الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université M'Hamed BOUGARA
Boumerdès

Institut de Génie Electrique et Electronique

وزارة التعليم العالي و البحث العلمي

جامعة امحمد بوقرة
بومرداس

معهد الهندسة الكهربائية و الالكترونيك

Département : Power and System

Polycopié Pédagogique

TITRE DU POLYCOPIE

Introduction to Scientific Computing with Matlab

Cours destiné aux étudiants de l'automatique

Rédigé par

Dr. BENZAOUI Messaouda

Année Universitaire : 2024/2025

Preamble

This course handout of the 'Introduction to scientific computing with Matlab' module is intended to first year Master's students of Control option, from the Institute of Electrical and Electronic Engineering, University of M'hamed Bougara.

Introduction to scientific computing with Matlab involves an introduction to the capabilities of MATLAB's calculation. This course handout was written to provide students with a basic overview of commands, which are useful in science and engineering.

This handout is broken up into eight chapters. That includes using the most recent versions of Matlab for scientific calculations. An introduction is included in the first chapter. It enables students to begin doing calculations globally in Matlab. Matrix computation and linear algebra are covered in the second chapter. Matlab is presented as a tool for computation in this chapter, and several of its primary commands that are helpful in linear algebra and numerical calculations for linear algebra are revealed. Such as solving linear equations and determining the eigenvalues of a square matrix.

The third chapter covers the graphing face: sketching data. Thus, plotting package in Matlab is discussed. This chapter shows how to plot curves in two and three dimensions; how to plot surfaces with simple examples to make sure of understanding the basic commands discussed.

The fourth chapter then, a brief introduction of Matlab as a programming language. This focuses on writing the own MATLAB commands, called m-files function (similar to functions in C and to FORTRAN). Using these functions, student can write a complicated sequence of statements.

The graphical user interface is covered in the fifth chapter. The student can see how to "program" with a "figure window" input/output user interface.

In the sixth chapter it is discussed how to deal with polynomial calculations such as evaluating polynomials, differentiating polynomials, and finding their zeroes. Polynomials are later used to interpolate data. The last two chapters each deal with a functionality toolboxes. Additionally, laboratories reinforce all of the material considering throughout this handout so that students may use the techniques presented.

List of Figures

Figure 1.1 Matlab desktop components 5

Figure 1.2 Matlab up icon 5

Figure 1.3 example1: introducing scalar variable 6

Figure 1.4 Variable poisoning in work space 6

Figure 1.5 Example result of the commands who and whos 7

Figure 1.6 Example result of the command clc 7

Figure 1.7 Example result of the command clear 8

Figure 1.8 Example result of the command clear applied to one variable 8

Figure 1.9 Example result of the command help 9

Figure 1.10 Example result of the command lookfor 9

Figure 1.11 Example result of the command doc 10

Figure 1.12 Example result of the command fix, round, floor, and ceil 10

Figure 2.1 Variables in short and long format 13

Figure 2.2. Different Polar presentations 14

Figure 2.3. Defining a row vector in Matlab 15

Figure 2.4. Defining a column vector in Matlab 15

Figure 2.5 Operations rules with vector in Matlab 16

Figure 2.6 Example of particular functions: Max/min, Sum/mean 16

Figure 2.7 Example of particular functions: linspace/logspace 17

Figure 2.8 Defining matrix 17

Figure 2.9 Example of indexing operations 18
Figure 2.10 Example of defining sub-matrix 18
Figure 2.11 Example of Three dimensional array 19
Figure 2.12 Example of matrix operation element by element 19
Figure 2.13 Example of particular functions: transpose and size 20
Figure 2.14 Example of particular functions: norm/rank 20
Figure 2.15 Example of particular functions: diag/det/eig 21
Figure 2.16 Example of particular functions: triu/tril, repmat 21
Figure 2.17 Example of particular matrices: eye/zeros, ones, rand. 22
Figure 2.18 Example of linear equation with Matlab. 24
Figure 3.1 Example in using the “plot” function. 26
Figure 3.2 Example in using the “stem” function. 27
Figure 3.3 Example in using the “stairs” function. 27
Figure 3.4 Use of two plots in the same figure. 28
Figure 3.5 Use of two plots in the same figure (second manner) 29
Figure 3.6 Example of using annotation in plot 30
Figure 3.7 Example of using annotation for different plots in same figure 30
Figure 3.8 Adding Title, legend, xlabel/ylabel to the figure 31
Figure 3.9 Example of using “gca” 32
Figure 3.10. Example of using “subplot” 33
Figure 3.11. Example of using “plotyy” 33
Figure 3.12. Example of using “loglog” and “semilog” 34
Figure 3.13. Example of calculating a derivate of function 35
Figure 3.14. Example of using “plot3” 36
Figure 3.15. Example of using “mesh” command 36
Figure 3.16. Example of using”surf” function 37
Figure 3.17. Example of using “surfc” 37
Figure 3.18. Saving figures 38

Figure 3.19. The previous saved figure seen out of Matlab 38
Figure 4.1 Example of Conditions without using “if… end” 43
Figure 4.2 Debugging procedure 47
Figure 5.1. Opening graphical user interface without using Command Window 52
Figure 5.2. Result of opening graphical user interface 52
Figure 5.3. Result of click on “Blank GUI (Default)” 53
Figure 5.4. Example of first step to create GUI 54
Figure 5.5. Property inspectors for push button 54
Figure 5.6. The first example figure seen throw Command Window 55
Figure 5.7. The first example “Callbacks” from the “push button” part 55
Figure 5.8. How to get menu from editor 56
Figure 5.9. The “Callbacks” ot he second example 56
Figure 5.10. Example of writing commands in Callbacks 57
Figure 5.11. Result of the previous example 57
Figure 5.12. Result of example 2 57
Figure 5.13. Result of example 3 58
Figure 6.1. Result example of polynomial arithmetic 62
Figure 6.2. Example of calculating a derivate of function 63
Figure 6.3. Example result of function “polyfit” 64
Figure 7.1. Result of “tf” function 67
Figure 7.2. Example results of “zpk” and “tf(..’inputdelay’..)” functions 68
Figure 7.3. Example results of stat space functions 70
Figure 7.4. Example of adding annotation in stat space 70
Figure 7.5. Example of Discrete time Transfer function/ State-space 71
Figure 7.6. Blocs with functions under 74
Figure 7.7. Example of blocks 74
Figure 7.8. Example of impulse response 75
Figure 7.9. Example of step response 76
Figure 7.10. Example of Bode function 77
Figure 7.11. Example of Nyquist function 77
Figure 7.12. Example of Nichols function 78

Figure 7.13. Example of PID controller using commands 79
Figure 8.1. Simulink Library blocs 82
Figure 8.2. Sinks blocs 83
Figure 8.3. Commonly used blocs 83
Figure 8.4. Sources blocs 84
Figure 8.5. Simscape blocs 84
Figure 8.6. Example of creating Simulink model using transfer function 85
Figure 8.7. Result of the previous example 86
Figure 8.8. Example of changing parameters 86
Figure 8.9. Example of creating Simulink model using mathematical model 87
Figure 8.10. Location and choice of powergui 88
Figure 8.11. Example of no connections problem using physical model 88
Figure 8.12. Result of the example 3 90
Figure 8.13. Result of the mask operation 90
Figure 8.14. The simple pendulum model 91
Figure 8.15. Result of application of ODE to simple pendulum model 92
Figure 8.16. Result of Simulink stat space simple pendulum model 93
Figure 8.17. Result of using transfer function for the simple pendulum model 94

List of tables

Table 4.1. Relational Operations. It returns logical array of Matlab 41
Table 4.2. MATLAB functions' to debug 49

Table of Contents

Preamble I

List of Figures, list of table II

General introduction 2

Chapter I: Introduction to Matlab

I.1 Introduction 4

I.2 Matlab desktop components 4

I.3. Variables and simple operations 5

I.4. Matlab's example of commands

I.4.1 'who' and 'whos' 7

I.4.2 'clc' and 'clear' 7

I.4.3 Help, lookfor and doc 9

I.4.4 Rounding 10

I.5 Conclusion 11

Chapter II: Matrix Computation and Linear Algebra

II.1 Introduction 13

II.2 Scalar variables:

II.2.1 short and long format 13

II.2.2 Polar and complex representation 13

II.3 Vectors in Matlab

II.3.1 Vector arithmetic operations 14

II.3.2 Particular functions to the vectors 16

II.4 Matrix in Matlab:

II.4.1 Matrix definition, indexing and matrix operations 17

II.4.2 Operations on matrices element-by-element 19

II.4.3 Particular functions for the matrices 19

II.4.4 Elementary matrices
II.5 Use of structure in Matlab 22
II.6 Solving system of linear equation with Matlab 23
II.7 Conclusion 24

Chapter III: Visualization and programming

III.1 Introduction 25
III.2 Plotting in 2D
II.2.1 Potting functions 25
II2.2 Multiple 2D plot in same figure 27
III.2.3 Adding titles, axis labels and annotation 28
III.2.4 Subplot, plotyy, loglog and semilog 31
III.2.5 Animate figures 33
III.2.6 Function derivation 33
III.3 Plotting in 3D: Plot3, mesh, surf plot 34
III.4 saving figures in Matlab 36
III.5 Conclusion 37

Chapter IV: Programming using M-file

III.1 Introduction 39
III.5 Programming in Matlab: m-file script/m-file function 39
III. 6 Programming with conditions:
III.6.1 if… end 39
III.6.2 if… else… end 40
III.6.3 if ... elseif …else… end 40
III.6.4 Switch case otherwise end 42
III.7 Programming with loops:
III.7.1 for …end 43

III.7.2 while … end 44
III.8 Debugging programs 45
III.9 Conclusion 48

Chapter IV: Building Structure & Graphical User Interface "GUI"

IV.1 Introduction 50
IV.2 Use of structure in Matlab: Example 50
IV.3 Graphical User Interface "GUI" aim and definition 51
IV.3.1 First step 51
IV.3.1.1 Palette components 51
IV.3.1.2 Layout area 52
IV.3.2 Second step: 52
Property inspector
IV.3.3 Last step: 54
Callbacks
IV.3.4 Examples 54
V.4 Conclusion 58

