab Sheet 04: Vectors and Matrices

### TP Objective:

The objective of TP04 is to familiarize you with operations on vectors and matrices, enabling you to use them to solve exercises.

## Part 01: Theoretical Part (Vectors: Lists and Arrays)

Matlab primarily employs lists (vectors) or arrays (matrices) for calculations. It is beneficial to learn how to manipulate these objects early on. In Matlab, a variable (a, x, etc.) or a list of numbers is a specific type of array.

To simplify, remember that in Matlab, everything is a matrix (and a matrix is an array). Although this might seem peculiar initially, it is what enables Matlab to be powerful and efficient in calculations [2.1].

### Constructing a Vector (a list)

You can define a list of numbers by providing its elements one after the other, separated by spaces or commas. The list is enclosed within square brackets [].

The distinction between comma, space, and semicolon:

- **Comma ( , )**: In Matlab, a comma is often used to separate elements within a vector or a matrix. For instance, [1, 2, 3] creates a vector with elements 1, 2, and 3.
- **Space**: Similarly, a space can be used to separate elements in a vector or a matrix. For example, [1 2 3] also creates a vector with elements 1, 2, and 3.
- **Semicolon ( ; )**: In contrast, a semicolon is employed to separate rows within a matrix. When used at the end of a row, it indicates that the next elements should belong to a new row. For instance [2.2.1]:

*Table 1. The distinction between comma, space, and semicolon*

| | |
|---|---|
| `>> P=[1, 8, 16]`<br>`P=[1 8 16]`<br><br>`P =`<br><br>`1 8 16` | `>> P=[1; 8; 16;]`<br><br>`P =`<br><br>`1`<br>`8`<br>`16` |
| This represents a vector containing the values 1, 8, and 16, arranged in a horizontal manner. | This represents a vector containing the values 1, 8, and 16, arranged in a vertical manner. Each element is placed on a new line to indicate the vertical arrangement. |

In this case, P is a matrix with one row and three columns, where the values 1, 2, and 3 are arranged horizontally.

| | | |
|---|---|---|
| 1 | 8 | 16 |

**Transpose of a Vector**

The transpose of a vector involves switching its rows and columns. If you have a vector arranged in a horizontal manner, the transpose will arrange its elements vertically. Similarly, if the vector is arranged vertically, the transpose will arrange its elements horizontally. This operation effectively changes the orientation of the vector while preserving its elements. The transpose operation converts a row into a column or vice versa:

```
>> vec1= [1 4 7]
>> vec1'
```

```
vec1 =

     1     4     7

>> vec1'

ans =

     1
     4
     7
```

*Figure 1. Transpose operation*

### Manipulating a Vector:

### Accessing Elements of a Vector:

You can extract individual elements from a vector. To access the element at index k of the vector a, you use $P(k)$.

For example, to retrieve the 3rd value of the vector P: $P(3)$

```
>> P(3)

ans =

    16
```

*Figure 2. Accessing Elements of a Vector*

### Example 1:

Consider a vector: $\boldsymbol{P_2(x) = x^5 - 2x^4 - 8x^3 + 16x^2 + 16x - 32}$

This polynomial function is of the 5th degree and is defined by its coefficients for each power of x.

The vector of coefficients for the polynomial P_2(x) is defined as:

$$P_2 = [1, -2, -8, 16, -32]$$

In this vector, each element corresponds to the coefficient of the corresponding power of xx in the polynomial $\boldsymbol{P_2(x)}$

```
>> P2=[1 -2 -8 16 -32]

P2 =

     1    -2    -8    16   -32
```

*Figure 3. Polynomial function is of the 5th degree*

*Table 2. Accessing Elements of a Vector*

| For example, to access the 2nd value of the vector $P_2$, you would use the following notation: **P2** : $P2(2)$ | To retrieve the first three values of the vector $P_2$, you can use the following notation: $P2(1:3)$ |
|---|---|
| `>> P2(2)`<br>`ans =`<br>`-2` | `>> P2(1:3)`<br>`ans =`<br>`1 -2 -8` |

To retrieve the values at odd indices of the vector **P2**, you can use the following notation: $P2(1:2:5)$

```
>> P2(1:2:5)

ans =

     1    -8   -32
```

*Figure 4. Retrieve the values at odd indices of the vector P2*

The expression $(a{:}\,p{:}\,b)$ creates a list whose elements range from ***a to b*** with a step of $\boldsymbol{p}$. When you don't provide the step, the default step value is 1. Accessing an element with a negative index leads to an error [2.3.3].

### The size of a vector

The $length()$ command returns the number of elements in a vector.

```
>> length(P2)

ans =

     5
```

*Figure 5. Size of a vector*

### Concatenate two vectors

To concatenate two vectors, you can use the square brackets [ ] notation.

For example: P3 = [P2, vec1].

The $linspace()$ function generates a vector of points between two values. The syntax $linspace(X1, X2, N)$ generates N points between X1 and X2. For N = 1, $linspace$ returns X2. For example: $P4 = linspace(1, 10, 10)$ generates a vector of 10 points evenly spaced between 1 and 10.

```
>> P4=linspace(1,10,10)

P4 =

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

*Figure 6. Concatenate two vectors*

### Constructing a matrix

You can create arrays with multiple rows by separating each row with a semicolon "**;**". For example: $P3 = [1\ -2\ -8\ 16\ -32\,;\ 1\ 2\ 3\ 4\ 5]$.

*Table 3. Constructing a matrix*

```
>> P3=[1 -2 -8 16 -32 ; 1 2 3 4 5]

P3 =

     1    -2    -8    16   -32
     1     2     3     4     5
```

```
>> P3=[1 -2 -8 16 -32 ; 1 2 3 4 5]

P3 =
          colonne 1   colonne 2                colonne 5
ligne 1       1          -2          -8          16          -32
ligne 2       1           2           3           4           5
```

### Accessing elements of a matrix

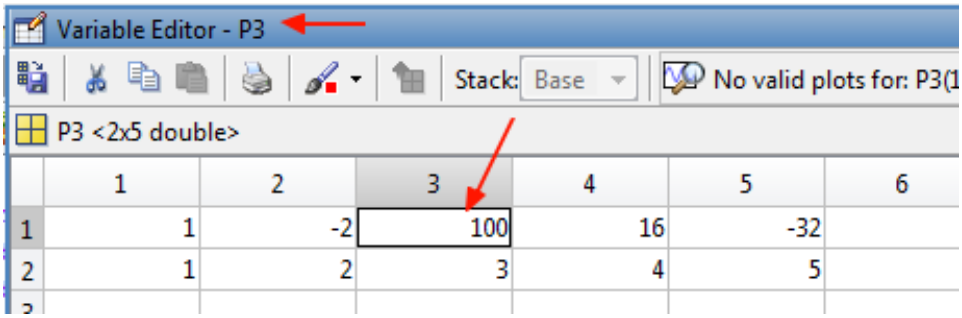
Similarly, you can extract parts of a matrix by specifying both the row and column indices. The first number indicates the row, and the second number indicates the column.

Example 2:

- To find the value of the element at row 1, column 3 of matrix P3: **P3(1, 3)**
- To find the values in the 4th column of matrix P3: **P3(:, 3)**
- To find the values in the 2nd row of matrix P3: **P3(2, :)**
- To find the values in the 1st row from the 3rd to 5th column of matrix P3: **P3(1, 3:5)**

### Changing an element in the matrix

- To modify the value of the element at row 1, column 3 to 100: **P3(1, 3) = 100**.
- Alternatively, in the Workspace, double-click on the matrix P3 (variable Editor), locate the element at row 1, column 3, change it to the value 100.




| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | -2 | 100 | 16 | -32 | |
| 2 | 1 | 2 | 3 | 4 | 5 | |

*Figure 7. Changing an element in the matrix*

You can run this code in the Command Window to see how the value of the element at (1,3) in the matrix **P3** changes to 100 :

```
% Create or define the matrix
P3 = [1 -2 -8 16 -32; 1 2 3 4 5];

% Change the value of element at (1,3) to 100
P3(1, 3) = 100;
```

## Part 02: Simulation Part (Matrix Operations)

To continue with the second part of the tutorial, you need to create a new folder named "TP04_INFO3" within the "TP_INFO3" directory and set it as the current folder in MATLAB. In this part, you'll explore various operations on matrices and vectors. MATLAB provides a wide range of operations, including addition, multiplication, transposition, and element-wise operations. For element-wise operations, you typically prefix the operator with a period ('.'). You will work with two matrices, A and B, as follows:

$$A = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix},\ A1 = \begin{pmatrix} 4 & 1 & 2 \\ 6 & 1 & 6 \\ 3 & 3 & 1 \end{pmatrix},\ B = \begin{pmatrix} 8 & 1 \\ 5 & 7 \end{pmatrix},\ V_1 = \begin{pmatrix} 5 \\ 4 \\ 6 \end{pmatrix},$$

$$V_2 = (2 \quad 1 \quad 6), V_3 = (4 \quad 9),$$

You can perform different matrix operations using these matrices. If you have any specific questions or tasks related to matrix operations, please let me know, and I'll be happy to assist you further.

## Exercise 1 (Concatenate, Compare matrices)

Using Matlab (Command Window space);

1. Matrix C includes matrix A and A1 horizontally: $C = [A, A1]$
2. Use the function $cat(dim, A, A1)$ to assemble the matrices A and A1:
$$C1 = cat(2, A, A1)$$
3. Compare the matrix C and C1, use the function: $isequal(C, C1)$
4. Compare matrix A and A1, use: $A == A1$
5. Conclude!!

## Exercise 2: Concatenate and Compare Matrices

In this exercise, you will practice concatenating and comparing matrices using MATLAB.

1. Create a new script in the "TP04_INFO3" folder and save it with the name "ConcatCompare_TP4.m".
2. Define two matrices, C and D, as follows:
$$C = \begin{pmatrix} 10 & 20 \\ 30 & 40 \end{pmatrix}; D = \begin{pmatrix} 50 & 60 \\ 70 & 80 \end{pmatrix};$$
3. Concatenate the matrices C and D horizontally and assign the result to a new matrix E.
$$E = [C, D];$$
4. Compare the matrices C and D element-wise and assign the result to a logical matrix F.
$$F = (C == D);$$
5. Display the matrices E and F using the "$disp$" function.

```
disp('Matrix E:');
disp(E);
disp('Matrix F (Element-wise comparison result):');
disp(F);
```

6. Run the script and observe the concatenated matrix E and the element-wise comparison result matrix F.

Feel free to modify and extend this exercise based on your learning needs. If you have any specific questions or tasks related to the exercise, please let me know, and I'll be here to assist you further.

## Exercise 3 (Add – Delete (row/column) in a matrix)

Using Matlab (Command Window space);

1. Calculate the dimension of the matrix A and B.
2. Calculate the number of elements in the vector V1 and V2.
3. Add the vector V1 in the 3rd column of matrix B.

*Table 4. Add – Delete (row/column) in a matrix*

| Method 1 | Method 2 |
|---|---|
| `>> B(1,3)=V1(1);`<br>`>> B(2,3)=V1(2);`<br>`>> B(3,3)=V1(3);`<br>`>> B`<br><br>`B =`<br><br>`8 1 5`<br>`5 7 4`<br>`0 0 6` | `>> B=[8 1; 5 7];`<br>`>> B(3,1)=0;`<br>`>> B=[B, V1]`<br><br>`B =`<br><br>`8 1 5`<br>`5 7 4`<br>`0 0 6` |

Here's how you can perform the tasks using MATLAB in the Command Window:

```
% Calculate dimensions
dim_A = size(A);
dim_B = size(B);
% Calculate number of elements in vectors
num_elements_V1 = numel(V1);
num_elements_V2 = numel(V2);
% Add vector V1 as a new column in matrix B
B_with_V1 = [B V1];
% Display results
disp("Dimension of matrix A:");
disp(dim_A);
disp("Dimension of matrix B:");
disp(dim_B);
disp("Number of elements in vector V1:");
disp(num_elements_V1);
disp("Number of elements in vector V2:");
disp(num_elements_V2);
disp("Matrix B with vector V1 added as a new column:");
disp(B_with_V1);
```

Copy and paste the above code into the MATLAB Command Window to execute the tasks. It will display the dimensions of matrices A and B, the number of elements in vectors V1 and V2, and the matrix B with vector V1 added as a new column.

```
>> B(3,:)=V2

B =

     8     1     5
     5     7     4
     2     1     6
```

*Figure 8. Add vector V2 in the 3rd row of matrix B.*

Here's how you can add vector V2 as a new row in the 3rd row of matrix B using MATLAB:

```
% Given matrices and vector
B = [8 1; 5 7];
V2 = [2 1 6];
% Add vector V2 as a new row in the 3rd row of matrix B
B_with_V2 = [B; V2];
% Display matrix B with vector V2 added as a new row
disp("Matrix B with vector V2 added as a new row:");
disp(B_with_V2);
```

Copy and paste the above code into the MATLAB Command Window to execute the task. It will display the matrix B with vector V2 added as a new row in the 3rd row.

*Table 5. Delete the second row of matrix A and assign the new matrix to C3*

| Method 1 | Method 2 |
|---|---|
| `>> C1=[A(1,:)];`<br>`>> C2=[A(3,:)];`<br>`>> C3=[C1; C2]`<br><br>`C3 =`<br><br>`8 1 6`<br>`4 9 2` | `>> A(2,:)=[]`<br><br>`A =`<br><br>`8 1 6`<br>`4 9 2`<br><br>`>> C3=A;` |

## Exercise 4 (Summation - product – transposition – square matrix)

1. Create **a new script** (Ctrl +N). Save with the name : **EX3_TP4.m**
2. Introduire les matrices dans le script.
3. Vérifier les matrices par l'exécution de programme  .
4. Calculer les dimensions de $A, A1, B, V1, V2$ *et* $V3$.
5. Calculer la matrice transposée de A et B : **transpose** (A), B'.
6. Calculer la matrice carrée de B : **B^2**.
7. A partir de la matrice A, extraire la sous matrice $A2 = (3{,}5\ ;\ 4{,}9)$
8. Calculer la somme des matrices A et A1 : $Som = A + A1$ ;

9. Calculer la somme des matrices B et A2 : $S = B + A2$ ;
10. Calculer la somme des matrices A et C3 : $Som1 = A + C3$ ;

```
>> Som1=A+C3
Error using  +
Matrix dimensions must agree.
```

**Here's how you can perform the operations described in Exercise 4 using MATLAB:**

*Table 6. Exercise 4 (Summation - product – transposition – square matrix)*

```
% Display matrices and vectors
disp("Matrix A:");
disp(A);
disp("Matrix A1:");
disp(A1);
disp("Matrix B:");
disp(B);
disp("Vector V1:");
disp(V1);
disp("Vector V2:");
disp(V2);
disp("Vector V3:");
disp(V3);

% Calculate dimensions
dim_A = size(A);
dim_A1 = size(A1);
dim_B = size(B);
dim_V1 = length(V1);
dim_V2 = length(V2);
dim_V3 = length(V3);
disp("Dimensions of A:");
disp(dim_A);
disp("Dimensions of A1:");
disp(dim_A1);
disp("Dimensions of B:");
disp(dim_B);
disp("Dimensions of V1:");
disp(dim_V1);
disp("Dimensions of V2:");
disp(dim_V2);
disp("Dimensions of V3:");
disp(dim_V3);

% Calculate transposed matrices
transpose_A = transpose(A);
transpose_B = B';
% Calculate square matrix
B_squared = B^2;
% Extract submatrix A2 from A
A2 = A(2:3, 1:2);
% Calculate matrix sum
Sum_A_A1 = A + A1;
Sum_B_A2 = B + A2;

% Calculate sum with
  incompatible dimensions (A +
  C3)
C3 = [1 2 3; 4 5 6; 7 8 9];
Sum_A_C3 = A + C3;
```

Copy and paste the above code into a new MATLAB script file (e.g., "EX3_TP4.m") and run it to perform the calculations and operations as described in the exercise.

To calculate the sum of the two matrices, the dimensions of the matrices must be the same.

- Calculate the product of the matrices A and V1: $prd = A * V1$ ;

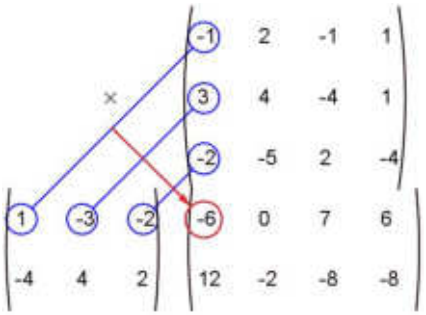



*Figure 9. Matrix Product*

For the product, the dimension of the column of the first matrix must be equal to the dimension of the row of the second matrix $(\boldsymbol{n},\boldsymbol{m}) * (\boldsymbol{n}',\boldsymbol{m}') \gg \boldsymbol{m} = \boldsymbol{n}'$.

- Manually check the result.