Chapter VI: Polynomial & Curve fitting

VI.1 Introduction 60
VI.2 Polynomials in Matlab
VI.2.1 Roots and poly 60
VI.2.2 Evaluation 60
VI.2.3 polynomial arithmetic: 60
Addition, subtraction, multiplication & division

VI.2.4 Differentiation & integration 62

V.3 Curve fitting 63

V.4 Conclusion 64

Chapter VII: Control System Toolbox

VII.1 Introduction 66

VII.2 Commands used to build Models for LTI

VII.2.1 Commands used to get continuous time model: Transfer function 66

VII.2.2 Commands used to get continuous time Transfer State-space 68

VII.2.3 Discrete time Transfer function/ State-space 70

VII.2.4 Conversion between Transfer function / State-space 71

VII.2.5 Combining LTI Models 72

VII.3 Response Analysis

VII.3.1 Transient Response Analysis 73

VII.3.2 Frequency Response Analysis 74

VII.4 Introduction to control designing 76

VII.5 Conclusion 78

Chapter VIII: Introduction to Simulink

VII.1 Introduction 80

VII.2 Simulink: Getting start 81

VII.3 Building models 82

VII.4 Simulink related command window

VII.4.1M-file/ command windows' functions 88

VII.4.2. ODE solver function 89
VII.5 Conclusion 92
General conclusion 93
bibliograpie 94

General introduction

Prior to Matlab, scientific computations and graphs were a little challenging. Because it must always be defined programs as in FORTRAN or C. In this handout, we will present scientific calculations with command or functions of Matlab. In addition, we will see programming using Matlab.

All computations in various scientific domains require reel variables, a matrix, and occasionally a complicated ounce, like in the case of electricity. These are Matlab mark matrices. As a result, we will utilize MATLAB as a calculator in the first chapter and as a "matrix" calculator after that. The matrix problem Ax = b will then be easy to solve. When doing scientific calculations, graphing is essential. This can be individually altered as 2D or 3D visuals in Matlab. Plotting surfaces and in two and three dimensions will be shown to the students. More frequently than not, specific "programs" are required based on scientific experiments. Chapter 4 introduces the so-called function m-files, which significantly updates MATLAB as a programming language [1,3].

In general, a program needs to get input from user and gives output. In older versions of Matlab this is got by text based interfaces. Data visualisation or GUI in MATLAB is based on the “Handle” Graphics System in which the objects organised in a Graphics Object Hierarchy can be manipulated by various high and low level commands. Chapter 5, introduces the graphical “GUIs” programming in Matlab [4,6].

In general, a scientist must explain or obtain models after seeing graphs. Therefore, these real-world data graphs must suit a mathematical model. fitting curves. Chapter 6 includes curve fitting. We present the idea at the command window level. One of the primary goals after having models is to "control." In this manner, we shall describe "Control System Toolbox" in the following chapter. This one demonstrates linear system analysis methods, after which feedback control systems are analyzed and designed. The sevnth chapter covers design techniques for both classical and state-space models, as well as some instructions for frequency-response analysis, root-locus plots, and system performance.

Finally, Mathematics MATLAB is like a powerful calculator that gives access to explore problems in science, engineering, and mathematics. Simulink is a toolbox in Matlab used for this aim. The last chapter presents this toolbox and shows how to use it in modeling, simulation and analysis of linear or non-linear dynamic systems analogous or discrete.

Chapter I: Introduction to Matlab

I.1 Introduction

I.2 Matlab desktop components

I.3. Variables and simple operations

I.4 Matlab's example of commands

I.4.1'who' and 'whose'

I.4.2 'clc' and clear

I.4.3 'help', 'lookfor' and 'doc'

I.4.4 Rounding

Before using Matlab, it is important to know what is it, how it is work, how can get and present results with it, and what kind of information are used? In this chapter, we find some definitions and answer of these questions. The purpose of this chapter is to help students get started properly and the other chapters cover each functionality in depth with examples.

I.1. Introduction

The numerical analyst Cleve Moler wrote the first version of Matlab in 1970. Matlab stands for Matrix Laboratory. MAtrix LABoratory or MATLAB is an interactive software package which was developed to perform numerical calculations on vectors and matrices. Today it is more powerful: It can do graphics in two and three dimensions. It contains a high-level programming language (a "simple C") which makes it quite easy to code complicated algorithms. It can numerically solve nonlinear initial-value ordinary differential equations. It can numerically solve nonlinear boundary-value ordinary differential equations. This because it contains a variety of toolboxes which allow it to perform applications from science and engineering. In more users can write their own toolboxes!

The Toolboxes are sets of MATLAB functions dedicated to each of the different scientific fields such as control system Toolbox, signal processing Toolbox, neural network, wavelet, Acquisition, etc.

The Simulink tool allows graphical programming. It has block libraries allowing the simulation of dynamic systems. We can extend it by generating s-function. This is a block which contains Matlab code and which can be used in Simulink.

These advantages do not mean that MATLAB is complicated programming language, but it can be seen as a powerful calculator for problems in science, engineering, and mathematics. And this access is available by using only a small number of commands and function because MATLAB's basic data element is a matrix (or an array) [1, 2].

- **Why Matlab?**

1. Math and computation
2. Programming
3. Plotting data
4. Modelling and simulation
5. Data analyse
6. Systems control
7. Engineering graphs.

I.2. Matlab desktop components

Under Windows, a double click on the Matlab icon brings up the following window:

1. The Command window
2. The command history
3. The Workspace
4. The current Directory

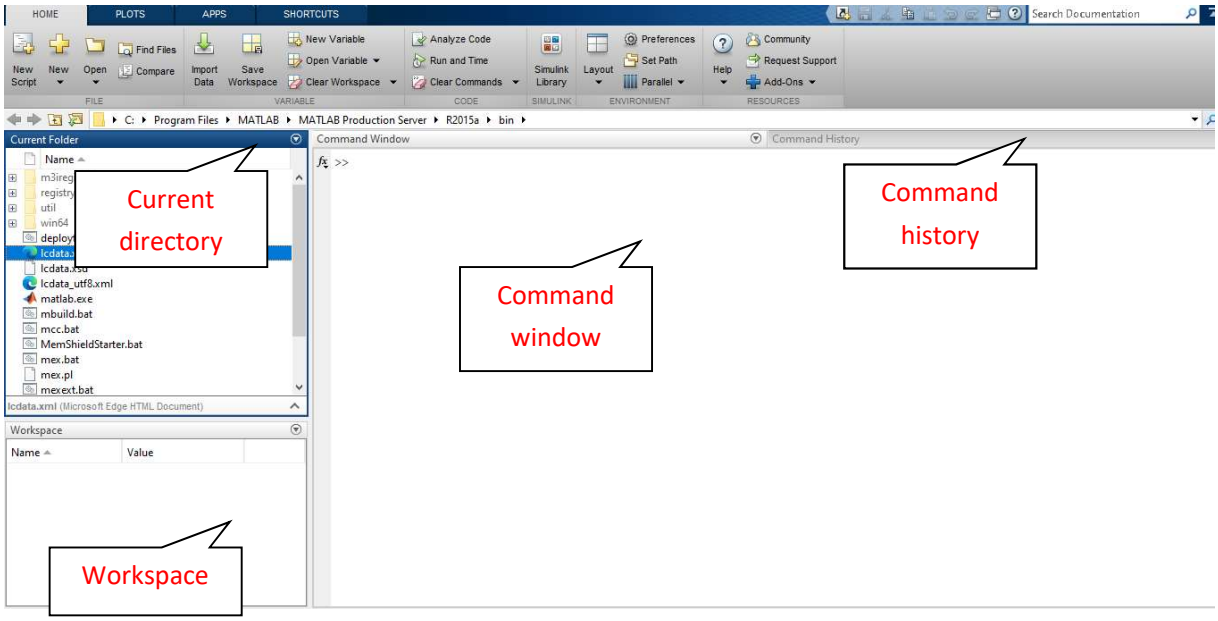


Figure 1.1 Matlab desktop components

In addition to drop-down menu the Matlab window offers icons as follows:

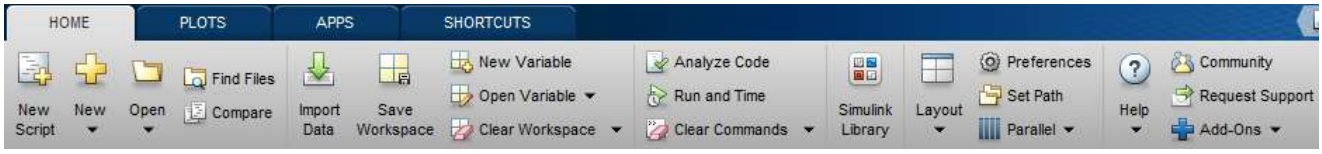


Figure 1.2 Matlab up icon

To open sheet (untitled) file editor command, to create a new document, to open a file or a program …etc.

I.3. Variables and simple operations

Matlab is matrix oriented. One can define or enter values of Variables on command window. Example:

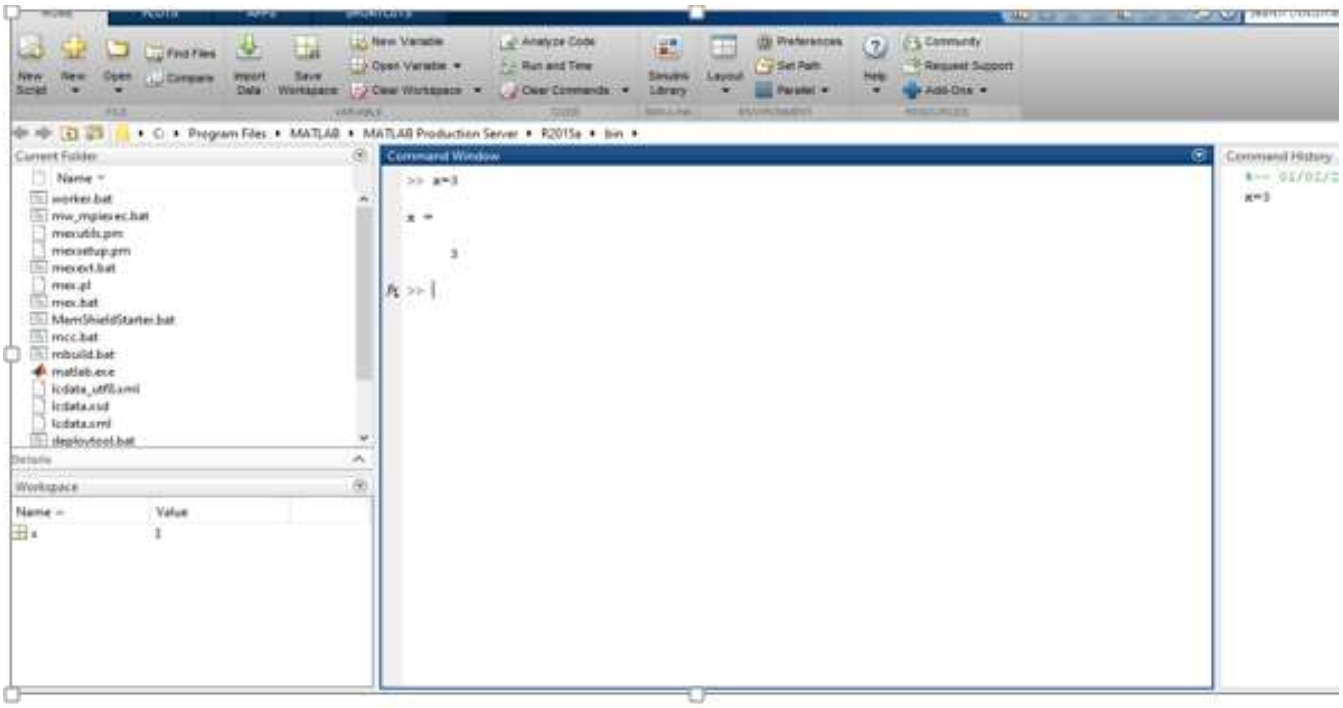


Figure 1.3 introducing scalar variable

Double clicking on the variable on workspace, it opens an array editor

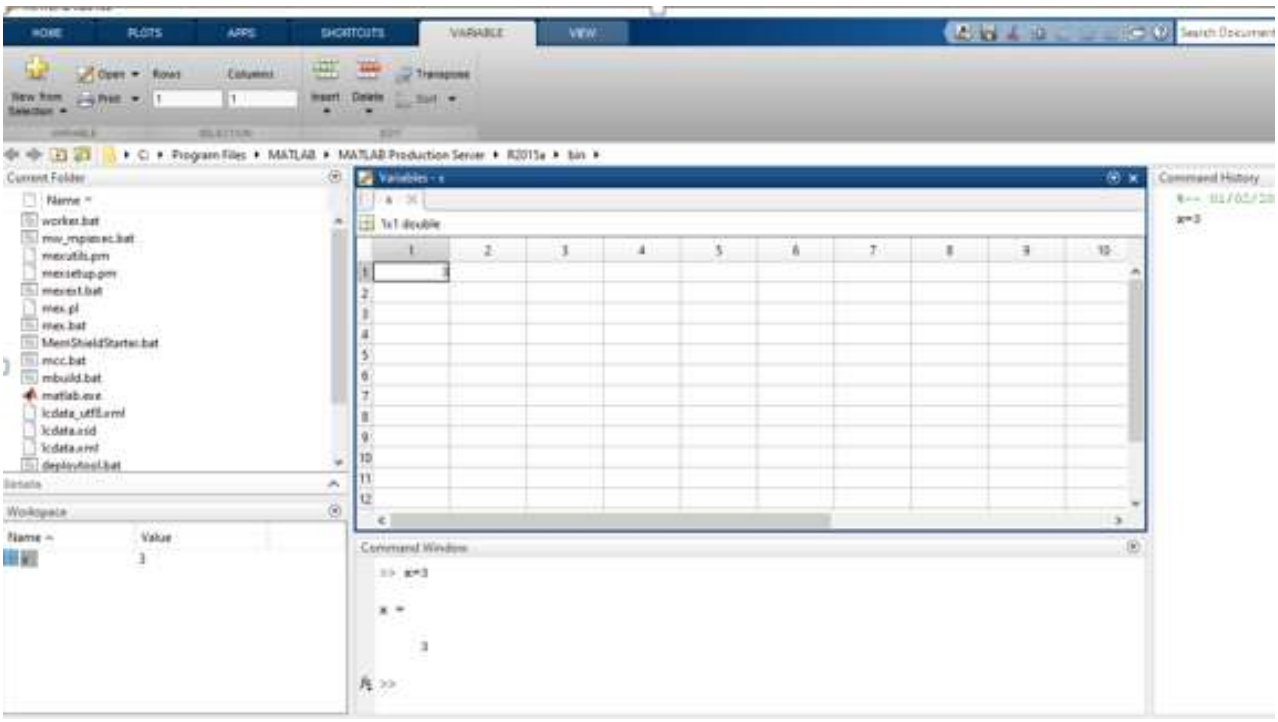


Figure 1.4 Variable poisoning in work space

I.4. Matlab's example of commands (see the help of Matlab)

I.4.1 'who' and 'whos'

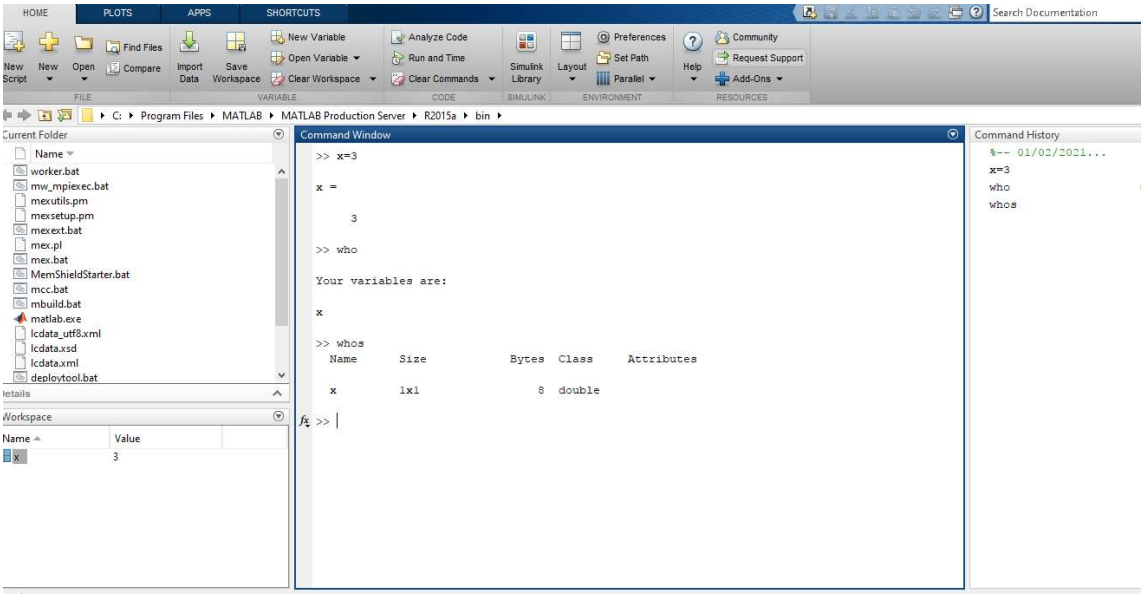


Figure 1.5 Example result of the commands who and whos

The command 'who' presents all variables in workspace. 'whos' gives in more their sizes and their types.

I.4.2 clc and clear

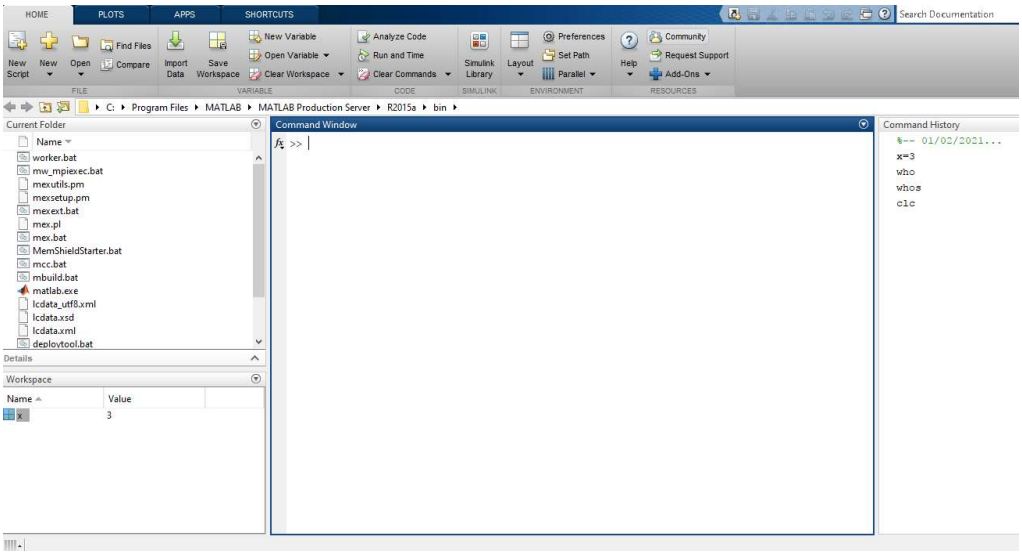


Figure 1.6 Example result of the command clc

clc : keeps workspace variables and clear the screen, Clear: Get remove workspace variables (fig1.7) or the indicated once (see fig 1.8).