```
>> prd=A*V1

prd =

    80
    77
    68
```

*Figure 10. Matrix Product*

## Exercise 5 (specific matrices)

Using Matlab (Command Window space);

1. Identity matrix: use the command eye(n,m): 1) (n=3,m=1), 2) (n=3,m=2), 3) (n=3,m=3 ).
2. Null matrix: use the command zeros(n,m): 1) (n=3,m=1), 2) (n=3,m=2), 3) (n=3,m=3).
3. Unit matrix: use the command ones(n,m): 1) (n=3,m=1), 2) (n=3,m=2), 3) (n=3,m=3).
4. Random matrix: use the command rand(n,m): 1) (n=3,m=1), 2) (n=3,m=2), 3) (n=3,m=3).
5. Magic Matrix: use the magic(n) command.

Here's how you can create and work with different types of matrices using MATLAB:

```
% Matrice d'identité
identity_1 = eye(3, 1);
identity_2 = eye(3, 2);
identity_3 = eye(3, 3);
disp("Identity Matrix (n=3, m=1):");
disp(identity_1);
disp("Identity Matrix (n=3, m=2):");
disp(identity_2);
disp("Identity Matrix (n=3, m=3):");
disp(identity_3);

% Matrice nulle
zero_matrix_1 = zeros(3, 1);
zero_matrix_2 = zeros(3, 2);
```

```
% Matrice unitaire
ones_matrix_1 = ones(3, 1);
ones_matrix_2 = ones(3, 2);
ones_matrix_3 = ones(3, 3);
disp("Ones Matrix (n=3, m=1):");
disp(ones_matrix_1);
disp("Ones Matrix (n=3, m=2):");
disp(ones_matrix_2);
disp("Ones Matrix (n=3, m=3):");
disp(ones_matrix_3);

% Matrice aléatoire
random_matrix_1 = rand(3, 1);
random_matrix_2 = rand(3, 2);
```

```
zero_matrix_3 = zeros(3, 3);
disp("Zero Matrix (n=3, m=1):");
disp(zero_matrix_1);
disp("Zero Matrix (n=3, m=2):");
disp(zero_matrix_2);
disp("Zero Matrix (n=3, m=3):");
disp(zero_matrix_3);
```

```
random_matrix_3 = rand(3, 3);
disp("Random Matrix (n=3, m=1):");
disp(random_matrix_1);
disp("Random Matrix (n=3, m=2):");
disp(random_matrix_2);
disp("Random Matrix (n=3, m=3):");
disp(random_matrix_3);

% Matrice Magique
magic_matrix = magic(3);
disp("Magic Matrix (n=3):");
disp(magic_matrix);
```

## Exercise 6 (introducing a matrix (2,2))

1. Create a new script (Ctrl+N). Save with the name: EX5_TP4.m
2. Create a function that takes 4 numbers as inputs and returns an M(2,2) matrix as output.
3. Use the command zeros (n,m) to initiate the matrix M(2,2).

Here's an example of how you can create a MATLAB script to define a function that takes four numbers as input and returns a 2x2 matrix:

```
% Define the function to create a 2x2 matrix
function M = createMatrix(a, b, c, d)
    % Initialize a 2x2 matrix with zeros
    M = zeros(2, 2);
    % Assign the input values to the matrix elements
    M(1, 1) = a;
    M(1, 2) = b;
    M(2, 1) = c;
    M(2, 2) = d;
end
% Test the function with example values
a = 1;
b = 2;
c = 3;
d = 4;
result_matrix = createMatrix(a, b, c, d);
% Display the result matrix
disp("Resulting 2x2 Matrix:");
disp(result_matrix);
```

Then, run the script to define the function and test it with the provided example values. The function create Matrix takes four input values a, b, c, and d, and constructs a 2x2 matrix using

the zeros function to initialize the matrix and assigns the input values to its elements. Finally, the resulting matrix is displayed using the disp function.

## Exercice 7 (introduire une matrice (2,2) par l'utilisateur)

- Create **a new script** (Ctrl +N). Save with the name : **EX6_TP4.m**
- Réaliser une fonction qui prend 4 nombres en entrées par l'utilisateur et renvoie en sortie une matrice $M1(2,2)$. $M1 = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}$

  Utiliser la commande $m_{11} = \boldsymbol{input}$ ('$entre\ le\ 1er\ nombre$ ') ;

  Utiliser la commande $\boldsymbol{zeros}(\boldsymbol{n}, \boldsymbol{m})$ pour initier la matrice $M1(2,2)$.

## Exercice 7 (introduire une matrice (2,2) par l'utilisateur)

1. Create a new script (Ctrl +N). Save with the name : EX6_TP4.m
2. Réaliser une fonction qui prend 4 nombres en entrées par l'utilisateur et renvoie en sortie une matrice $M1(2,2)$. $M1 = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}$
   - Utiliser la commande m_11=input ('entre le 1er nombre ') ;
   - Utiliser la commande $zeros(n, m)$ pour initier la matrice M1(2,2).

Here's how you can create a MATLAB script to define a function that takes four numbers as input from the user and returns a 2x2 matrix:

```
% Define the function to create a 2x2 matrix from user input
function M1 = createMatrixFromUserInput()
    % Prompt the user to enter four numbers
    m_11 = input('Enter the 1st number: ');
    m_12 = input('Enter the 2nd number: ');
    m_21 = input('Enter the 3rd number: ');
    m_22 = input('Enter the 4th number: ');
% Initialize a 2x2 matrix with zeros
    M1 = zeros(2, 2);
% Assign the user input values to the matrix elements
    M1(1, 1) = m_11;
    M1(1, 2) = m_12;
    M1(2, 1) = m_21;
    M1(2, 2) = m_22;
end
% Call the function to create the matrix from user input
result_matrix = createMatrixFromUserInput();
% Display the result matrix
disp("Resulting 2x2 Matrix:");
disp(result_matrix);
```

When you run the script, the function `createMatrixFromUserInput` prompts the user to enter four numbers. It then constructs a 2x2 matrix using the user's input values and the `zeros` function to initialize the matrix. Finally, the resulting matrix is displayed using the `disp` function.

### Exercise 8 (introducing a matrix (3,3) by the user)

1. Create a new script (Ctrl+N). Save with the name: EX7_TP4.m
2. Create a function that takes 9 numbers as inputs by the user and returns a $M2(3,3)$ matrix as output.
    - Use the command $a_{11} = input\ ('enter\ the\ 1st\ number')$;
    - Use the command zeros(n,m) to initiate the matrix M2(3,3).

### Exercise 9 (Summation Matrix(2,2) + Matrix(2,2) by choice)

1. We are looking to create a function that sums two matrices A + B.
2. Create a new script (Ctrl+N). Save with the name: EX8_TP4.m

Here's how you can create a MATLAB script to define a function that adds two 2x2 matrices and returns the result:

```
% Define the function to add two 2x2 matrices
function sum_matrix = addMatrices(matrixA, matrixB)
    if size(matrixA) == [2, 2] && size(matrixB) == [2, 2]
        % Perform matrix addition
        sum_matrix = matrixA + matrixB;
    else
        disp('Both input matrices should be 2x2.');
        sum_matrix = [];
    end
end
% Example matrices A and B
A = [1, 2; 3, 4];
B = [5, 6; 7, 8];
% Call the function to add matrices A and B
result_matrix = addMatrices(A, B);
% Display the result matrix
disp("Resulting Sum Matrix:");
disp(result_matrix);
```

This script defines a function `addMatrices` that takes two input matrices and returns their sum if both matrices are 2x2. The example matrices A and B are provided, and the function is called to compute their sum. The result is displayed using the `disp` function.

## Exercise 10 (Matrix(3,3) x Vector(3,1) multiplication by choice)

1. Create a new script (Ctrl+N). Save with the name: EX9_TP4.m
2. We are looking to create a function that produces the product between matrix(3,3) and a vector(3,1) entered by the user and displays the result as output.
   - Use the command b11=input ('enter the 1st number');
   - Use the command zeros(n,m) to initiate the matrix M2(3,3).

here's how you can create a MATLAB script for Exercise 10, where you'll implement a function that performs the multiplication of a 3x3 matrix and a 3x1 vector entered by the user:

```
% Define the function to perform matrix-vector multiplication
function result_vector = matrixVectorMultiplication(matrix, vector)
    if size(matrix) == [3, 3] && size(vector) == [3, 1]
        % Perform matrix-vector multiplication
        result_vector = matrix * vector;
    else
        disp('Matrix should be 3x3 and vector should be 3x1.');
        result_vector = [];
    end
end

% Prompt the user to enter matrix elements
matrix = input('Enter a 3x3 matrix [a11, a12, a13; a21, a22, a23; a31, a32,
   a33]: ');

% Prompt the user to enter vector elements
vector = input('Enter a 3x1 vector [b1; b2; b3]: ');

% Call the function to perform matrix-vector multiplication
result = matrixVectorMultiplication(matrix, vector);

% Display the result vector
disp('Resulting Vector:');
disp(result);
```

This script defines a function `matrixVectorMultiplication` that takes a 3x3 matrix and a 3x1 vector as input and returns their multiplication result. The user is prompted to enter the matrix and vector elements, and the function is called to compute the multiplication. The result is displayed using the `disp` function.

## Part 03: Experimental part (MATLAB – SIMULINK)

### Exercise 11 (solving a matrix linear system AX=Y)

1. Create a new script (Ctrl+N). Save with the name: EX10_TP4.m
2. Create a function that solves the matrix equation AX=Y and displays the result of the X as output, with the matrix A(3,3), the vectors Y(3,1), X(3,1).
3. Measure the execution time of the program in seconds; use command tick; and knock;
   - Use the command a11=input ('enter the 1st number');
   - Use the command zeros(n,m) to initiate the matrix A(3,3).
   - Use the command zeros(n,m) to initiate the vector X(3,1).
   - Use the zeros(n,m) command to initiate the Y(3,1) vector.

### Exercise 12 preparation (determinant of a matrix)

1. Create a new script (Ctrl+N). Save with the name: EX11_TP4.m
2. Write a function that returns the determinant of an N(2,2) matrix given by the user.
   - Use the command var11=input ('enter the 1st number');
   - Use the zeros(n,m) command to initiate the N(2,2) matrix.

### Preparation exercise 13 (inverse of a matrix (2,2))

1. Create a new script (Ctrl+N). Save with the name: EX12_TP4.m
2. Write a function that inverts an L(2,2) matrix given by the user.
   - Use the command a11=input ('enter the 1st number');
   - Use the zeros(n,m) command to initiate the L(2,2) matrix.

### Exercise 14 of preparation (inverse of a matrix (3,3))

1. Create a new script (Ctrl+N). Save with the name: EX12_TP4.m
2. Realize a function that inverts a matrix G(3,3) given by the user.
   - Use the command G11=input ('enter the 1st number');
   - Use the command zeros(n,m) to initiate the matrix G(3,3).

# Lab Sheet 05
# Control Statements (Loops, if, and switch)

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

## Lab Sheet 05: Control Statements (Loops, if, and switch)

### TP Objective:

The objective of TP05 is to introduce you to instructions and control conditions. Conditional structures allow you to execute actions only if a pre-established test returns the value 'TRUE'.

### Part 01: Theoretical part (Conditional Instructions: if and switch)

Conditional instructions in MATLAB enable you to perform certain actions based on specific conditions. The most common structure is the "if" statement, which allows you to execute code if a condition is true. Here's how it's generally used [2.1]:

There are two possible commands for performing condition tests on the data.

- The ***if*** () statement is used to test the value of a variable and perform different processing depending on the cases tested.

```
if condition
    % Code to execute if the condition is true
else
    % Code to execute if the condition is false
end
```

The condition should be a logical expression that can be evaluated as either "true" or "false". If the condition is true, the code between the first "if" and "else" is executed. If the condition is false, the code between "else" and "end" is executed. There are variations of the "if" statement as well:

1. The "if-elseif-else" statement: Used when you have multiple conditions to check. The code under "elseif" is executed if the previous condition was false and the "elseif" condition is true.

```
if condition1
    % Code to execute if condition1 is true
elseif condition2
    % Code to execute if condition2 is true
else
    % Code to execute if none of the conditions are true
end
```

- The ***switch ()*** instruction makes it possible to choose between different cases, and to match a processing adapted to each of the recognized cases.

The "switch-case" statement: Used to perform different actions based on the value of an expression.

```
switch expression
    case value1
        % Code to execute if expression == value1
    case value2
        % Code to execute if expression == value2
    otherwise
        % Code to execute if none of the values match
end
```

- The ternary operator: A concise way to perform a simple condition in a single line.