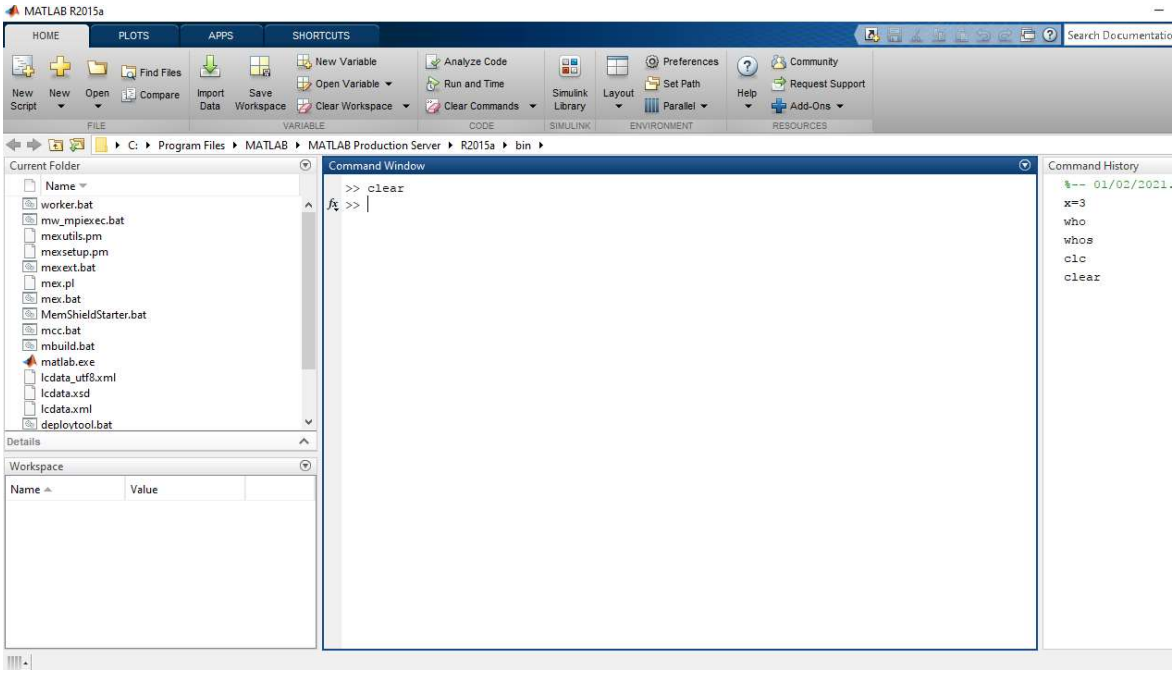


Figure 1.7 Example result of the command clear

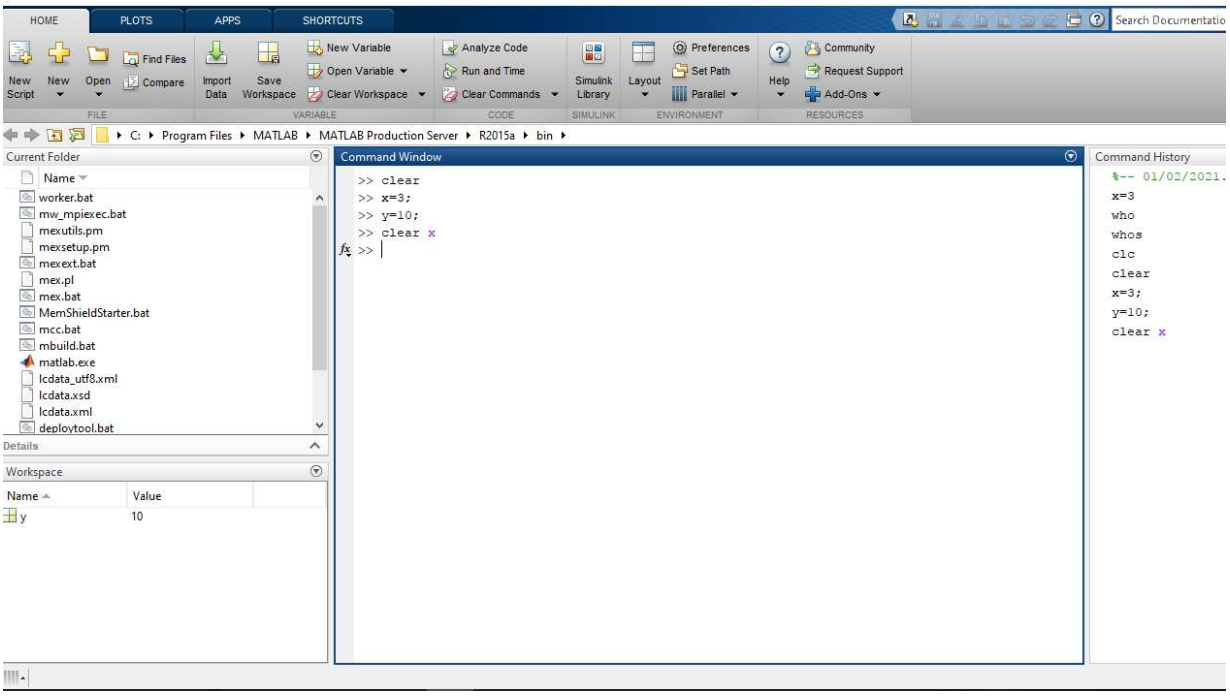


Figure 1.8 Example result of the command clear applied to one variable

I.4.3 help, lookfor and doc

'help' requires the precise function name, whereas 'lookfor' provides all relevant names and 'doc' as help but plus more detail.

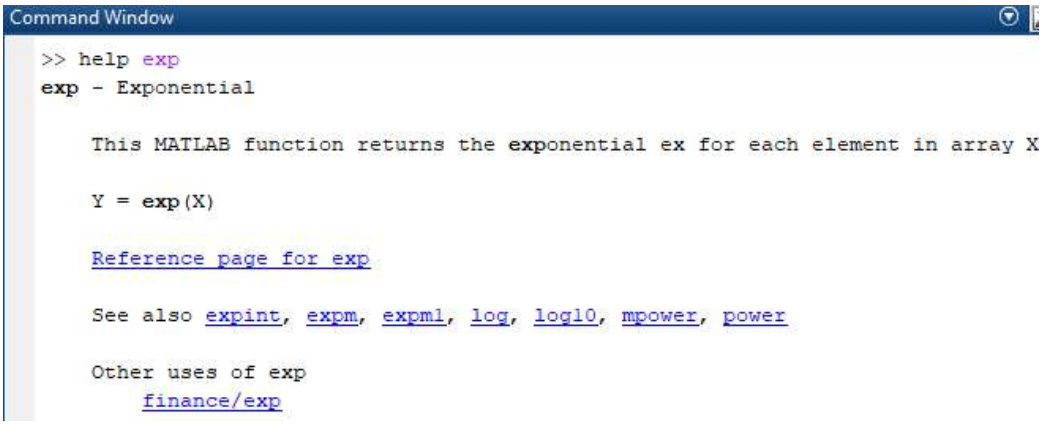
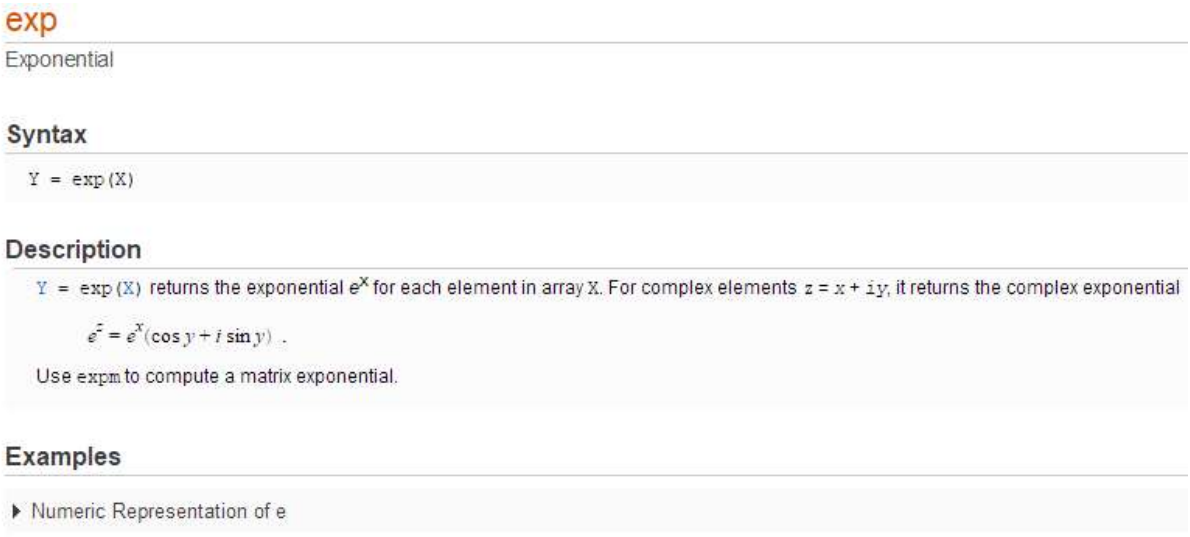


Figure 1.9 Example result of the command help

Figure 1.10 Example result of the command lookfor



exp

Exponential

Syntax

```
Y = exp(X)
```

Description

`Y = exp(X)` returns the exponential e^X for each element in array `X`. For complex elements $z = x + iy$, it returns the complex exponential

$$e^z = e^x(\cos y + i \sin y) .$$

Use `expm` to compute a matrix exponential.

Examples

▸ Numeric Representation of e

Figure 1.11 Example result of the command doc

I.4.4 Rounding

Fix(.X) → It rounds every X component to zero, round(X) → It rounds every X component to the closest integer number,

floor(X) → It rounds every X component toward $-\infty$, ceil(X) →round It rounds every X component toward $+\infty$

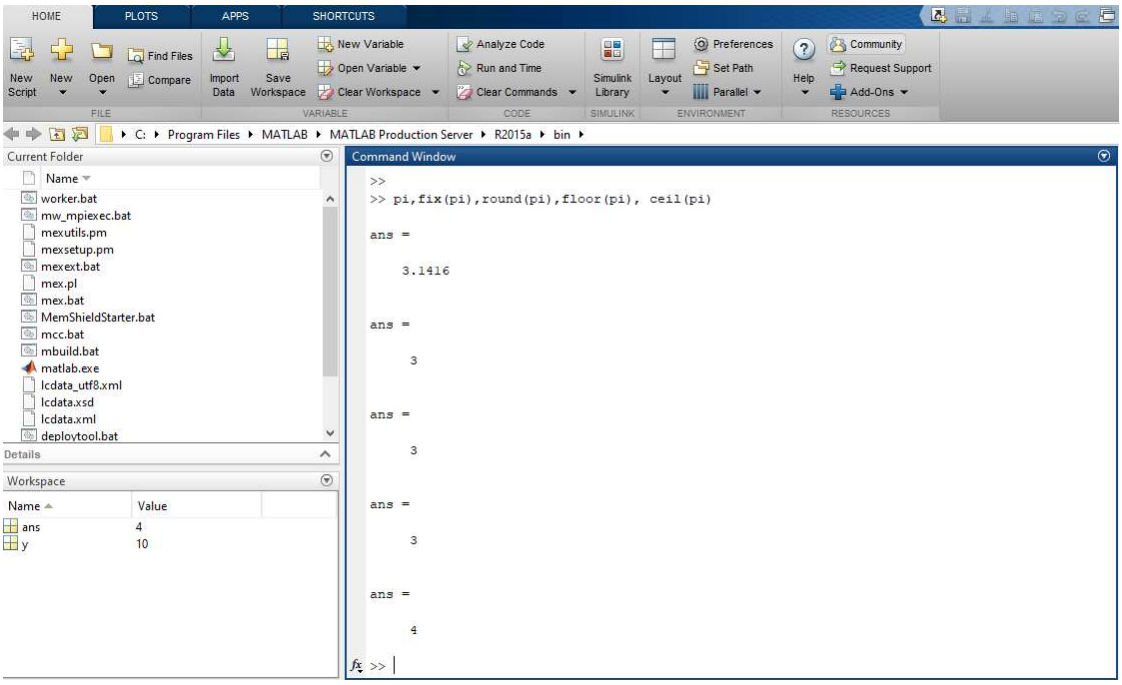


Figure 1.12 Example result of the command fix, round, floor, and ceil

I.5 Conclusion

In conclusion, the aim of this first chapter was to introduce calculation with Matlab. That had demonstrated in several examples respectively. These examples serve to learn in a more detailed manner, while examining the following chapters.

Chapter II: Matrix Computation and Linear Algebra

II.1 Introduction

II.2 Scalar variables:

Short and long format, Polar and complex representation

II.3 Vectors in Matlab:

Arithmetic operations, Particular functions for vectors

II.4 Matrix in Matlab:

Definition, indexing and matrix arithmetic operations, Matrix Operations on matrices element-by-element, Particular functions for the matrices, elementary matrices.

II.5 Use of structure in Matlab

II.6. Solving system of linear equation with Matlab

II.7 conclusion

Most important part in scientific calculation uses « tables » to present and save result data. However, in most computer languages it have to work on each piece of data separately and use loops to cycle over all the pieces. In MATLAB this can frequently do complicated data in one, or a few, statements (avoid loops). In this chapter one can see the manner, in Matlab that can be presented. One might additionally discover how Matlab can be used to perform linear algebraic computations, including determining determinants and solving linear equations.

II.1. Introduction

A powerful characteristic of Matlab is that all variables are defined by default as a matrix with complex elements. Thus, all real numbers are considered as a one-row, one-column matrix whose imaginary parts are zeros. In this chapter we will see how to use MATLAB as a "scalar" calculator, and then as a "matrix" calculator. Following this, we will be able to solve the matrix equation Ax = b where A is a square matrix.

II.2. Scalar variables

II.2.1 Short and long format

```
>> x=pi

x =

    3.1416

>> format long
>> x=pi

x =

   3.141592653589793

>> format short
>> x=pi

x =

    3.1416
```

Figure 2.1 Variables in short and long format

'format' to change the display form. With 'short' only 4 values after comma and 'long' there are more!

II.2.2 Polar and complex representation

- let's take a complex number $x = 4 - 3i$

→ $x = 5\langle -0.64$ with $\begin{cases} a = 5 = \sqrt{4^2 + 3^2} \\ b = -0.64 = tan^{-1}(-\frac{3}{4}) \end{cases}$

Matlab accepts complex numbers in algebraic or polar form. To display, it always uses the 'i' to represent the number $i^2 = -1$ but accepts the notation i and j (i.e. $j^2 = -1$ and $i^2 = -1$).

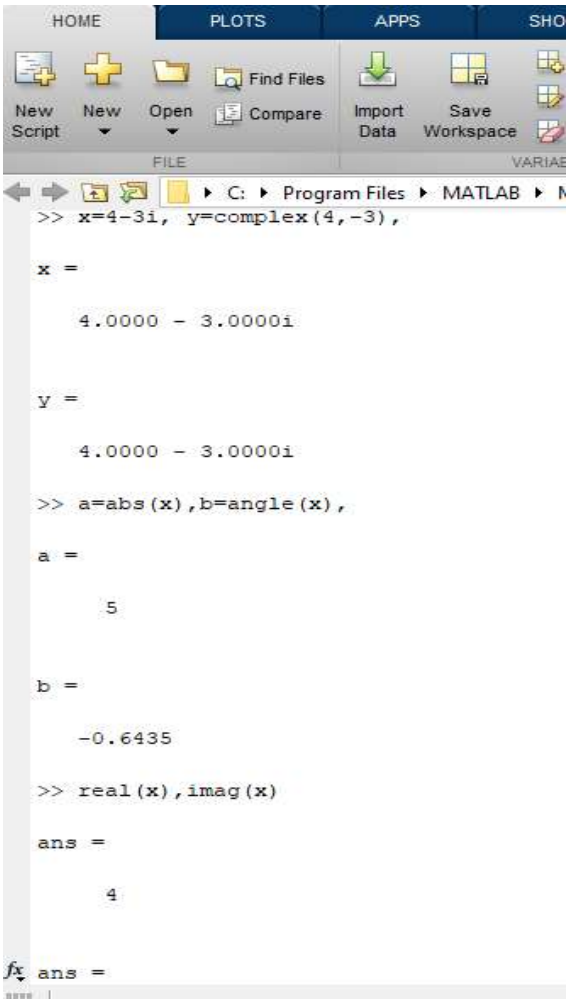


Figure 2.2. Different Polar presentations

II.3 Vectors in Matlab

Matlab has functions specific to vectors as well as the matrix

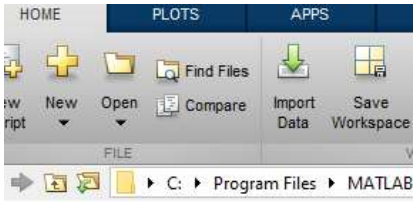
II.3.1 Vector arithmetic operations

- a row vector can be written:

```
>> x=[1 2 3 4],

x =

     1     2     3     4

>> x=[1, 2, 3 ,4]

x =

     1     2     3     4

>> x=1:4

x =

     1     2     3     4

>> x=1:2:8

x =

     1     3     5     7
```

Figure 2.3. Defining a row vector in Matlab

- a column vector can be written:




```
>> y=[1; 2; 3 ;4]

y =

     1
     2
     3
     4
```

Figure 2.4. Defining column vector in Matlab

- addition cannot be between column and row vectors but multiplication or division is possible when vectors has the same number of elements:

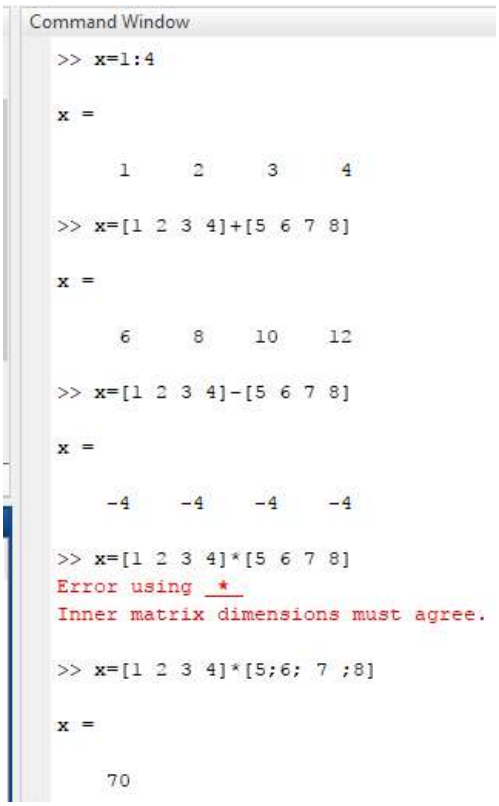


Figure 2.5 Operations rules with vector in Matlab

II.3.2 Particular functions for the vectors

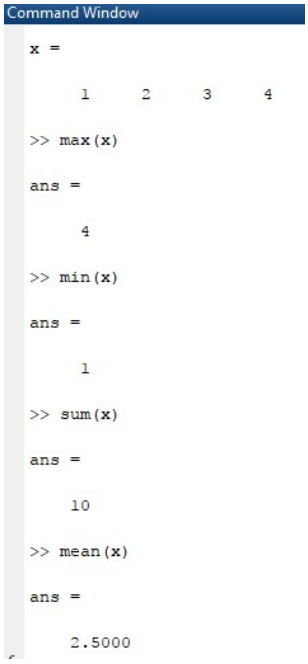


Figure 2.6 Example of Particular functions: Max/min, Sum/mean

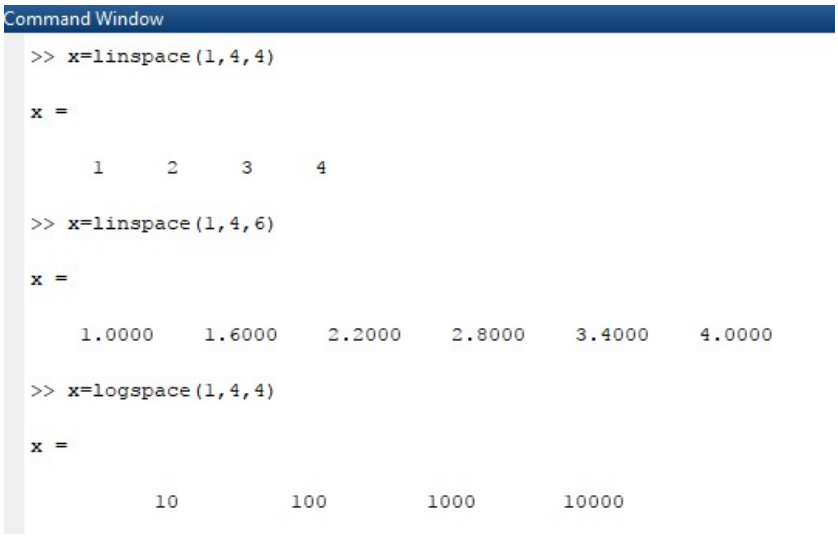


Figure 2.7 Example of Particular functions: linspace/logspace

II.4. Matrix in Matlab

II.4.1 Matrix definition, indexing and matrix operations

- To define matrix in Matlab , we use:

  Square bracket, then the element of 1rst row (separated by commas or space, we step the second row by a semi column …etc. then we end by the square bracket.