```
variable = (condition) ? value_if_true : value_if_false;
```

This assigns "variable" the value "value_if_true" if the condition is true, otherwise the value "value_if_false". These conditional structures allow you to control the flow of execution in your code based on different logical conditions.

Les opérateurs de comparaison et les opérateurs logiques sont utilisés essentiellement dans les instructions de contrôle :

*Table 1. Comparison operators, logics*

| **Comparison (relational) operators** | **Logical operators** |
|---|---|
| • Strictly superior: (X > Y)<br>• Strictly lower: (X< Y)<br>• Compare two objects: (X= = Y)<br>• Greater than or equal: (X>= Y)<br>• Less than or equal: (X<= Y)<br>• Not equal to: (X~=Y) | • And : & : (X &Y)<br>• Or : \| : (X \| Y)<br>• Not : X (-X) |

## Example 1 (If-else-end conditional statements)

1. Create a new folder with name TP05_INFO3 in folder TP_INFO3.
2. Change Current folder to TP05_INFO3 in Matlab.
3. Create a new script (Ctrl +N). Save with the name: Example1_TP5.m
4. Assign the following values (5,1,10/2) to the successive variables (a,b,c):

*Table 2. Example 1: If-else-end conditional statements)*

| **Description :** | **Script :** |
|---|---|
| **Lines (1, 2, 3): Initiation of variable values.**<br>**Lines (5 to 8): test**<br>The if statement: tests the variable a is different from c (value!!)<br>If the value of a=5 is different from c=10/2=5<br>So calculate:<br>the new value of b =(old b=1)+1 =1+1=2<br>disp('b='): display in the CW space the expression: b=<br>disp(b): display the value of b: 2;<br>**Lines (9 to 14): against the test**<br>The else statement: "tests the variable a is equal to c"<br>No the value of a=5 is equal to c=10/2=5<br>***SO :***<br>disp(a), disp(c): display in the CW space the expression the value of the variable a and c.<br>b=b-1: calculate the new value of b and assign to b<br>disp('b='): display in the CW space the expression: b=<br>disp(b): display the value of b: 0;<br>**Line (15): end**<br>The end statement: terminates and closes the if condition. | 1 – a=5;<br>2 – b=1;<br>3 – c=10/2;<br>4<br>5 – if a~=c<br>6 – b=b+1;<br>7 – disp('b= ');<br>8 – disp(b);<br>9 – else<br>10 – disp(a)<br>11 – disp(c)<br>12 – b=b-1;<br>13 – disp('b= ');<br>14 – disp(b)<br>15 – end<br>*Figure 1. Example 1 (If-else-end conditional statements)* |

Here's the MATLAB code for Example 1:

```
% Example 1: Conditional instructions using if-else-end
a = 5;
b = 1;
c = 10/2;
if a > b
    fprintf('a is greater than b\n');
else
    fprintf('a is not greater than b\n');
end
if b == c
    fprintf('b is equal to c\n');
else
    fprintf('b is not equal to c\n');
end
```

In this example, we use the $\boldsymbol{if - else - end}$ structure to check conditions and execute different code blocks based on the results. The $\boldsymbol{fprintf}$ function is used to display messages in the Command Window [2.2].

## Example 2 (Instructions conditionnées if-elseif-end)

- Create **a new script** (Ctrl +N). Save with the name : **Exemple2_TP5.m**
- On cherche une fonction qui affiche une matrice carrée de taille **n ; soit zeros(), ones() ou bien rand() selon les entrée 1 ou 2 ou 3.**
- Utiliser l'instruction ***if***.

## Example 2 (if-elseif-end conditional statements)

1. Create a new script (Ctrl+N). Save with the name: Example2_TP5.m
2. We are looking for a function that displays a square matrix of size n; either zeros(), ones() or rand() depending on entry 1 or 2 or 3.
3. Use the if statement.

Here's the MATLAB code for Example 2:

```
function [M]= Exemple2_TP5 ()
n=input('entrer la taille de la matrice ');
var=input('entrer une valeur pour la matrice : 1=zeros, 2=ones, 3=rand ');

if var == 1
    M = zeros(n);
elseif var == 2
    M = ones(n);
elseif var == 3
    M = rand(n);
else
 error('numero d''exemple non prevu ...');
end

end
```

*Figure 2. Example 2 (if-elseif-end conditional statements)*

```
% Example 2: Conditional instructions using if-elseif-end
n = input('Enter the size of the square matrix (1, 2, or 3): ');
if n == 1
    matrix = zeros(n);
elseif n == 2
    matrix = ones(n);
elseif n == 3
    matrix = rand(n);
else
    fprintf('Invalid input! Please enter 1, 2, or 3.\n');
    matrix = [];
end
if ~isempty(matrix)
    disp('Generated matrix:');
    disp(matrix);
end
```

In this example, the user inputs a value n, and based on the value entered (1, 2, or 3), the program generates a square matrix using zeros(), ones(), or rand() functions. The if-elseif-end structure is used to handle multiple conditions. If an invalid input is given, an error message is displayed.

## Example 3 (Conditioned instructions, Numbers: switch … case … end)

1. Create a new script (Ctrl+N). Save with the name: Example3_TP5.m
2. We are looking for a function that displays a square matrix of size n either zeros(), ones() or rand() depending on the input 1 or 2 or 3.
3. Use the switch statement.

Here's the MATLAB code for Example 3:

```
function [M]= Exemple3_TP5 ()
n=input('entrer la taille de la matrice ');
var=input('entrer une valeur pour la matrice : 1=zeros, 2=ones, 3=rand ');
switch var
    case var==1
        M = zeros(n);
    case var == 2
        M = ones(n);
    case var == 3
        M = rand(n);
    otherwise
        error('numero d''exemple non prevu ...');
end
end
```

*Figure 3. Example 3 (Conditioned instructions, Numbers: switch … case … end)*

Sure, here's the MATLAB code for Example 3 using the switch statement:

```
% Example 3: Conditional instructions using switch-case-end
n = input('Enter the size of the square matrix (1, 2, or 3): ');
switch n
    case 1
        matrix = zeros(n);
    case 2
        matrix = ones(n);
    case 3
        matrix = rand(n);
    otherwise
        fprintf('Invalid input! Please enter 1, 2, or 3.\n');
        matrix = [];
end
if ~isempty(matrix)
    disp('Generated matrix:');
    disp(matrix);
end
```

In this example, the user inputs a value n, and the switch statement is used to handle different cases based on the value entered (1, 2, or 3). Depending on the case, the program generates a square matrix using zeros(), ones(), or rand() functions. If an invalid input is given, an error message is displayed.

## Part 02: Simulation Part (Conditions)

### Exercise 1 (test of a real variable: if)

1. Create a new script (Ctrl +N). Save with the name: EX1_TP5.m
2. Create a function that tests a real variable an if positive, zero or negative and returns it by display.
3. Use the if statement.

Here's the MATLAB code for Exercise 1:

```
% Exercise 1: Test of a real variable using if
x = input('Enter a real number: ');
if x > 0
    disp('The number is positive.');
elseif x == 0
    disp('The number is zero.');
else
    disp('The number is negative.');
end
```

In this exercise, the user inputs a real number x, and the program uses the if statement along with elseif and else to determine if the number is positive, zero, or negative. It then displays the corresponding message based on the condition.

### Exercise 2 (test of a real variable: switch)

1. Create a new script (Ctrl +N). Save with the name: EX2_TP5.m
2. Create a function that tests a real variable a whether positive, zero or negative and returns by display.
3. Use the switch statement.

Here's the MATLAB code for Exercise 2:

```
% Exercise 2: Test of a real variable using switch
a = input('Enter a real number: ');
switch sign(a)
    case 1
        disp('The number is positive.');
    case 0
        disp('The number is zero.');
    case -1
        disp('The number is negative.');
    otherwise
        disp('The input is not a real number.');
end
```

In this exercise, the user inputs a real number a, and the program uses the switch statement along with the $sign()$ function to determine if the number is positive, zero, or negative. It then displays the corresponding message based on the condition. The otherwise statement is used to handle cases where the input is not a real number.

## Exercise 3 (Saving a file)

1. Create a new script (Ctrl +N). Save with the name: EX3_TP5.m
2. Make a function that displays "Do you want to save?" write Y or N:" and it returns by display the recording or the cancellation of the file.
3. Use the if statement.
4. Reuse the switch statement.

Here's the MATLAB code for Exercise 3:

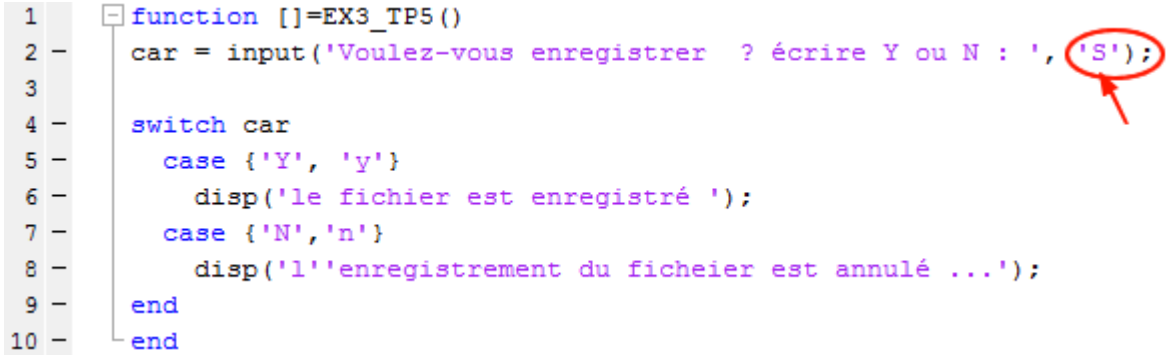



*Figure 4. Saving a file*

```
% Exercise 3: File Saving Confirmation
choice = input('Do you want to save? Type Y or N: ', 's');
if strcmpi(choice, 'Y')
    disp('File saved.');
elseif strcmpi(choice, 'N')
    disp('File not saved.');
else
    disp('Invalid choice.');
end
```

In this exercise, the program asks the user whether they want to save a file by displaying a prompt. The user's input is stored in the variable choice. The program then uses an if statement to check whether the user chose 'Y' (yes) or 'N' (no) and displays a message accordingly. The strcmpi() function is used for case-insensitive comparison. If the input is neither 'Y' nor 'N', an "Invalid choice" message is displayed.

## Part 03: Experimental part (MATLAB – SIMULINK)

### Exercise 4 (test of inversion of a matrix)

1. Create a new script (Ctrl+N). Save with the name: EX4_TP5.m
2. Create a random matrix of size 3. Am=rand(3);
3. Create a function that returns the inversion of a matrix if possible or inversion error if the determinant of a matrix equals 0.
4. Use statement: if
5. Use instruction: inv(matrix)
6. Use instruction: error('reverse error')
7. For the test: Use the following square matrix: M=[2,4,6;3,-8,-5;5,-4,1].

### Preparation exercise 5 (Adding the matrix inversion test to EX12_TP4)

1. Create a new script (Ctrl+N). Save with the name: EX5_TP5.m
2. Copy the script for exercise EX12_TP4.m into the script for EX5_TP5.
3. Modify the script by adding the function that returns the inversion test of a matrix if possible or inversion error if the determinant of a matrix equals 0.
4. Use instruction: switch.
5. For the test: Use the following square matrix: M=[2,4,6;3,-8,-5;5,-4,1].

### Exercise 6 preparation (Add a test for polynomial 2nd degree)

1. Create a new script (Ctrl+N). Save with the name: EX6_TP5.m
2. Copy the script for exercise 2ordr_TP3.m into the script for EX6_TP5.
3. Let's create a program that finds the roots of a quadratic equation ax^2+bx+c=0 with the conditions of the discriminant.

# Lab Sheet 06
# Function Files

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : Computer Science 3

## Lab Sheet 06: Function Files "loop operations"

### TP Objective:

The objective of TP06 is to familiarize you with control statements and loop operations. The loop operation enables the repetition of a specific command multiple times while varying a parameter.

## Part 01: Theoretical part (Loop: for and while)

### The statement: for ...end

The "for" statement in MATLAB is used to execute one or multiple MATLAB instructions within a repetitive loop, based on one or more conditions. It is particularly useful for defining sequences through recursion, as we will see [2.1].

### For Loop Body:

- Index: A variable called the loop index.
- Lower Bound and Upper Bound: Two real constants known as loop parameters.
- Sequence of Instructions: The set of operations to be executed for index values ranging from the lower bound to the upper bound with an increment of 1.

- To force an immediate exit from the loop, use the break or return statement.
- To iterate over the values of a single column vector, transpose it first to create a row vector [2.2].

**Syntaxe**

```
for indice = borne_inf : borne_sup
Séquence d'instructions
end
```

*Figure 1. Syntaxe for … end*

## Example 1 "Counts function"

1. Create a new script (Ctrl + N). Save with the name: Example1_TP6.m
2. Develop a MATLAB function that counts from 1 to 10.
3. Utilize the for loop instruction.

Here's the MATLAB code for Example 1:

```
% Example 1: Counting from 1 to 10 using a for loop
for i = 1:10
    disp(i);
end
```

When you run this code, it will display the numbers from 1 to 10 in the MATLAB Command Window.

## Example 2 " with loop statement"

1. Create a new script (Ctrl+N). Save with the name: Example2_TP6.m
2. Use instruction: for.
3. Write a function that displays the number of iterations from 0 to n.
4. Modify the display function from n/2 to n.

Here's the MATLAB code for Example 2:

```
% Example 2: Displaying iterations using a for loop
n = 10; % Change n to the desired value
% Display numbers from 0 to n
disp('Numbers from 0 to n:');
for i = 0:n
    disp(i);
end
% Display numbers from n/2 to n
disp(['Numbers from ' num2str(n/2) ' to ' num2str(n) ':']);
for i = n/2:n
    disp(i);
end
```

Replace the value of **n** with the desired number of iterations. When you run this code, it will display the numbers in the specified range using a for loop [2.3.3].

- Write a function that displays the number of iterations from 0 to n.

```
1      function []= Exemple2_TP6()
2
3 -    n=input('entrer n : ');
4 -    for i=0:n
5 -        disp(i)
6 -    end
```

*Figure 2. Function that displays the number of iterations from 0 to n*

- Modify the display function from n/2 to n.

```
1      function []= Exemple2_TP6()
2
3 -    n=input('entrer n : ');
4 -    for i=n/2:n
5 -        disp(i)
6 -    end
```

*Figure 3. Function display function from n/2 to n.*

## Example 3

1. Create a new script (Ctrl+N). Save with the name: Example3_TP6.m
2. Use instruction: for.
3. We are looking for a function that calculates the sum of 1 to n.

$$S = 1 + 2 + 3 + 4 + \cdots + n$$

4. Modify the function, We are trying to calculate the sum of a to b.

$$S_{ab} = a + 1 + 2 + \cdots + b$$

Here's the MATLAB code for Example 3:

```
% Example 3: Calculating the sum using a for loop
n = 10; % Change n to the desired value
% Calculate the sum of 1 to n
sum_1_to_n = 0;
for i = 1:n
    sum_1_to_n = sum_1_to_n + i;
end
disp(['Sum of 1 to ' num2str(n) ': ' num2str(sum_1_to_n)]);
% Calculate the sum of squares from 1 to n^2
sum_squares = 0;
for i = 1:n
    sum_squares = sum_squares + i;
end
disp(['Sum of squares from 1 to ' num2str(n) '^2: '
num2str(sum_squares)]);
```

Replace the value of `n` with the desired number. This code calculates the sum of 1 to n and the sum of squares from 1^2 to n^2 using a for loop.

- We are looking for a function that calculates the sum of 1 to n.

```
1   function [] = Exemple3_TP6 ()
2
3 -   n = input('n= ');
4     % Initiation de S avec valeur 0
5 -   S = 0 ;
6 -   for i=1:n
7 -   S = S+i;
8 -   end
9 -   disp(S);
```

*Figure 4. Function that calculates the sum of 1 to n.*

- Modify the function, We are trying to calculate the sum of a to b.

```
1   function [] = Exemple3_TP6 ()
2
3 -   a = input('a= ');
4 -   b = input('b= ');
5
6     % Initiation de S avec valeur a
7 -   S = a ;
8 -   for i=a+1:b
9 -       S = S+i;
10 -  end
11 -  disp(S);
```

*Figure 5. calculate the sum of a to b*

## The statement: *while ... end*

The "while" statement executes one or more instructions while a condition is true.

### While loop body

- Logical expression: is an expression whose result can be true or false;
- Statement sequence: is the processing to be performed as long as the logical expression is true.

**Syntaxe :**

***while*** expression logique

Séquence d'instructions

***end***

*Figure 6. Syntaxe **while ... end***

**Note**

Logical expression is usually the result of a test (eg i < Imax) or the result of a logical function (eg all(x)). It is imperative that the processing of the sequence of instructions acts on the result of the logical expression, otherwise we loop indefinitely.

## Example 4

1. Create a new script (Ctrl+N). Save it with the name: Example4_TP6.m
2. Use the "while" statement.
3. Create a MATLAB function that counts from 1 to 10 using the code from Example1_TP6.
4. Write a MATLAB function that calculates the sum of numbers from 1 to n using the code from Example3_TP6.

The MATLAB code for the tasks mentioned in Example4_TP6:

```
% Example4_TP6.m
% Create a MATLAB function that counts from 1 to 10
function countToTen()
    num = 1;
    while num <= 10
        disp(num);
        num = num + 1;
    end
end

% Call the function to count from 1 to 10
countToTen();

% Create a MATLAB function that calculates the sum of numbers from 1 to n
function sumResult = calculateSum(n)
    sumResult = 0;
    num = 1;
    while num <= n
        sumResult = sumResult + num;
        num = num + 1;
    end
end

% Call the function to calculate the sum of numbers from 1 to 10
n = 10;
result = calculateSum(n);
disp(['The sum of numbers from 1 to ' num2str(n) ' is: '
num2str(result)]);
```

Save this code in a file named "Example4_TP6.m" and run it in MATLAB. It will execute the tasks described in the example.

- Create a MATLAB function that counts from 1 to 10 using the code from Example1_TP6.

```
function [] = Exemple4_TP6 ()
i=0;
while i<10
    fprintf('i=%d ', i);
    i=i+1;
end
```

*Figure 7. Counts from 1 to 10 using while*

- Write a MATLAB function that calculates the sum of numbers from 1 to n using the code from Example3_TP6.

```
function [] = Exemple4_TP6 ()

n=input('n= ');
i=0;
k=0;
while i<=n
    k=k+i;
    fprintf('i=%d ', i);
    i=i+1;
end
fprintf('\n la somme = %d \n', k);
end
```

*Figure 8. Counts from 1 to n using while*

## Example 5

- Create **a new script** (Ctrl +N). Save with the name : **Exemple5_TP6.m**
- Use instruction: **while**.
- Réaliser une fonction Matlab qui calcule le produit de 1 jusqu'à $n$.
- Réaliser une fonction Matlab qui permet de calculer la factorielle de $n$.

## Example 5

1. Create a new script (Ctrl+N). Save with the name: Example5_TP6.m
2. Use statement: while.
3. Create a Matlab function that calculates the product of 1 to n.
4. Create a Matlab function that calculates the factorial of n.

The MATLAB code for Example 5 tasks:

```
% Example5_TP6.m
% Create a MATLAB function to calculate the product of numbers from 1 to
n
function productResult = calculateProduct(n)
    productResult = 1;
    num = 1;
    while num <= n
        productResult = productResult * num;
        num = num + 1;
    end
end

% Create a MATLAB function to calculate the factorial of n
function factorialResult = calculateFactorial(n)
    if n == 0
        factorialResult = 1;
    else
        factorialResult = 1;
        num = 1;
        while num <= n
            factorialResult = factorialResult * num;
            num = num + 1;
        end
    end
end

% Call the functions to calculate and display results
n = 5;
productResult = calculateProduct(n);
factorialResult = calculateFactorial(n);

disp(['Product of numbers from 1 to ' num2str(n) ': '
num2str(productResult)]);
disp(['Factorial of ' num2str(n) ': ' num2str(factorialResult)]);
```

Save this code in a file named "Example5_TP6.m" and run it in MATLAB. It will execute the tasks described in the example.

Create a Matlab function that calculates the product of 1 to n.

```
function [] = Exemple5_TP6 ()
disp('calcule de produit de 1 jusqu''a n');
n=input('n= ');
i=1;
k=1;
while i<=n
    k=k*i;
    fprintf('i=%d ', i);
    i=i+1;
end
fprintf('\n le produit de 1 à %d égal = %f \n', n, k);
end
```

*Figure 9. Matlab function that calculates the product of 1 to n*

Create a Matlab function that calculates the factorial of n.

```
function [] = Exemple5_TP6 ()
disp('calcule de produit de 1 jusqu''a n');
n=input('n= ');
i=1;
k=1;
if n==0
    fprintf('\n la factorielle de %d égal = 1 \n', n);
else
while i<=n
    k=k*i;
    fprintf('i=%d ', i);
    i=i+1;
end
fprintf('\n la factorielle de %d égal = %f \n', n, k);
end
end
```

*Figure 10. Matlab function that calculates the factorial of n*

## Part 02: Simulation Part (Conditions)

### Exercise 1 (display elements of any vector)

1. Create a new script (Ctrl +N). Save with the name: Exercise1_TP6.m
2. Realize a function that displays the elements of a vector one-by-one on the same line.
3. Create a vector of 1 to n numbers (n<20).
4. Use statement: for/while.

The MATLAB code for Exercise 1:

```
% Exercise1_TP6.m

% Create a MATLAB function to display elements of a vector on the same
line
function displayVectorElements(vector)
    for i = 1:length(vector)
        fprintf('%d ', vector(i));
    end
    fprintf('\n');
end

% Create a vector of 1 to n numbers (n < 20)
n = 10;
vector = 1:n;

% Call the function to display vector elements
displayVectorElements(vector);
```

Save this code in a file named "Exercise1_TP6.m" and run it in MATLAB.

It will create a vector of numbers from 1 to 10 and then display the elements of the vector on the same line. You can modify the value of `n` as needed.

vector of 1 to n numbers :

```
function [] = Exercice1_TP6 ()
disp('affichage des éléments d''un vecteur de 1 à n');
n=input('n= ');
P=1:1:n;

i=1;
while i<=n
    fprintf('l''element %d est : P(%d)= %.f \n', i, i,P(i));
    i=i+1;
end

end
```

*Figure 11. vector of 1 to n numbers*

## Exercise 2 (display elements of matrix)

1. Create a new script (Ctrl+N). Save with the name: Exercise2_TP6.m
2. Create a matrix M=[1 2 3; 3 4 5; 6 7 8];
3. Realize a function that displays the elements of a matrix one-by-one.
4. Use loops: for or while (you must use 2 loops).

The MATLAB code for Exercise 2:

```
% Exercice2_TP6.m

% Create a matrix M
M = [1 2 3; 3 4 5; 6 7 8];

% Function to display elements of a matrix one-by-one
function displayMatrixElements(matrix)
    [rows, cols] = size(matrix);
    for i = 1:rows
        for j = 1:cols
            fprintf('%d ', matrix(i, j));
        end
        fprintf('\n');
    end
end

% Call the function to display matrix elements
displayMatrixElements(M);
```

Save this code in a file named "Exercice2_TP6.m" and run it in MATLAB.

It will create a matrix `M` and then display its elements one-by-one using nested loops.

You can modify the matrix `M` as needed.

Function that displays the elements of a matrix one-by-one

```
1   function [] = Exercice2_TP6 ()
2 -   clc;
3 -   disp('affichage des éléments un-par-un d''une matrice');
4 -   M=[2 3 4; 5 6 7; 8 9 10];
5 -   disp(M);
6 -   for i=1:3
7 -       for j=1:3
8 -       fprintf('M(%d,%d)= %.f \n',i,j,M(i,j));
9 -       end
10 -  end
11 -  end
```

*Figure 12. Function that displays the elements of a matrix one-by-one*

## Exercise 3 (display elements of matrix)

1. Create a new script (Ctrl +N). Save with the name: Exercise3_TP6.m
2. Write a function that displays the elements of an n-dimensional random matrix one-by-one.

The MATLAB code for Exercise 3:

```
% Exercice3_TP6.m

% Function to display elements of a matrix one-by-one
function displayMatrixElements(matrix)
    [rows, cols] = size(matrix);
    for i = 1:rows
        for j = 1:cols
            fprintf('%f ', matrix(i, j));
        end
        fprintf('\n');
    end
end

% Generate a random matrix of dimension n
n = 3; % You can change this value to the desired dimension
randomMatrix = rand(n);

% Call the function to display random matrix elements
displayMatrixElements(randomMatrix);
```

Save this code in a file named "Exercice3_TP6.m" and run it in MATLAB. It will generate a random matrix of dimension `n` (you can change the value of `n` as needed) and then display its elements one-by-one using nested loops.

*Displays the elements of an n random matrix:*

```
n=input('dimension de la matrice carrée égal');
M=rand(n);
```

*Figure 13. Displays the elements of an n random matrix*

## Exercise 4 (display elements of matrix)

1. Create a new script (Ctrl +N). Save with the name: Exercise4_TP6.m
2. Realize a function that displays the 3rd row of a magic matrix of dimension 5.

The MATLAB code for Exercise 4:

```
% Exercise4_TP6.m

% Function to display the 3rd row of a magic matrix
function displayThirdRowMagicMatrix()
    n = 5; % Dimension of the magic matrix
    magicMatrix = magic(n);

    thirdRow = magicMatrix(3, :);

    fprintf('3rd Row of Magic Matrix:\n');
    disp(thirdRow);
end

% Call the function to display the 3rd row of the magic matrix
displayThirdRowMagicMatrix();
```

Save this code in a file named "Exercise4_TP6.m" and run it in MATLAB. It will generate a magic matrix of dimension 5 and then display its 3rd row using indexing.

Function that displays the 3rd row of a magic matrix :

```
>> A=magic(5);
for j=1:5
fprintf('A(5,%d)=%.f, ', j,A(5,j));
end
A(5,1)=11, A(5,2)=18, A(5,3)=25, A(5,4)=2, A(5,5)=9,
```

*Figure 14. function that displays the 3rd row of a magic matrix*

## Exercise 5 (introduce a matrix element by element)

1. Create a new script (Ctrl +N). Save with the name: Exercise5_TP6.m
2. Realize a function which introduces element-by-element a square matrix of dimension n.
3. Use statement: for and while.

The MATLAB code for Exercise 5:

```
% Exercise5_TP6.m
% Function to introduce a square matrix element by element
function introduceMatrix()
    n = input('Enter the dimension of the square matrix: ');
    matrix = zeros(n, n); % Initialize a matrix of zeros
    for i = 1:n
        for j = 1:n
            matrix(i, j) = input(['Enter element at position ('
num2str(i) ',' num2str(j) '): ']);
        end
    end
    fprintf('Matrix entered:\n');
    disp(matrix);
end
% Call the function to introduce a matrix element by element
introduceMatrix();
```

Save this code in a file named "Exercise5_TP6.m" and run it in MATLAB. It will prompt you to enter the dimension of the square matrix and then allow you to input each element of the matrix one by one. Finally, it will display the entered matrix.

```
1   function [] = Exercice5_TP6 ()
2   clc;
3   disp('introduce a matrix element by element');
4   n=input('entrer nbr des lignes ');
5   m=input('entrer nbr des colonnes ');
6   M=zeros(n,m);
7
8   for i=1:n
9       for j=1:m
10          fprintf(' l''element Matrix(%d,%d)= ',i,j);
11          M(i,j)=input('');
12      end
13  end
14  fprintf('la matrice(%d,%d) introduire égal \n',i,j);
15  disp(M);
16  end
```

*Figure 15. Introduce a matrix element by element*

## Part 03: Experimental part (MATLAB – SIMULINK)

### Exercise 6 preparation (test of inverting a matrix)

1. Create a new script (Ctrl+N). Save with the name: Exercise6_TP6.m
2. Write a function that contains a square matrix M of order 12 containing the integers from 1 to 144 rows per row.
3. Calculate the determinant of M.
4. Calculate the inverse of the matrix M.
5. Extract from this matrix:
   - A sub-matrix A formed by the coefficients A_ij for: 1<(i,j)<6;
   - A sub-matrix B of order 3 formed by the first even coefficients of M_ij.
   - Modify by the value of zeros all the values of the diagonal of the matrix M.

Save this code in a file named "Exercise6_TP6.m" and run it in MATLAB. It will perform the specified operations on the square matrix M and display the results.

# Lab Sheet 07
# Graphics
# (Management of graphic windows, plot (), fplot ())

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : *Computer Science 3*

## Lab Sheet 07: Graphics (Management of graphic windows, plot (), fplot ())

**TP Objective:**

The goal of TP07 is to introduce you to the management of graphical windows and tools for creating high-quality scientific graphs that effectively present scientific data.

### Part 01: Theoretical part (Management of 2D graphics windows)

If it is beneficial to perform numerical calculations, it is also valuable to have a graphical representation of the results. We will start by plotting the graph of a function. The main command that allows you to plot a graph in MATLAB is the ***plot*** () *or* ***fplot*** () function.

The ***plot***() Command [2.3]:

The ***plot***() function is used to graphically plot a set of 2D points with coordinates:

$$(\boldsymbol{x_i}\ ,\ \boldsymbol{y_i}),\ \boldsymbol{avec\ i = 1, \dots, N}$$

Where:

- $\boldsymbol{x}$: is the vector containing the x-coordinate values $x_i$, and
- $\boldsymbol{y}$: is the vector containing the y-coordinate values $y_i$.

In other words, you provide the **x** and **y** vectors as inputs to the ***plot***() function, and it will create a graph connecting the points defined by these coordinates.

**Syntaxe**

$plot\ (y)$

$plot\ (x, y)$

$plot\ (x, y, 'PropertyName', PropertyValue, \dots)$

*Figure 1.* ***Plot ()*** *Command*

The vectors `x` and `y` must be of the same dimension, but they can be either row vectors or column vectors. By default, the points `(xi, yi)` are connected by straight line segments.

The first thing to do is to look at the syntax of the function (and understand it):

$$plot(x, y)$$

In this syntax, `x` and `y` are the input vectors containing the x and y coordinates of the data points you want to plot. When you execute this command, MATLAB will create a 2D plot with the points connected by straight lines [2.3.1].

For more advanced plotting and customization, you can also provide additional arguments to the $plot()\ function$, such as line styles, markers, colors, and labels.

- The `plot` command takes one, two, or three arguments, or multiples of three.
- If there's only one argument, `plot` plots the values of the vector provided as the argument against its index.
- If there are two arguments, `X` and `Y`, `plot` plots `Y` against `X`.
- The third argument allows you to pass options:
- (Change color, customize the appearance of the graph, etc.)

This flexibility in argument usage enables you to create a wide variety of plots and customize them according to your needs. You can also add labels, titles, and legends to make your plots more informative and presentable. Additionally, MATLAB offers a range of functions and options for creating different types of plots, including line plots, scatter plots, bar plots, and more [2.3.2].

### Example 1 'the graph of the function'

1. Create a new script $Ctrl + N$. Save with the name: Example1_TP7.m
2. Draw the graph of the function f(x)= 5x+2 between -5 and +5:

```
% Vecteur X
x=[-5:0.001:5];
% Vecteur f(x)
f = 5*x+2;
% graphe y=f(x)
plot (x, f);
```

*Figure 2. The $\boldsymbol{plot}()$ Command*

3. Compare with the instruction $plot(x, f)$;
4. Use Hold on to display both functions on the same figure.

## Example 2

1. Create a new script (Ctrl+N). Save with the name: Example2_TP7.m
2. Draw the graph of the function g(x)= 5x+2 between -5 and +5;

```
% Vecteur X
x=[-5:1:5];
% Vecteur g(x)
g = 5*x+2;
% graphe y=g(x)
hold on
plot (x, g);
```

*Figure 3. The plot () Command*

Here's how you can do that:

```
% Create a vector of x values from -5 to 5
x = -5:0.1:5;
% Calculate the corresponding y values for the function f(x) = 5x + 2
y = 5*x + 2;

% Create the plot for f(x) = 5x + 2
plot(x, y, 'r', 'LineWidth', 2); % Red line with a line width of 2
hold on; % Hold the current figure for further plots

% Create the plot for f(x) = x^2
plot(x, x.^2, 'b--', 'LineWidth', 2); % Blue dashed line with a line
width of 2

% Add labels and a legend
xlabel('x');
ylabel('y');
title('Comparison of f(x) = 5x + 2 and f(x) = x^2');
legend('f(x) = 5x + 2', 'f(x) = x^2');

% Show the grid
grid on;

% Release the hold on the figure
hold off;
```

- This code will create a figure that compares the graphs of the functions:

$$f(x) = 5x + 2 \; and \; f(x) = x\text{^}2$$

- The `hold on` command allows you to add multiple plots to the same figure, and the `legend` command adds a legend to differentiate between the two functions. The `grid on` command adds a grid to the plot for better visualization. Finally, `hold off` releases the hold on the figure after all plots have been added [2.3.3].

## Example 3

1. Create a new script (Ctrl+N). Save with the name: Example3_TP7.m
2. Draw the graph of the function y(x)= x*sin(x) between -2π and 2π:

```
17      % Vecteur X
18      x=[-2*pi:0.01:2*pi];
19      % Vecteur y(x)
20      y = x.*sin(x);
21      % graphe y=y(x)
22      plot (x, y);
```

*Figure 4. Function $y(x) = x * sin(x)$ with step 0.01*

3. Use the grid statement for a grid in the figure.

Here's how you can plot the graph of the function y(x) = x*sin(x) between -2π and 2π:

```
% Create a vector of x values from -2*pi to 2*pi
x = -2*pi:0.01:2*pi;

% Calculate the corresponding y values for the function y(x) = x*sin(x)
y = x .* sin(x);

% Create the plot for y(x) = x*sin(x)
plot(x, y, 'b', 'LineWidth', 2); % Blue line with a line width of 2

% Add labels and a title
xlabel('x');
ylabel('y');
title('Graph of y(x) = x*sin(x)');

% Show the grid
grid on;
```

This code will plot the graph of the function y(x) = x*sin(x) using blue lines. The `grid on` command adds a grid to the plot for better visualization. The x values range from -2π to 2π in small increments, and the corresponding y values are calculated using the given function.

## Example 4

1. Create a new script (Ctrl+N). Save with the name: Example4_TP7.m
2. Draw the graph of the function $y1(x) = x * \sin(x)$ between $-2\pi$ *and* $2\pi$:

```
26    % Vecteur X
27    x=[-2*pi:1:2*pi];
28    % Vecteur y(x)
29    y1 = x.*sin(x);
30    % graphe y=y1(x)
31    hold on
32    plot (x, y1);
```

*Figure 5. fonction $y(x) = x * sin(x)$ with step 1*

3. Compare the results of examples 3 and 4, Concluded!

In the previous examples, we have defined a vector $x_i$ of values equally distributed between -5 and 5 or else -2π and 2π (with a step of 0.001 and 0.01 in the first case and of 1 in the second case) and we have calculated the image by the function f,g,y or y1 of these values (vector of images $y_i$), and by the instruction plot we have displayed the coordinate points (x(i), y(i)).

For a curve of any function, you can specify to MATLAB what should be its color, what should be the line style and/or what should be the symbol at each point $A(x_i, y_i)$

For this we give a third input parameter to the plot command which is a string of 3 characters of the form 'cst' with c designating the color of the line, s the symbol of the point and t the type of line. The possibilities are:

| **Couleur de trait** | | **symbole du point** | | **type de trait** | |
|---|---|---|---|---|---|
| ***y*** | *jaune* | *.* | *point* | *-* | *trait plein* |
| ***m*** | *magenta* | *o* | *cercle* | *:* | *pointillé court* |
| ***c*** | *cyan* | *x* | *marque x* | *-* | *pointillé long* |
| ***r*** | *rouge* | *+* | *plus* | *-.* | *pointillé mixte* |
| ***g*** | *vert* | *** | *étoile* | | |
| ***b*** | *bleu* | *s* | *carré* | | |
| ***w*** | *blanc* | *d* | *losange* | | |
| ***k*** | *noir* | *v* | *triangle (bas)* | | |
| ***^*** | *triangle (haut)* | | | | |
| ***<*** | *triangle (gauche)* | | | | |
| ***>*** | *triangle (droit)* | | | | |
| ***p*** | *pentagone* | | | | |
| ***h*** | *hexagone* | | | | |

Default values are 'c=b', 's=.' and 't = -' which corresponds to a solid blue line connecting the points between them.

It is not mandatory to specify each of the three characters. You can just specify one or two. The others will be the defaults. It is possible to plot several curves on the same figure by specifying several arrays x1, y1, x2, y2, ..., as parameters of the plot instruction. If you want the curves to have a different appearance, you will use different color and/or line style options after each pair of x,y vectors.

### Save a figure

The **print** command allows you to save a graphical figure from a graphics window into a file in various image formats. The syntax of the **print** command is as follows:

$$\boldsymbol{print\ -f < num > \ -d < format >< nomfic >}$$

**Where:**

- $< \boldsymbol{num} >$ Refers to the number of the graphics window (figure 1, 2 ...).
- $< \boldsymbol{format} >$ Specifies the format in which the figure will be saved. There are many supported formats, and you can obtain the complete list by typing help print. For example, image formats like JPEG.
- $< \boldsymbol{filename} >$ The name of the file in which the figure will be saved.

This command allows you to create permanent records of your plotted data in various formats for future use or sharing.

## Part 02: Simulation Part (Conditions)

### Exercise 1

1. Create **a new script** (Ctrl +N). Save with the name : **Exercice1_TP7.m**
2. On trace sur l'intervalle $x \in [-5, 5]$ la fonction $\boldsymbol{f(x) = x^2\ cos(x)}$ en trait plein bleu, et la fonction $\boldsymbol{g(x) = x\ cos(x)}$ en trait pointillé rouge.

```
34    % Vecteur X
35    x = [-5:0.01:5];
36    % Vecteur f(x) et g(x)
37    f = x.^2.*cos(x); g = x.*cos(x);
38    % les graphes y=f(x) et y1=g(x)
39    plot(x,f,'b-',x,g,'r:');
```

*Figure 6. Fonction $\boldsymbol{f(x) = x^2\ cos(x)}$*

Here's how you can create the script to plot the given functions on the interval [-5, 5]:

```
% Create a vector of x values
x = linspace(-5, 5, 100);
% Calculate the values of the functions
f = x.^2 .* cos(x);
g = x .* cos(x);
```

```
% Plot the functions
figure;
plot(x, f, 'b-', 'LineWidth', 2); % Blue solid line
hold on;
plot(x, g, 'r--', 'LineWidth', 2); % Red dashed line

% Add labels and title
xlabel('x');
ylabel('y');
title('Plot of f(x) = x^2 cos(x) and g(x) = x cos(x)');
legend('f(x)', 'g(x)');
grid on;

% Save the figure as an image
print -dpng Exercice1_Plot.png

% Display the figure
hold off;
```

This script will generate a plot of the functions `f(x) = x^2 cos(x)` and `g(x) = x cos(x)` on the interval `x ∈ [-5, 5]`, using a solid blue line for the first function and a red dashed line for the second function.

The plot will include labels, a title, and a legend. It will also display a grid. Finally, the script will save the plot as a PNG image named "Exercice1_Plot.png" in the same directory as the script.

## Exercise 2 (the loglog() statement)

1. Create a new script (Ctrl+N). Save with the name: Exercise2_TP7.m
2. If x and y are two vectors of the same dimension, the statement loglog(x,y) displays the vector log(x) against the vector log(y). The loglog() command is used in the same way as the plot() command.
3. What is the slope of the line?

```
41    % Vecteur x
42    x = [1:10:1000];
43    % Vecteur y=f2(x)
44    f2 = x.^3;
45    % les graphes y=f2(x) et log(y)=log(f2(x))
46    plot(x,f2)
47    loglog(x,f2);
```

*Figure 7. The loglog() statement*

The slope of the line in a log-log plot corresponds to the exponent of the power-law relationship between the variables being plotted. In other words, if you have a function of the form:

$$y = C \cdot x^a$$

Where:

- ( $C$ ) is a constant and ( $a$ ) is the exponent, then when you plot this relationship on a log-log scale, the slope of the line will be equal to ( $a$ ).

Let's take an example to illustrate this concept:

```
% Create a vector of x values
x = logspace(-1, 1, 100);

% Calculate the values of y using a power-law relationship: y = x^2
y = x.^2;

% Plot the log-log graph
figure;
loglog(x, y, 'b-', 'LineWidth', 2); % Blue solid line

% Add labels and title
xlabel('log(x)');
ylabel('log(y)');
title('Log-Log Plot: y = x^2');

grid on;

% Save the figure as an image
print -dpng Exercice2_LogLogPlot.png

% Display the figure

% Calculate the slope using a linear fit (polyfit)
coefficients = polyfit(log(x), log(y), 1);
slope = coefficients(1);
disp(['Slope of the line (exponent): ', num2str(slope)]);
```

In this example, the slope of the line should be approximately 2, which matches the exponent of the power-law relationship $y = x^2$.

## Improve the readability of a figure

## Caption a figure

To caption a figure:

1. The xlabel command is used to add a legend text under the abscissa axis. The syntax is: xlabel(' caption ')
2. The ylabel command does the same for the y-axis: ylabel(' legend ')
3. The title command gives a title to the figure. The syntax is: title(' the title^' )
4. The gtext command allows you to place the text at a position chosen on the figure using the mouse. The syntax is gtext(' some text ').

Enhancing the readability of a figure is essential to effectively convey information. Adding labels and legends to the figure can greatly improve its interpretability. Here are the commands to add labels and a legend to a figure in MATLAB:

**1. Adding Axis Labels:**

- Use the `xlabel` function to add a label to the x-axis: `**xlabel('X Label')**`
- Use the `ylabel` function to add a label to the y-axis: `**ylabel('Y Label')**`

Example:

```
xlabel('Time (s)');
ylabel('Amplitude');
```

**2. Adding a Title:**

- Use the `title` function to add a title to the figure: `**title('Figure Title')**`

Example:

```
title('Sinusoidal Waveform');
```

**3. Adding a Legend:**

- Use the `legend` function to add a legend to the figure.
- Specify the legend entries as a cell array of strings: `legend('Legend1', 'Legend2', ...)`

Example:

```
legend('Signal A', 'Signal B');
```

**4. Adding Text Annotations:**

- Use the `text` function to add text annotations at specific locations on the figure.
- Specify the coordinates (x, y) and the text: `text(x, y, 'Annotation')`

Example:

```
text(2, 5, 'Maximum Value');
```

### 5. Interactive Text Placement:

- Use the `**gtext**` function to interactively place text on the figure using the mouse.

Example:

```
gtext('Click to place text');
```

Here's how you can use these commands to enhance the readability of your figures:

```
% Create a sample plot
x = linspace(0, 2*pi, 100);
y1 = sin(x);
y2 = cos(x);
figure;
plot(x, y1, 'b-', 'LineWidth', 2);
hold on;
plot(x, y2, 'r--', 'LineWidth', 2);
xlabel('X Axis');
ylabel('Y Axis');
title('Sine and Cosine Functions');
legend('Sine', 'Cosine');
text(pi/2, 1, 'Maximum Value of Sine');
gtext('Annotation with gtext');
grid on;
% Save the figure
print -dpng EnhancedFigure.png;
```

By using these commands, you can make your figures more informative and easier to understand for your audience.

## Exercise 3 (Improving the legend of a figure)

1. Create a new script (Ctrl +N). Save with the name: Exercise3_TP7.m
2. Use the various previous commands to caption a figure.

```
1 -    P = 5;
2 -    t = [0:.01:2];
3 -    c = 12*exp(-2*t) - 8*exp(-6*t);
4 -    plot(t,c); grid
5 -    xlabel('temps en minutes')
6 -    ylabel('concentation en gramme par litre')
7 -    title(['evolution de la concentration du produit '])
8 -    gtext('concentration maximale')
```

*Figure 8. Improving the legend of a figure*

## Exercise 4 (Drawing the graph of a function; the $fplot$ () command)

The fplot command allows you to draw the graph of a function over a given interval.

Syntax:

$$\boldsymbol{fplot('nomf', [xmin\ ,\ xmax])}$$

From where:

- $\boldsymbol{nomf}$: Either the name of an embedded MATLAB function (cos, sin…), or an expression defining a function of the variable x, or the name of a user function.
- $[\boldsymbol{xmin}\ ,\ \boldsymbol{xmax}]$ : is the interval for which the graph of the function is drawn.

1. Create a new script $Ctrl + N$. Save with the name: Exercise4_TP7.m
2. Let's illustrate the three ways to use the fplot command. We seek the graph of the sine function between -2π and 2π using the instruction:
$$\boldsymbol{fplot('sin', [-2 * pi\ 2 * pi])}$$
3. To draw the graph of the function $\boldsymbol{h(x) = x\ sin(x)}$ between -2π and 2π , we can define the user function $\boldsymbol{h(x)}$ as follows (be careful to read x.*sin(x) and not x*sin(x)):
$$\begin{cases} \boldsymbol{function\ y = h(x)} \\ \quad \boldsymbol{y = x.* sin(x)} \end{cases}$$
4. Then find the graph of the function $\boldsymbol{h(x)}$ by the instruction:
$$\boldsymbol{fplot('h', [-2 * pi\ 2 * pi])}$$
5. The other way to do this is to run the statement:
$$\boldsymbol{fplot('x * sin(x)', [-2 * pi\ 2 * pi])}$$

## Exercise 5 (Drawing the graph of a function; the $fplot$ () command)

1. Create a new script (Ctrl+N). Save with the name: Exercise5_TP7.m
2. Let us illustrate two functions:
$f(x) = \frac{sin(x)}{x}$ sur $[-5;\ 5]$ et $g(x) = \frac{cos(x)}{x}$ on $[-1;\ 1]$ ;
let's use the instruction:
$$\boldsymbol{fplot('[f, g], [-5, 5\ , -1, 1]}$$
3. Label the figure and use curves with different colors.

Here's how you can use the `fplot` command to plot the two given functions and label the figure with curves of different colors:

```
% Create a new script
% Save with the name: Exercise5_TP7.m

% Define the functions
f = @(x) sin(x) ./ x;
g = @(x) cos(x) ./ x;
% Plot the functions
```

```
figure;
fplot(f, [-5, 5], 'b'); % Blue color for f(x)
hold on;
fplot(g, [-1, 1], 'r'); % Red color for g(x)

% Label the figure
xlabel('x');
ylabel('y');
title('Graphs of f(x) and g(x)');
legend('f(x) = sin(x)/x', 'g(x) = cos(x)/x');
grid on;
% Save the figure
print -dpng Exercise5_Figure.png;
```

In this script, we define the functions `$f(x)$` *and* `$g(x)$` using function handles. Then, we use the `fplot` command to plot these functions over their respective intervals. The `'b'` and `'r'` arguments in the `fplot` commands specify the colors blue and red for the curves. We also label the figure, add a legend to distinguish between the two functions, and apply a grid. Finally, the figure is saved as a PNG image.

## Part 03: Experimental part (MATLAB – SIMULINK)

### Exercise 6 preparation ()

1. Create a new script $Ctrl + N$. Save with the name: Exercice7_TP7.m
2. Graph the function $f(x) = -48e^{-18x}\sin(314x - 120) - 48x$ on [-10; 10] using;
3. The fplot statement;
4. The plot statement.
5. Conclude!!

### Exercise 7 preparation (test of inverting a matrix)

1. Create a new script (Ctrl+N). Save with the name: Exercice7_TP7.m
2. Graph the square function on [-1; 1] using the instruction: plot()
3. Graph the square function on [-1; 1] using the instruction: fplot()
4. Compare the two results.

### Preparation exercise 8 (Superimposition of curves in the same figure)

5. Create a new script (Ctrl+N). Save with the name: Exercise8_TP7.m
6. Plot the functions $\boldsymbol{f(x) = exp(x)}$ on [-1; 1]; $g(x) = x$ on [-1; 1]; $h(x) = log(x)$ over [1/e; e] in the same figure.
7. Use the instructions: $fplot()$ *or* $plot()$
8. Use both statements: $hold\ on$ *and* $hold\ off$

# Lab Sheet 08:
# Using toolboxes

Nour Bachir University Center of El-Bayadh
Institute of Sciences
Department of Technology
Specialization: ETT, ELN, TLC, HYD, and GC,
Level: 2nd Year University Common Core (Semester 03)
Lab Sessions : *Computer Science 3*

# Lab Sheet 08: Using toolboxes

**TP Objective:**

The goal of this TP08 is to familiarize you with operations on vectors and matrices and how to use them to solve exercises.

## Part 01: Theoretical Section (Toolbox)

The MATLAB Toolbox is a collection of functions and tools specifically designed for a particular application. Users can utilize these functions to perform specific tasks without the need to write their own code [2.1].

To use a Toolbox, you simply need to install and add it to your MATLAB workspace. Once added, you can access the functions of the Toolbox from the command line or the graphical user interface. Some Toolboxes might require a license to be used, but there are also many free Toolboxes available.

The term "Toolbox" in MATLAB refers to a set of MATLAB functions (also known as "Toolboxes") that provide additional functionality for a specific application or domain of study. These Toolboxes are separate from the core MATLAB system but can be added to the MATLAB environment as needed.

For example, there are Toolboxes available for control systems, image processing, optimization, signal processing, statistics, and many other domains. These Toolboxes provide a comprehensive set of functions and tools to solve problems and perform tasks in the respective field.

Once a Toolbox is installed, its functions can be called just like any other MATLAB function, and its user interface tools (such as dialog boxes or graphical user interfaces) can be used to interact with the Toolbox.

## Control System Toolbox

Control System Toolbox is a package for MATLAB that consists of tools specifically developed for control applications. This toolbox provides data structures to describe common system representations, such as state-space models and transfer functions, along with analysis and design tools for control systems. It also includes simulation tools for systems.

In this guide, you will learn about the basic commands of the Control System Toolbox. By the end of this exercise, you should be able to understand and use the Control System Toolbox to create and analyze linear systems. It's recommended to make extensive use of MATLAB's help command. It's also advised to create a script file (e.g., EX01.m) where you write your commands. By running a script file instead of typing commands directly at the MATLAB prompt, it's easier to correct errors, and your work will also be saved for future use [2.2].

### Example 1

The system you will be working with is:

$$\dot{x} = Ax + Bu = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$

$$y = Cx = [1 \quad 0]$$

### Creation and conversion of systems

Control System Toolbox supports several system representations of time-invariant linear systems. In this exercise, we will use two of the most common representations; state space models and transfer functions.

Define the system matrices A, B, C and D given above. (What is the value of D in the model?) Create a system state space description using ss and name it sys_ss. Learn how to use ss using the help function (help ss). At this point you should have a description of the system state space.

Now let's create an equivalent transfer function model of the system above. This could, as you know, be done using the formula:

$$G(s) = C(sI - A)^{-1} B + D$$

However, Matlab can also be used for the task. Use the ***tf*** command to convert the state space model to a transfer function and name it $sys_{tf}$. Note that $tf$ can be used for creation of transfer functions as well as conversion.

What is the syntax in the two cases respectively?

To create and convert systems in the Control System Toolbox, you can follow these steps:

1. Define the system matrices A, B, C, and D as provided in the given system description.
2. Create a state-space representation of the system using the `ss` function:

```
A = [0 1; -1 -1];
B = [0; 1];
C = [1 0; 1 1];
D = 0; % D matrix is 0 in this case
sys_ss = ss(A, B, C, D);
```

3. To convert the state-space representation to a transfer function, you can use the `tf` function:

```
sys_tf = tf(sys_ss);
```

The syntax for creating a state-space representation using the `ss` function is:

```
sys_ss = ss(A, B, C, D);
```

And the syntax for converting a state-space representation to a transfer function using the `tf` function is:

```
sys_tf = tf(sys_ss);
```

- In these commands, `A`, `B`, `C`, and `D` are the system matrices defined earlier.

- Please note that the provided D matrix is 0, which indicates that there is no direct feedthrough from the input to the output in this system.

## Example 2 (Stability analysis)

The stability of a linear system is determined by the location of its poles in the complex plane [2.3].

- What is the stability condition?

Use the ssdata and tfdata commands to extract the necessary data from the models, and eig and roots to determine system stability. Verify that the roots of the characteristic polynomial of the transfer function are the same as the eigenvalues of the system matrix.

- What are the eigenvalues/poles?

Is the system stable?

You can also use the command pole, or for a graphical view, pzmap.

The stability of a linear system is determined by the location of its poles in the complex plane. For a continuous-time linear system, the system is stable if and only if all of its poles have negative real parts.

To analyze the stability of the system using the state-space representation and transfer function representation, you can follow these steps:

1. Extract the system matrices from the state-space representation:
```
[A, B, C, D] = ssdata(sys_ss);
```

2. Extract the system poles using the `eig` function:
```
eigenvalues = eig(A);
```

3. Extract the system poles using the `pole` function:
```
poles = pole(sys_tf);
```

4. Extract the system poles using the `pzmap` function:
```
pzmap(sys_tf);
```

The `eig` function returns the eigenvalues of the system matrix `A`, which are also the poles of the system. The `pole` function returns the poles of the transfer function, and the `pzmap` function provides a graphical representation of the poles in the complex plane.

To determine if the system is stable, you need to check if all the eigenvalues (poles) have negative real parts. If all eigenvalues have negative real parts, the system is stable. If any eigenvalue has a positive real part, the system is unstable.

Please execute these commands in MATLAB to analyze the stability of the given system [2.3.1].

## Example 3 (Time Domain Analysis)

1. Use the step command to plot the step response of the system.
2. Connect the characteristics of the step response to the location of the poles.
3. If you have time, use initial and lsim to study the system response.

All blocks needed for this model:

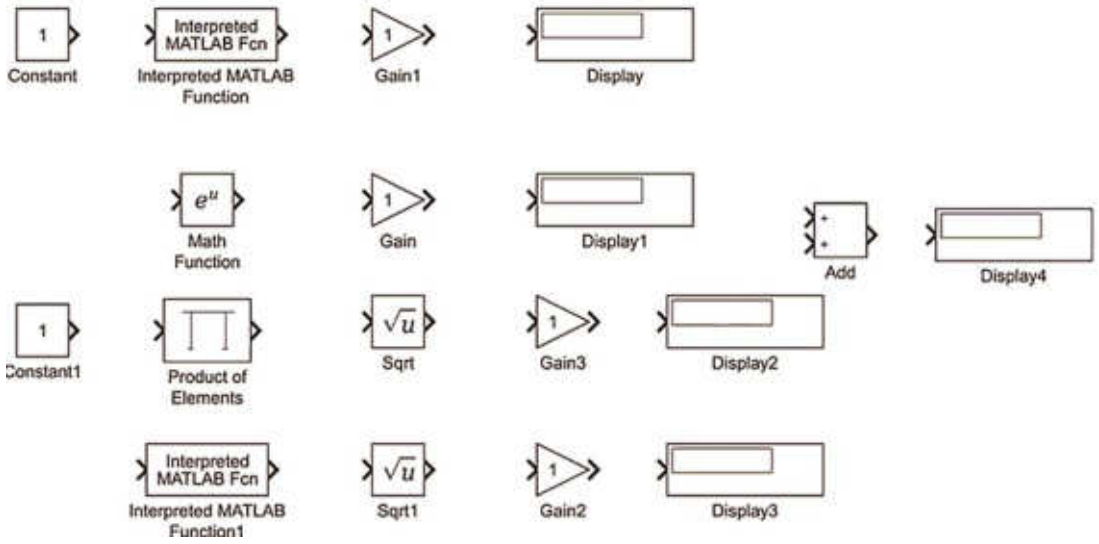



*Figure 1. Blocs SIMULINK*

To analyze the time-domain behavior of the system and understand its response, you can follow these steps [2.3.2]:

1. Use the `step` function to plot the step response of the system:

```
step(sys_tf);
```

2. Use the `initial` function to study the initial response of the system:

```
initial(sys_tf);
```

3. Use the `lsim` function to simulate the response of the system to a given input:

```
t = 0:0.01:10; % Time vector
u = sin(t);    % Input signal
lsim(sys_ss, u, t);
```

In the step response plot, observe the rise time, settling time, and overshoot. Connect these characteristics to the location of the poles in the complex plane. Poles with larger real parts typically result in faster decay and faster response.

For the `initial` and `lsim` functions, you can specify different input signals to observe the system's behavior under various conditions.

Execute these commands in MATLAB to analyze the time-domain behavior of the given system.

**Some Useful MATLAB Commands:**

| | |
|---|---|
| plot | Linear plot. |
| subplot | Breaks the Figure window into small axes. |
| axis | Control axis and scaling appearance. |
| hold | Hold current graph. |
| grid | Grid lines. |
| title | Graph title. |
| xlabel, ylabel | Axes labels. |
| tf | Create a transfer function model. |
| tfdata | Extract numerator and denominator. |
| ss | Create a state-space model. |
| ssdata | Extract state-space matrices. |
| zpk | Create a zero/pole/gain/model. |
| step | Step response. |
| initial | Response of state-space model with given initial state. |
| pole | System poles. |
| zero | System zeros. |
| roots | Find polynomial roots. |
| pzmap | Pole-zero map. |
| eig | Compute eigenvalues and eigenvectors. |
| bode | Draw a bode frequency response. |
| lsim | Simulate time response of LTI models to arbitrary inputs. |
| simulink | Open the simulink browser. |
| sim | Use a Simulink model from a Matlab script |
| simset | Set options for the sim command |
| ictools | Interactive tools for control |
| pplane6 | Phase plane analysis, download from our webside |
| linmod | Obtain the state-space linear model of the system of ordinary differential equations described in a simulink model, note that the model must contain a simulink block *out* from sinks and a simulink block *in* from sources |

## Introduction to Simulink

Simulink is a powerful simulation environment in MATLAB that allows you to model, simulate, and analyze dynamic systems using block diagrams. It provides a graphical way to represent and simulate complex systems, making it easier to understand and design control, signal processing, and other dynamic systems.

Simulink offers a range of predefined blocks that represent various elements of a system, such as sources, sinks, transfer functions, integrators, summing points, and more. You can drag and drop these blocks into your model and connect them to create a visual representation of the system's behavior.

Simulink models can be created graphically by connecting these blocks and specifying their parameters. This approach is intuitive and useful for representing systems that involve multiple components and interactions.

Alternatively, you can also create Simulink models using MATLAB code. This allows you to define the system's behavior using equations, making it suitable for systems that can be described by mathematical expressions. Key features of Simulink:

1. Block Diagram Modeling: Create models by connecting predefined blocks that represent various system components.
2. Simulation: Simulate the behavior of your system over time and observe its response.
3. Analysis: Analyze simulation results and system behavior under different conditions.
4. Customization: Customize block parameters, simulation settings, and visualization options.
5. Hierarchical Modeling: Organize complex models into subsystems and hierarchies for better organization and modularity.
6. Real-Time Simulation: Simulate real-time systems and hardware-in-the-loop (HIL) testing.

In Simulink, you can simulate continuous-time, discrete-time, and hybrid systems. It's widely used in various fields including control systems, signal processing, communication systems, automotive systems, and more.

Overall, Simulink provides a powerful and versatile platform for modeling and simulating dynamic systems, helping engineers and researchers design and analyze complex systems effectively.

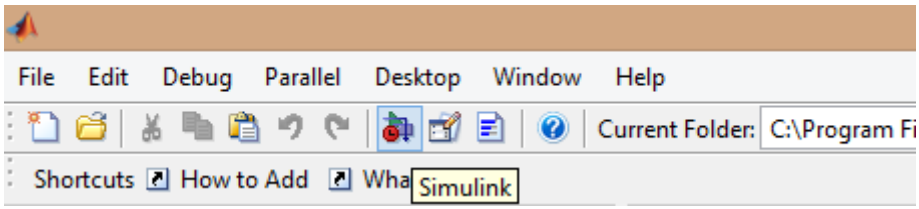



*Figure 2. SIMULINK icon*

To understand how models are described and simulated using functional diagrams, it's best to run small examples on a computer. The rest of Section 2 presents a few examples. If you are familiar with Simulink, you can skip directly to Section 3.

### How to Start Simulink

To start Simulink, simply enter the command Simulink in the MATLAB Command Window. This will open a new window with blocks and tools for creating and simulating models, as shown in the figure below.

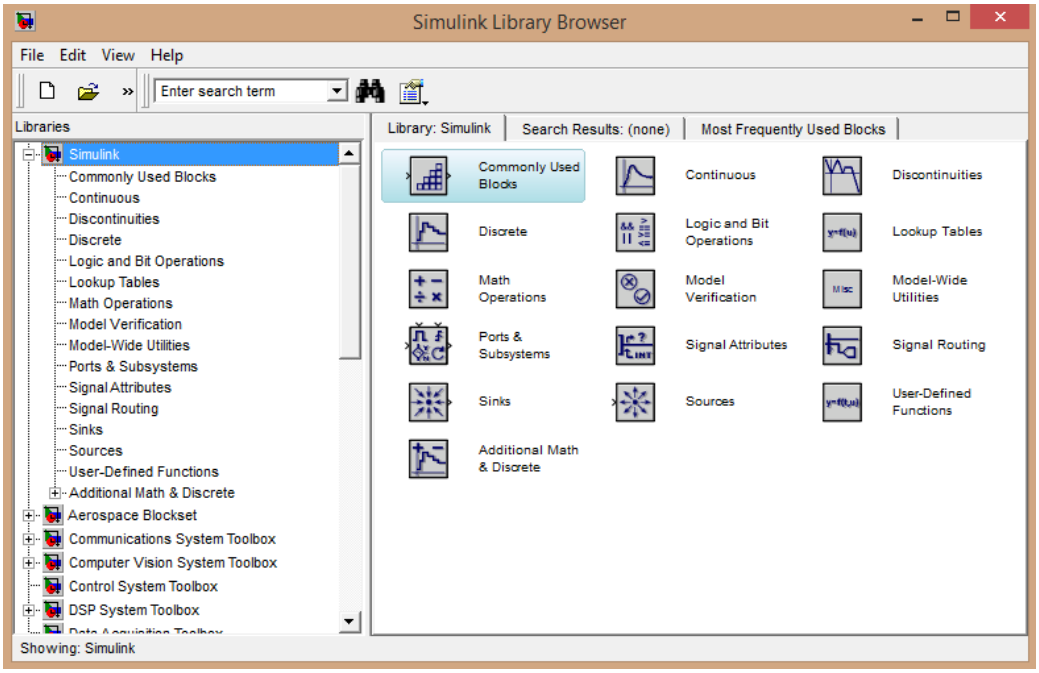



*Figure 3. Simulink Library Browser*

### A simple system

We can help guide you through creating a simple system in Simulink using the steps you provided:

1. Click on "File" in the Simulink window and choose "New" -> "Model."
2. Click on the "Continuous" block and drag a "Transfer Fcn" block into the new window named "untitled."
3. Repeat the process with "Sources" -> "Step" and "Sinks" -> "Scope" blocks.

4. Use the mouse (left-click) to draw arrows and connect the ports on the blocks.
5. Your functional diagram should now look similar to the figure below.

However, since I can't display images directly, you might want to refer to the Simulink user guide or interface itself to see the visual representation of the system you've created. If you have any questions about specific steps or concepts, feel free to ask!

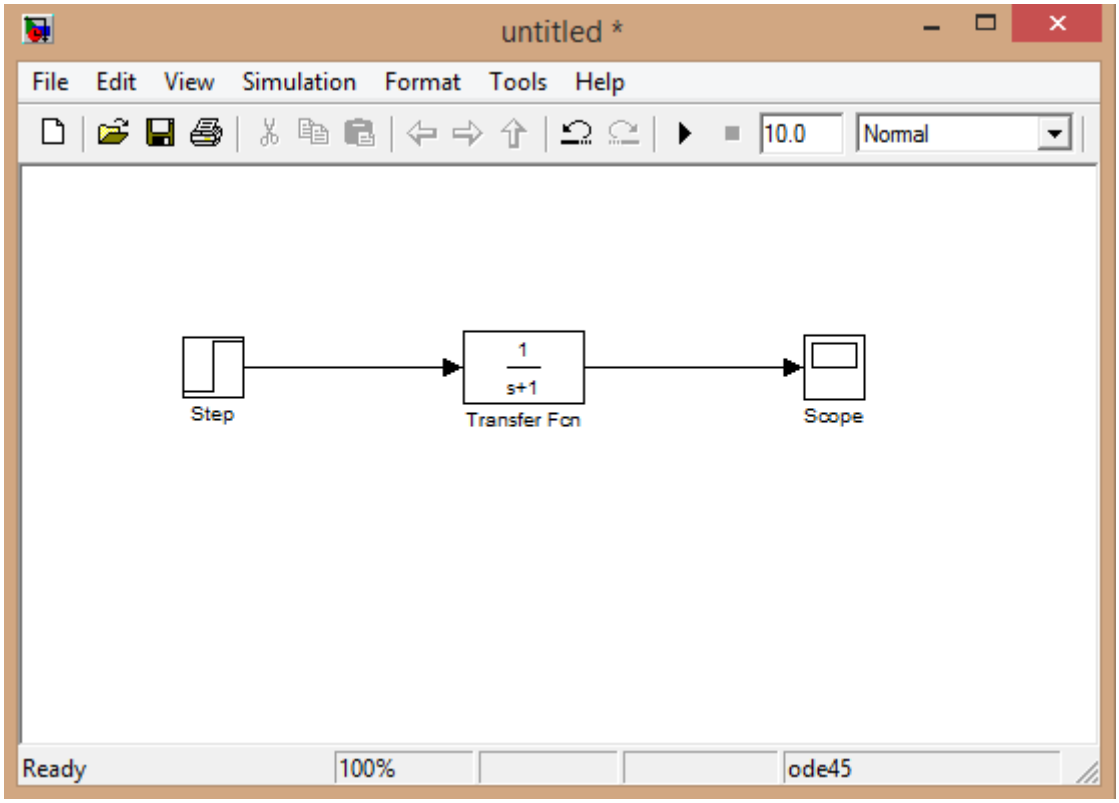



*Figure 4. transfer function order 1 with step in model Simulink*

Here are the steps to follow based on your instructions:

1. Choose "Simulation" -> "Parameters" in the window named "untitled."
2. Set the Stop Time to 5 in the simulation parameters.
3. Open the "Scope" window by double-clicking on it.
4. Set the horizontal range to 6 in the Scope window.
5. Start the simulation by selecting "Simulation" -> "Start" or by pressing Ctrl + T in the "untitled" window.

These steps will help you configure the simulation parameters and visualize the output using the Scope block. If you encounter any issues or have questions about specific steps, feel free to ask for further assistance!

## Example 4 (changing a system)

To change the system to:

$$\frac{7}{s^2 + 0.5s + 2}$$

The steps to change the system to the new transfer function $\frac{7}{s^2+0.5s+2}$ in Simulink:

1. Double-click on the "**Transfert Fcn**" block in the Simulink model.

In the "Transfer Function" block dialog box that appears, you'll find options to change the Numerator and Denominator coefficients of the transfer function. Change these coefficients to match the new transfer function $\frac{7}{s^2+0.5s+2}$.

2. Once you've entered the new coefficients, click "OK" to close the dialog box.

After making these changes, your Simulink model will now represent the new transfer function $\frac{7}{s^2+0.5s+2}$. You can then proceed with the steps mentioned in the previous example to set simulation parameters, adjust the scope, and start the simulation to visualize the response of the new system.

- Vous devriez maintenant avoir un schéma fonctionnel comme dans la figure ci-après.

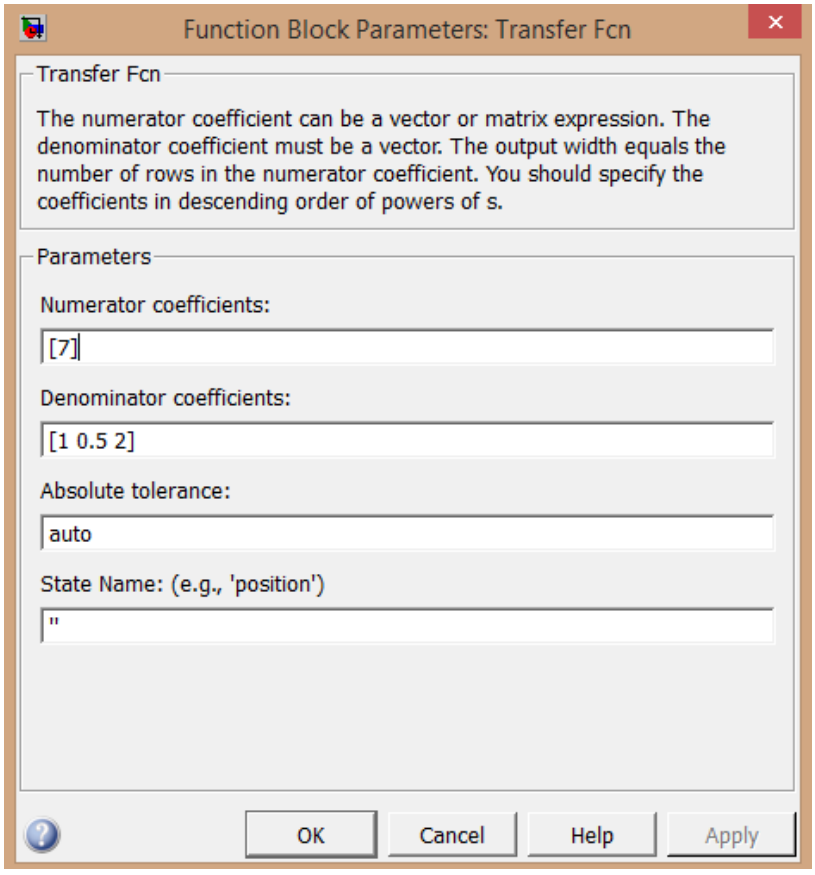



*Figure 5. Function block parameters: transfert fcn*

The system becomes:

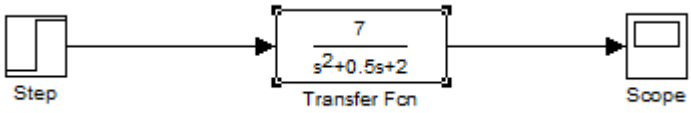



*Figure 6. Model Simulink*

To change the input signal, follow these steps in Simulink:

1. Delete the "Step Fcn" block by clicking on it and using Edit -> Cut (or Ctrl-x).
2. Add a "Sources -> Signal Gen" block to the model.
3. Double-click on the "Signal Gen" block to open its dialog box. Here you can select the type of signal (sine, square, etc.), set the amplitude and frequency, and adjust other parameters as needed.
4. Modify the simulation parameters: Go to Simulation -> Configuration Parameters, and change the "Stop Time" to a large value like 99999 to simulate an "infinite" simulation. You can stop the simulation manually using Simulation -> Stop (or Ctrl-t).

Can the amplitude of the input signal be changed during the simulation? Also, try changing the Transfer Fcn block's parameters during the simulation.

**Using MATLAB Variables in Simulink Blocks:**

Any variables defined in the MATLAB workspace can be used in Simulink. For example, if you define variables "num" and "den" in the MATLAB workspace, you can use them in Simulink to define the transfer function numerator and denominator.

1. Define the variables in the MATLAB workspace:

```
num = [7 6];
den = [1 2 3 4];
```

2. In the Transfer Fcn block dialog box, replace Transfer Fcn -> Numerator with "num" and Transfer Fcn -> Denominator with "den".

**Saving Simulation Results to MATLAB Variables:**

To save the input and output signals, add two copies of the "Sinks -> To Workspace" block to the model. Make sure the "Save format" option is set to "Array" in the block's dialog box. Connect them to the input and output of the Transfer Fcn block.

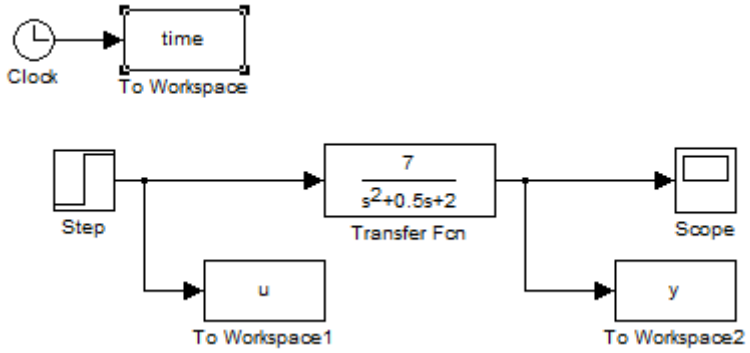



*Figure 7. Model Simulink with workspace function*

Additionally, add a "Sources -> Clock" block and connect it to a "Sinks -> To Workspace" block. Name the variables as "u" (for input), "y" (for output), and "t" (for time).

By setting up these blocks, you can save the simulation results (input, output, and time) into MATLAB workspace variables "u," "y," and "t" respectively.

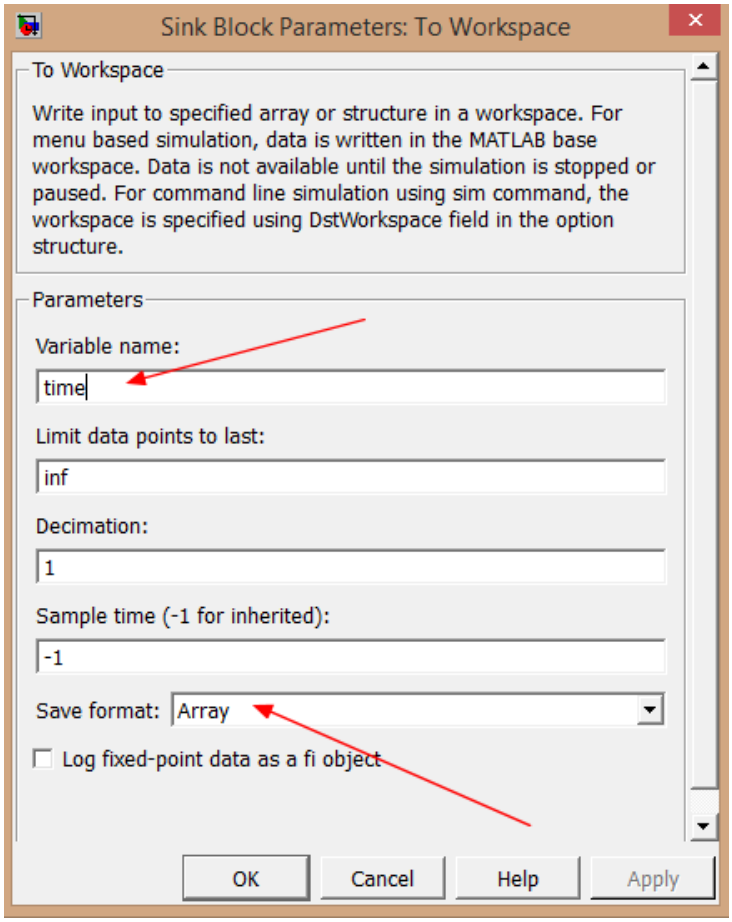



*Figure 8. Sink block parameters: To Workspace*

To use the simulation results in MATLAB calculations, you can follow these steps:

1. Set up the Simulink model as described earlier, with appropriate input and output signals.
2. Run the simulation for a sufficiently long time until the output becomes stationary.

Once the simulation is completed and the results are saved in the MATLAB workspace variables "u," "y," and "t," you can perform calculations using these variables.

To calculate the maximum value of "y" when the system has stabilized, you can use the following steps:

1. Calculate the index "n" corresponding to the length of the "y" signal:

```
n = length(y);
```

2. Calculate the maximum value of "y" considering only the second half of the signal (when the system has stabilized):

```
max_y_stabilized = max(y(n/2:n));
```

In this code, "n/2" corresponds to the index where the second half of the signal begins (after stabilization). "max(y(n/2:n))" calculates the maximum value of "y" within that range.

By using these calculations, you can determine the maximum value of the output signal "y" after the system has stabilized.

**Using Simulink Models in Matlab Scripts**

Frequently, working with MATLAB scripts (**.m** files) proves advantageous as it allows for the preservation of a sequence of commands. It is feasible to incorporate Simulink models into MATLAB scripts using the **sim** command. By utilizing the **simset** command, options for the **sim** command can be specified.

To employ the model from the preceding example, it is recommended to first save the model and name it "mymodel.mdl". Following this, proceed to create a MATLAB script named "mysim.m" and input the following commands:

```
% Load the model
load_system('mymodel.mdl');
```

```
% Set simulation parameters
options = simset('SrcWorkspace', 'current');

% Simulate the model
sim('mymodel.mdl', [], options);

% Extract and analyze simulation results
time = simout.Time;
output = simout.Data;

% Plot the simulation results
plot(time, output);
xlabel('Time');
ylabel('Output');
title('Simulation Results');

% Unload the model
close_system('mymodel.mdl', 0);
```

In this manner, the **sim** function is employed to simulate the model, and the resultant data is extracted and visualized through plotting. Furthermore, the script ensures the appropriate handling of the model by loading and unloading it as necessary.