Example:$A = \begin{bmatrix} 2 & 6 & 8 \\ 1 & 5 & 3 \\ 0 & 9 & 7 \end{bmatrix}$

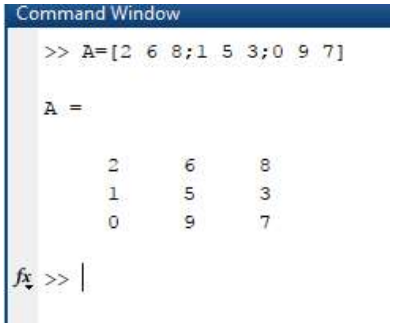


Figure 2.8 Defining matrix

- To index matrix, we use parentheses" (index row, index column)", so that we locate one or some elements.
- For sub matrix we use vectors of row and column indices: A([indexes of rows],[indexes of columns]) or a column operator A(m:n,k:l)

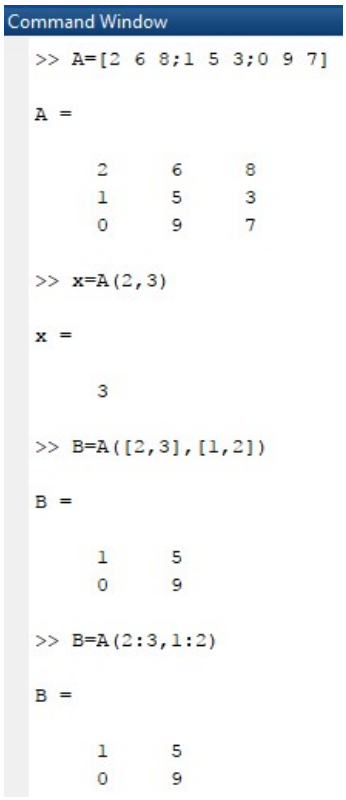


Figure 2.9 Example of indexing operations

- We can add or delete a vector or a sub-matrix. example:

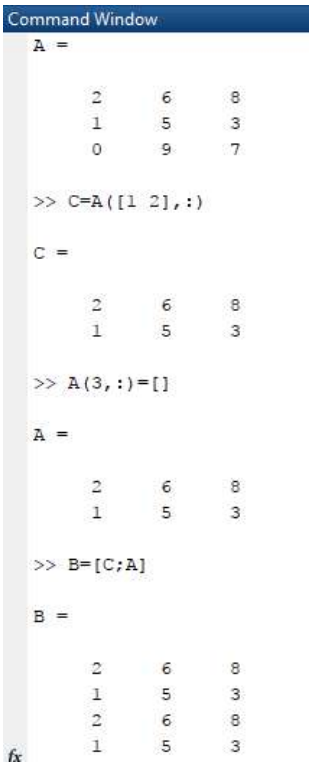


Figure 2.10 Example of defining sub-matrix

A three-dimensional array is a stack in three matrix dimensions: row index, column and page.

```
>> M(:,:,1)=A;%creat first page
>> M(:,:,2)=[55 0 4;9 3 1;10 .9 2];%%creat second page
```

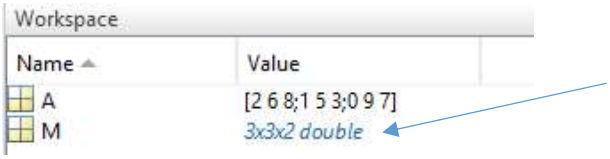


Figure 2.11 Example of Three dimensional array

II.4.2 Matrix operations element-by-element

```
Command Window
     2     6     8
     1     5     3
     0     9     7

>> B=[linspace(1,4,3);linspace(1,10,3);[9 7 4]]

B =

    1.0000    2.5000    4.0000
    1.0000    5.5000   10.0000
    9.0000    7.0000    4.0000

>> A*B

ans =

   80.0000   94.0000  100.0000
   33.0000   51.0000   66.0000
   72.0000   98.5000  118.0000

>> A.*B

ans =

    2.0000   15.0000   32.0000
    1.0000   27.5000   30.0000
         0   63.0000   28.0000
fx
```

Figure 2.12 Example of matrix operation element-by-element

II.4.3 Particular functions to matrices

- Dimension and transpose are given such as:

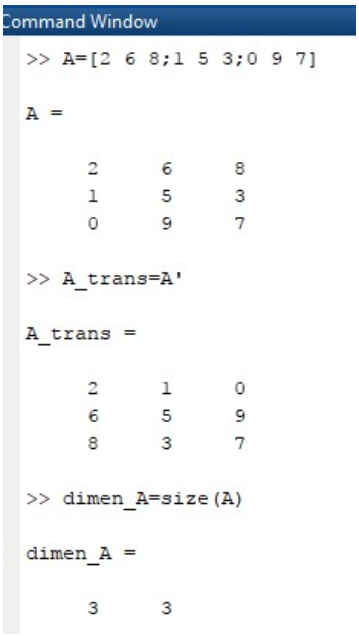


Figure 2.13 Example of Particular functions: transpose and size

- inv → the inverse of matrix, norm→matrix (or vectors) norm, rank→ number of linearly independent vectors.

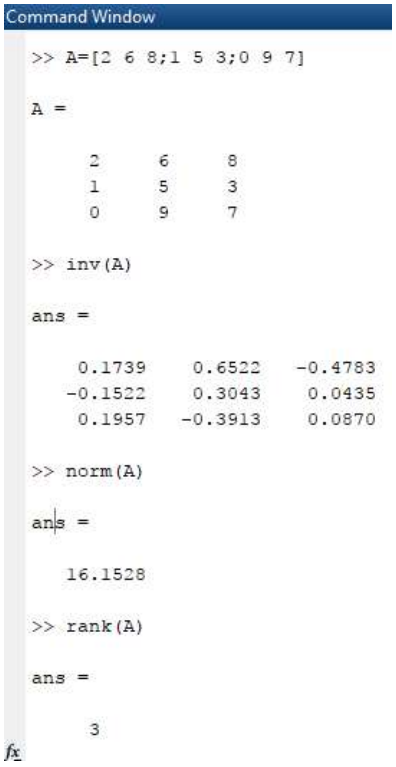


Figure 2.14 Example of particular functions: norm/rank

- det → determinant, diag→ diagonal elements or A(i,i) and eig → the Eigen values

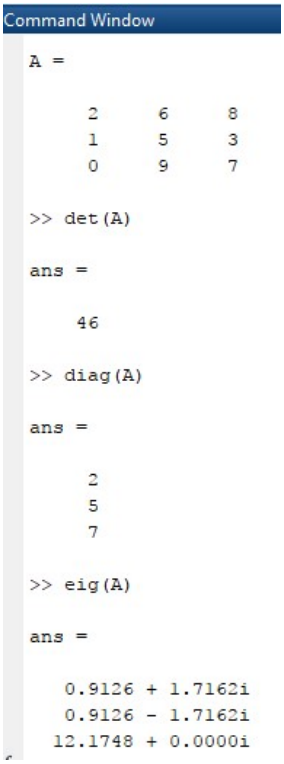


Figure 2.15 Example of particular functions: diag/det/eig

- repmat: containing n copies of A in the row and column,

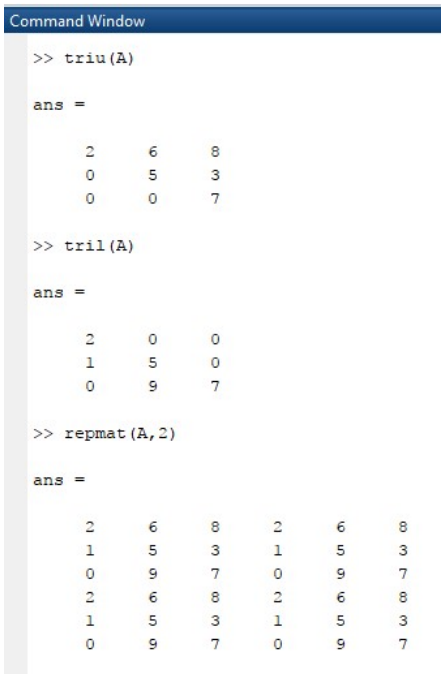


Figure 2.16 Example of particular functions: triu/tril, repmat

- triangular upper /lower→ the upper / lower elements of A.