```
tfinal = 300;
options = simset('reltol',1e-5,'refine',10,'solver','ode45');
sim('mymodel',tfinal,options);
%plot results
figure(1)
clf
subplot(211)
plot(t,u);
ylabel('u')
subplot(212)
plot(t,y)
ylabel('y')
```

When you execute the script, you should observe a plot displaying both the input and output of the transfer function. For further insights into the usage of the **simset** and **sim** commands, you can utilize the **help** command to access the documentation.

For saving systems within Simulink, you can navigate to "File" and then choose "Save As" or "File" -> "Save". This enables you to save the current model configuration and its associated parameters, which can then be loaded and manipulated in subsequent sessions.

## Part 02: Simulation part (A flow system)

Consider a simple tank as in the basic control course

$$\dot{h} = \frac{1}{A}(u - q)$$

$$q = a\sqrt{2gh}$$

This can be implemented in Simulink as shown in the figure below. The function $f(u)$ has a value of $a * sqrt(2g) * sqrt(u[1])$. The Sum block has received two inputs with different signs by assigning the string "-+|" to Sum->Sign List. The input and output blocks are located under Sources and Sinks, respectively.

These blocks inform Simulink about what should be considered as inputs and outputs to this subsystem. The block titles can be modified by clicking on them. Select the entire system by holding down the left mouse button and drawing a square around it.
Then, choose Edit->Create Subsystem. The result is that the system is represented by a single block. Use Edit->Copy to create the double-tank system shown below.

## Exercise 1

1. Use the ***linmod command*** to find a linearized model of the double tank around

$$h_1^0 = h_2^0 = 0.1$$

2. Use settings:

$$A_1 = A_2 = 2.7\ 10^{-3}, a_1 = a_2 = 7 \times 10^{-6}, g = 9.8$$

Also notice in the figure the Simulink block from sources and the Simulink block from sinks, which are needed for the ***linmod command***.

```
>> A=2.7e-3; a=7e-6; g=9.8;
```

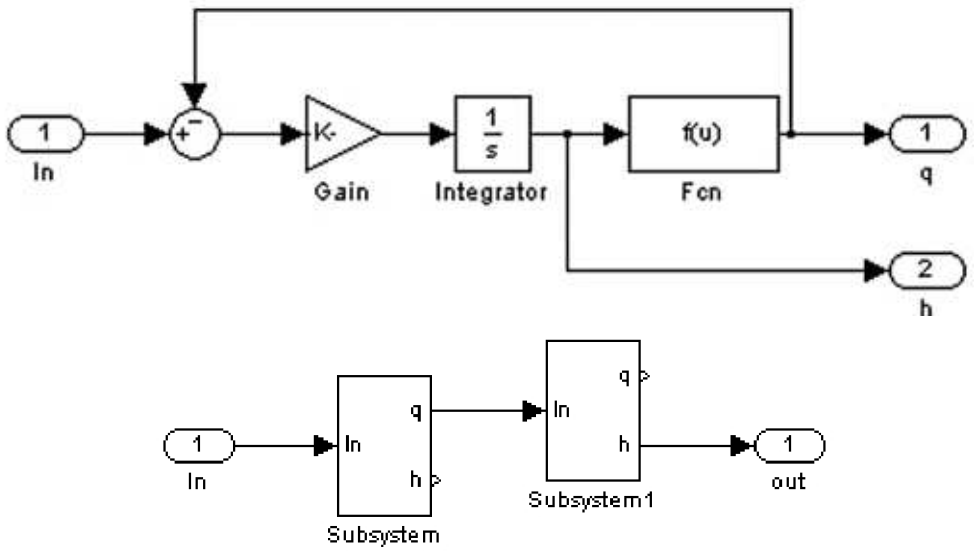



*Figure 9. Model flow system*

```
>> x0 = [0.1000 0.1000]';
>> u0 = a*sqrt(q*g*x0(1));
>> [aa, bb,cc,dd]=linmod('flow',x0,u0);
```

## Part 03: Experimental part (MATLAB – SIMULINK)

### Exercise 6

To build a Simulink model to compute the values of the cosine function: $g(t) = cos(\omega t)$ for $(t = 0)$ $to$ $(3)$ $with$ $(3000)$ incremental steps and different values of $\omega = [\pi, 2\pi, 3\pi, 5\pi, 7\pi]$, you can follow these steps:

**Open Simulink:**

- Open MATLAB and type `simulink` in the command window to open the Simulink environment.

**Create the Model:**

- Drag and drop a "Sine Wave" block from the Simulink library onto the canvas.
- Double-click on the block to open its properties.
- Set the "Frequency" parameter to `omega` (which will be specified later).
- Set the "Amplitude" parameter to `1` (since you want to compute \(\cos(\omega t)\)).
- Connect the output of the "Sine Wave" block to a "Scope" block to visualize the output.

**Set Up MATLAB Script:**

- Create a MATLAB script (e.g., `simulate_cos.m`) to set up the simulation.
- Define the values of \(\omega\) as an array: `omega_values = [pi, 2*pi, 3*pi, 5*pi, 7*pi]`.
- Use a loop (e.g., `for` loop) to iterate over the values of \(\omega\):
- Inside the loop, set the value of \(\omega\) in the "Frequency" parameter of the "Sine Wave" block.
- Simulate the model using the `sim` function with the desired time range (e.g., `0:0.001:3`).
- Use the `get` function to extract the simulation data from the "Scope" block.
- Plot the output using the `plot` function.

Here's a simplified version of the MATLAB script `simulate_cos.m`:

```
% Define omega values
omega_values = [pi, 2*pi, 3*pi, 5*pi, 7*pi];

% Loop over omega values
for i = 1:length(omega_values)
    omega = omega_values(i);

    % Load the Simulink model
    open_system('your_simulink_model_name');

    % Set the omega parameter of the Sine Wave block
    set_param('your_simulink_model_name/Sine Wave', 'Frequency',
num2str(omega));

    % Simulate the model
    sim_time = 0:0.001:3;
    sim_output = sim('your_simulink_model_name', 'StopTime', 'sim_time');

    % Extract simulation data from the Scope block
    sim_data = sim_output.ScopeData.signals.values;

    % Plot the simulation results
    figure;
    plot(sim_time, sim_data);
    title(['Cosine Function for \omega = ', num2str(omega)]);
    xlabel('Time');
    ylabel('Amplitude');

    % Close the Simulink model
    close_system('your_simulink_model_name', 0);
end
```

Make sure to replace `"your_simulink_model_name"` with the actual name of your Simulink model. This script will simulate the model for each value of \(\omega\) and plot the results in separate figures.

Note: The script assumes that you have a Simulink model with a "Sine Wave" block and a "Scope" block as described earlier.

## Exercise 7

To create a Simulink model to simulate a skydiver's acceleration using the given formula:

$$a = g(1 - \frac{v^2}{3600})$$

Where: $g = 9.81\, m/s^2$ \, \text{m/s}^2 $g$=9.81m/s2,

**You can follow these steps:**

1. **Open Simulink:**

   Open MATLAB and type **simulink** in the command window to open the Simulink environment.

2. **Create the Model:**
   - Drag and drop a "Constant" block from the Simulink library onto the canvas.
   - Double-click on the "Constant" block to open its properties.
   - Set the "Value" parameter to the value of g = 9.81$g$=9.81.
   - Drag and drop a "Product" block and a "Sum" block onto the canvas.
   - Connect the "Constant" block to the "Product" block and the output of the "Product" block to the "Sum" block.
   - Drag and drop two "Gain" blocks and connect the "Product" block's input to one of the "Gain" blocks.
   - Set the gain of the "Gain" block to -1−1.
   - Drag and drop a "Sum" block and connect the outputs of the two "Gain" blocks to the inputs of the "Sum" block.
   - Connect the output of the "Sum" block to the "Product" block's second input.
   - Connect the output of the "Sum" block to the "Scope" block to visualize the output.
   - Double-click on the "Gain" blocks to set their gains as needed.
   - Double-click on the "Scope" block to configure its settings.

3. **Set Up MATLAB Script:**
   - Create a MATLAB script (e.g., **simulate_skydiver.m**) to set up the simulation.
   - Define the time range for simulation (e.g., **0:0.1:100**).
   - Use a loop (e.g., **for** loop) to iterate over different initial velocities:
     - Inside the loop, set the initial velocity value.
     - Simulate the model using the **sim** function with the desired time range and initial velocity.

- Use the **get** function to extract the simulation data from the "Scope" block.
- Plot the acceleration vs. time for each initial velocity.

Here's a simplified version of the MATLAB script **simulate_skydiver.m**:

```
% Define parameters
g = 9.81; % m/s^2
time_range = 0:0.1:100;

% Loop over initial velocities
initial_velocities = [0, 10, 20, 30]; % m/s
for i = 1:length(initial_velocities)
    initial_velocity = initial_velocities(i);
    % Load the Simulink model
    open_system('your_simulink_model_name');
    % Set the initial velocity parameter
    set_param('your_simulink_model_name/Gain', 'Gain',
num2str(initial_velocity));
    % Simulate the model
    sim_output = sim('your_simulink_model_name', 'StopTime',
'time_range');
    % Extract simulation data from the Scope block
    sim_data = sim_output.ScopeData.signals.values;

    % Plot the simulation results
    figure;
    plot(time_range, sim_data);
    title(['Skydiver Acceleration for Initial Velocity = ',
num2str(initial_velocity)]);
    xlabel('Time');
    ylabel('Acceleration');

    % Close the Simulink model
    close_system('your_simulink_model_name', 0);
end
```

Make sure to replace **'your_simulink_model_name'** with the actual name of your Simulink model. This script will simulate the model for each initial velocity and plot the acceleration vs. time for each case.

Note: The script assumes that you have a Simulink model with the components as described earlier.

## Exercise 8

The introduction of transfer functions is done in several polynomial forms, ZPK form (zeros, poles, gain), state form. Let's take a few types of second and third order servo systems to practice declaring these systems.

$$f_1(s) = \frac{s+5}{s^3 + 8s^2 + 17s + 10}$$

$$f_2(s) = \frac{s+10}{2s^2 + 3s + 1}$$

$$f_3(s) = 4.25\,\frac{(s-2)(s+1.2)}{(s+0.3)(s+1)(s+5)}$$

1. Consult the online help on the Tf, ZPK, TF2SS, TF2ZP commands
2. Create a command file tp3ex1.m for the MATLAB commands of exercise N°1
3. Introduce **f1(s)** and **f2(s)** in MATLAB
4. Give the poles of these two systems
5. Introduce the **f**ଷ(**s**) system using the ZPK command
6. Go from the transfer function representation of **f1(s)** and **f2(s)** to the ZPK form known by the Evans form

   **H(s)** = **K** (**s-Z(1)**)(**s-Z(2)**)...(**s-Z(n)**) (**s-P(1)**)(**s-P(2)**). ..(**s-P(n)**)).
7. Going from the transfer function representation of f୨(s) and f୥(s) to the state form

   Ẋ = AX + Bu, y = CX + Du.
8. Go from the ZPK representation of fଷ(s) to the form Ẋ = AX + Bu, y = CX + Du.
9. Save the workspace for this exercise in tp3ex1.mat

## Exercise 9 (Building Block Diagrams)

Let the functional diagram given by the figure below:

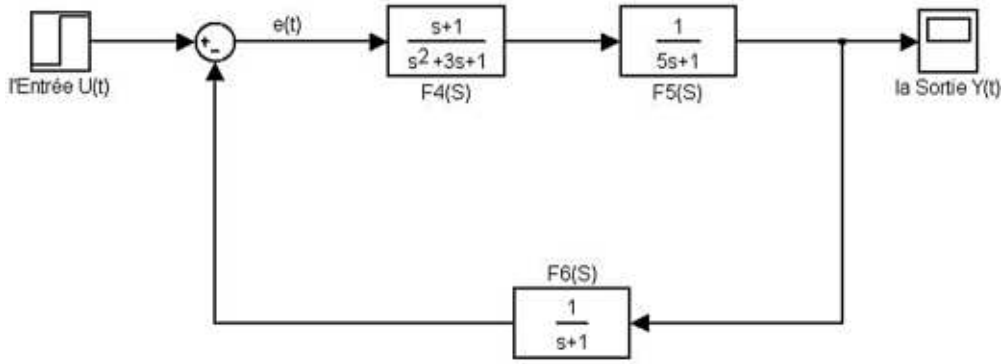



*Figure 10. Building Block Diagrams*

1. Consult the online help on the Series, Parallel, feedback commands
2. Create a command file tp3ex2.m for the MATLAB commands of exercise N°2
3. Give the open loop transfer function
4. Give the closed loop transfer function (Y(s)/U(s))
5. Check the result analytically.
6. Calculate the poles of the closed-loop transfer function
7. Go from this transfer function to the different forms (ZPK, SS) of the system in BF
8. Save the workspace for this exercise in tp3ex2.mat

# Bibliography

# Bibliography

Course / Lab Sheet : Computer science 3

Depending on the availability of documentation at the establishment level, websites...etc.


**[1] SCILAB :**

**[1.1]** **Informatique: Programmation et simulation en Scilab 2014** - Auteurs : Arnaud Bégyn, Jean-Pierre Grenier, Hervé Gras.

**[1.2]** **Scilab : De la théorie à la pratique - I. Les fondamentaux**. Livre de Philippe Roux 2013.

**[2] MATLAB :**

**[2.1] French References:**

**- Introduction à MATLAB et SIMULINK,** Un guide pour les élèves de l'École Nationale Supérieure d'Ingénieurs Electriciens de Grenoble, Paolino Tona.

**- C++ pour les nuls,** Stephen Randy Davis**.**

**[2.2] Arabic References :**

**الماتلاب للمهندسين،** المهندس عدنان شاهين

**[2.3] English References :**

- **Essential MATLAB for Engineers and Scientists**, Seventh Edition, Brian D. Hahn & Daniel T. Valentine.
- **Numerical Analysis Using MATLAB and Excel**, Steven T. Karris.
- **Matlab for Dummies**, Jim Sizemore, John Paul Mueller.

# Fin

Cours / TP : Informatique 3