II.4.4 Elementary matrices

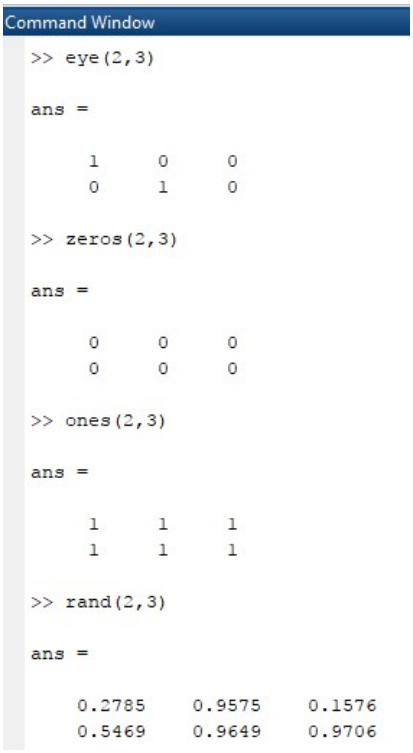


Figure 2.17 Example of specific matrices: eye/zeros, ones, rand

II.5 Use of structure in Matlab

An ordinary variable in Matlab (vector, matrix, or Tree dimensional array) contains elements with the same type and dimension. Unlike structure, elements can have different characteristics also; the access to element is by **names** instead of **indices**.

Example (student has name, date, mark)

```
>> M01(1,1,1).name='sss';M01(1,1,1).date='18/12/13' ;M01(1,1,1).mark=[14 13 10];
>> M01(1,2,1).name='ttt';M01(1,2,1).date='10/02/13' ;M01(1,2,1).mark=[17 15 12];
>> M01(1,1,2).name='uuu';M01(1,1,2).date='11/11/13' ;M01(1,1,2).mark=[10 14 16];
>> M01(1,2,2).name='vvv';M01(1,2,2).date='03/05/13' ;M01(1,2,2).mark=[16 13 11];
>> M01

M01 =

1x2x2 struct array with fields:

    name
    date
    mark
```

```
>> L01=struct('name',{'ccc','ddd','fff','ggg'},'date',{'16/02/15','09/04/15','02/10/15','29/03/15'},'mark',{[16 11],[13 10],[11 09],
>> L01

L01 =

1x4 struct array with fields:

    name
    date
    mark
```

It can be noticed that we can apply any Matlab command to the field of a structure

```
>> mean([L01.mark])

ans =

   10.8750
```

```
>> mean([L01(1,1).mark])

ans =

   13.5000
```

II.6 Solving system of linear equation with Matlab

A linear system $A\,x = B$ with: A is n×n and B is n×1

Has the solution: $x = A^{-1}B$.

In Matlab this can be done by several manners:

Example:

$$\begin{cases} 3x_1 + 9x_2 = 66 \\ 4x_1 + 7x_2 = 3 \end{cases}$$

In the next figure, one can see how to solve the system with three manners.

```
>> A=[3 9;4 7];B=[66;3];
>> X=A\B

X =

   -29
    17

>> X=inv(A)*B

X =

   -29
    17

>> C=[A,B]; X_ref=rref(C);X=X_ref(:,end)

X =

   -29
    17

>> X = linsolve(A,B)

X =

   -29
    17
```

Figure 2.18 Example of linear equation with Matlab

II.7 Conclusion

Matrix computation is covered in this chapter. From the simplest "1×1" matrix or scalar values to "n×m×p" matrix. It shows how it is easy to deal with! Then, the linear Algebra calculations are exposed. Some useful commands are given in examples. Furthermore, this chapter finishes with approaches to resolving linear equations.

Chapter III: Visualization in Matlab

III.1 Introduction

III.2 Plotting in 2D:

Potting functions, multiple 2D plot in same figure, titles, axis labels and annotation, Subplot, plotyy, loglog and semilog, animate figures, function derivation

III.3 Plotting in 3D: Plot3, mesh, surf plot

III.4 saving figures in Matlab

III.5 Conclusion

The manner to present scientific calculation with graphs was some quit default before Matlab. Since it needs to defined programs each time as in FORTRAN or C. One of the reasons Matlab was created was to make graphing easier. The primary commands or functions that are defined and their applications are presented in this chapter.

III.1. Introduction

A given graph invented several objects and properties. In Matlab, this can be modified independently as 2D or 3D graphics. Student will learn to plot in two and three dimensions and to plot surfaces. Then, one can see Matlab as a programming language and how can write the own MATLAB commands, called function m-files [1,3].

III.2 Plotting in 2D

II.2.1 Potting functions

The main function is plot. We can use also, stem, stairs, bar, histogram…all are used to plot graphs between two vectors X and Y. The command uses the first vector for x-axis, the second vector for y-axis [1].

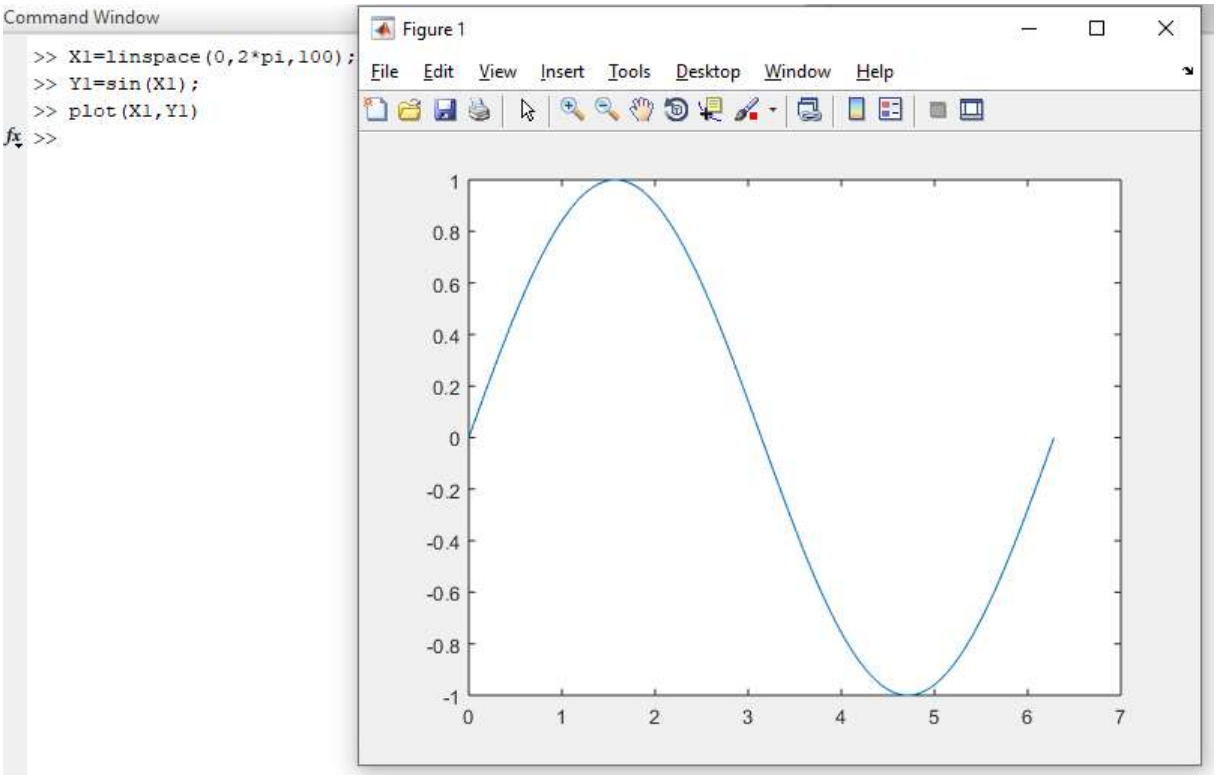


Figure 3.1 Example in using the "plot" function

If we use only one vector, it will be as in y-axis and the x-axis is will be taken from the indices. In the example so after, the function 'stem' is used.

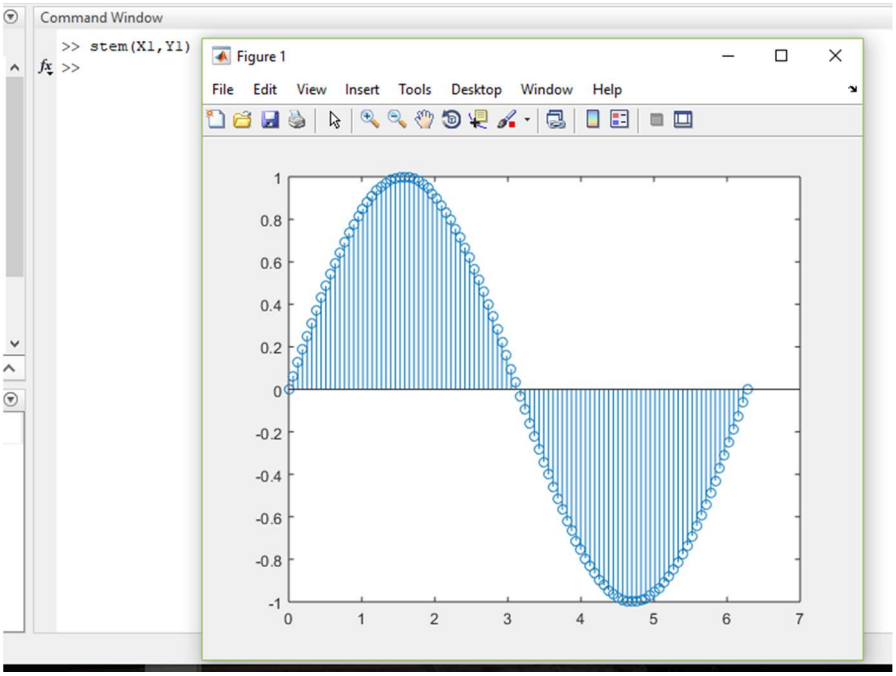


Figure 3.2 Example in using the "stem" function

The same axis rules as hold to the plot function are also applicable. But as we can see, it gives an additional presentation in terms of adding vertical lines vertical lines from the x-axis to the y values that terminate in circles. The command, in the next figure, draw the curve in the form of steps of stairs.

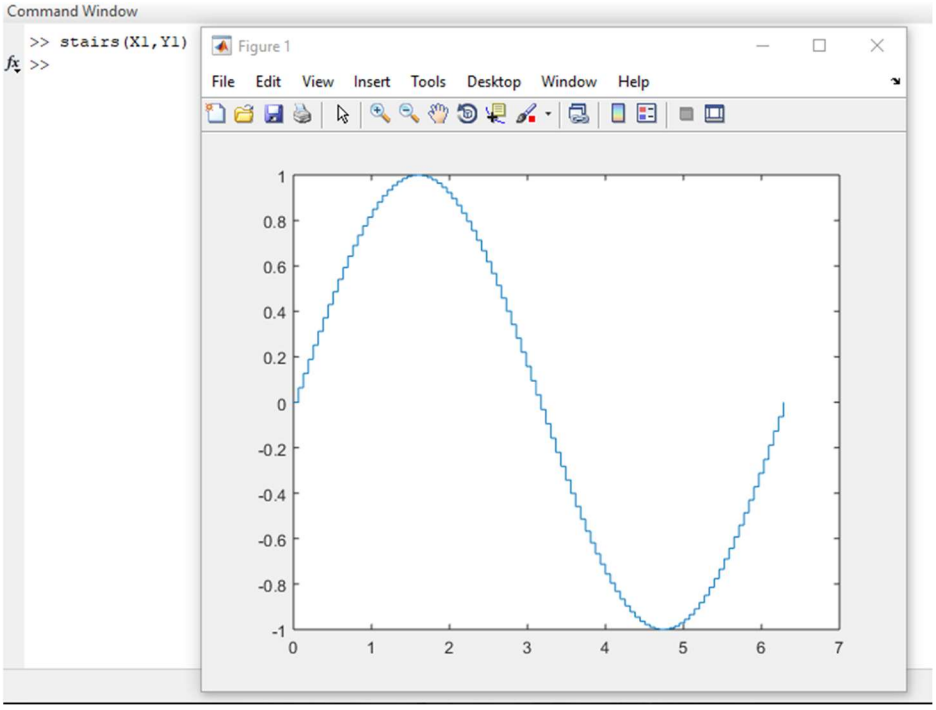


Figure 3.3 Example in using the "stairs" function

III.2.2. Multiple 2D plot in same figure

With plot, one can draw several curves at the same time in the same figure. To do it, there is two manners:

plot(X1,Y1,X2,Y2)

or

plot (X1,Y1), hold on

plot(X2,Y2) ,hold off

The result is the same.

For the other function (stairs, stem) only the second manner is used.

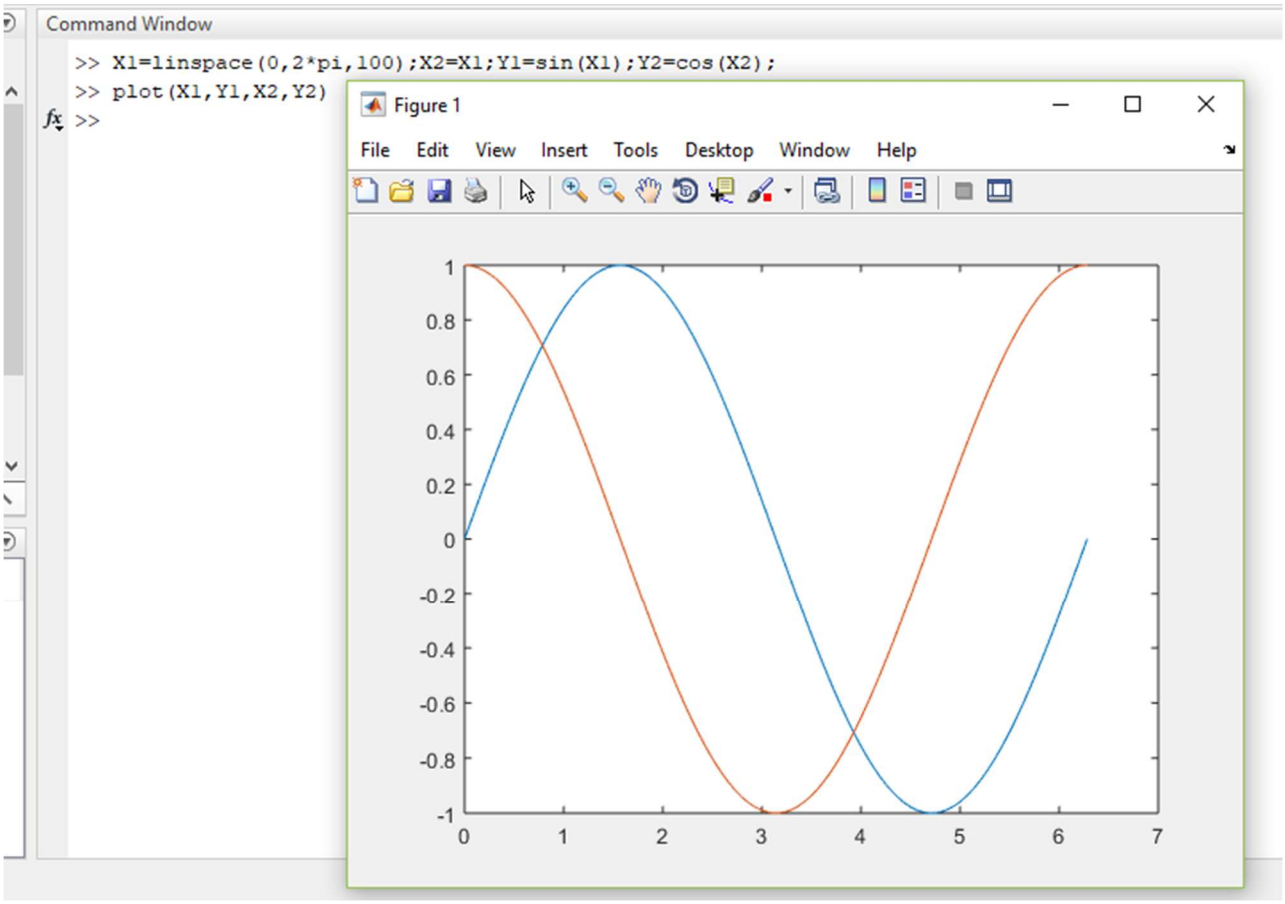


Figure 3.4 Use of two plots in the same figure

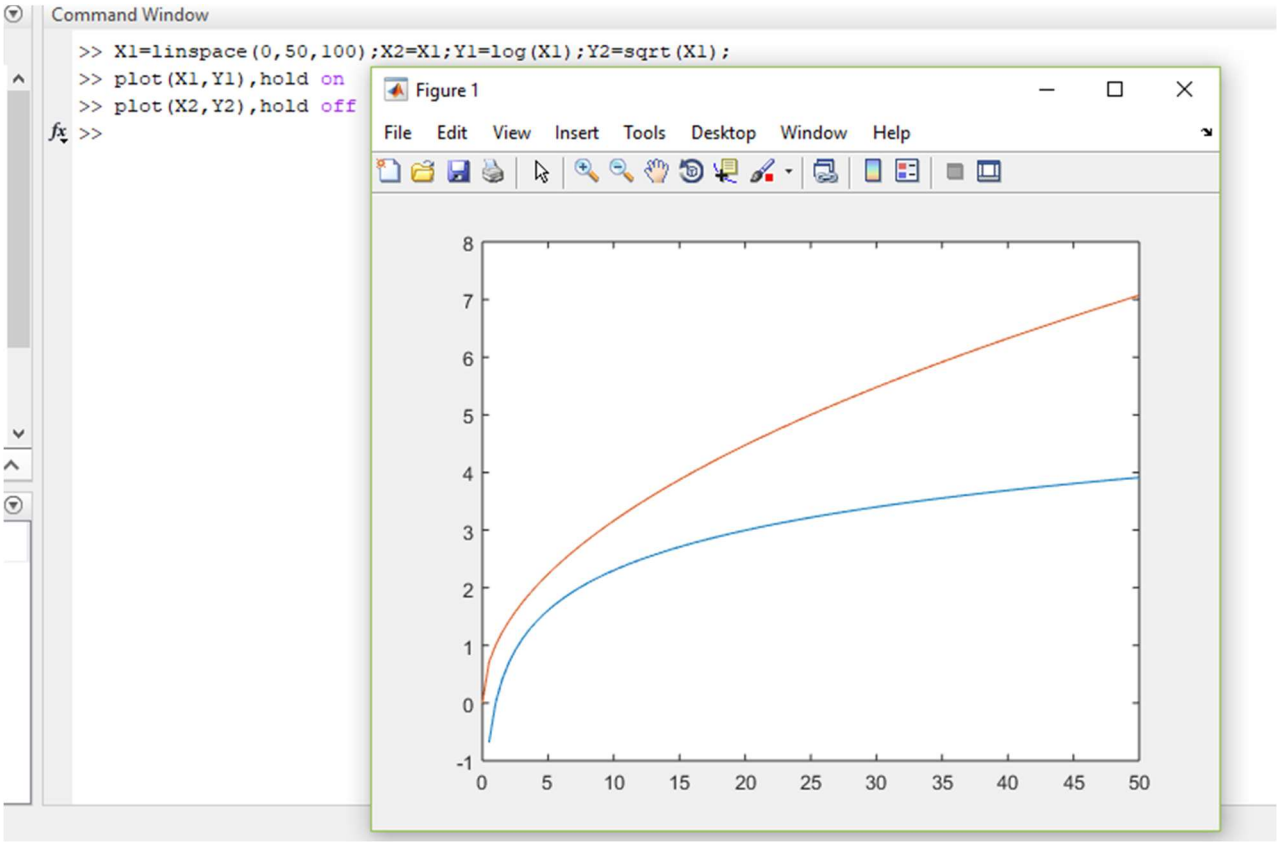


Figure 3.5 Use of two plots in the same figure (second manner)

The command allows the vectors: y1 and y2, we function respectively on x1 and x2.

III.2.3 Adding titles, axis labels and annotation

plot(X1,Y1,'color dot_form line_form','LineWidth',n);

plot(X1,Y1,'color dot_form line_form', X2,Y2,'color dot form line form','LineWidth',n);

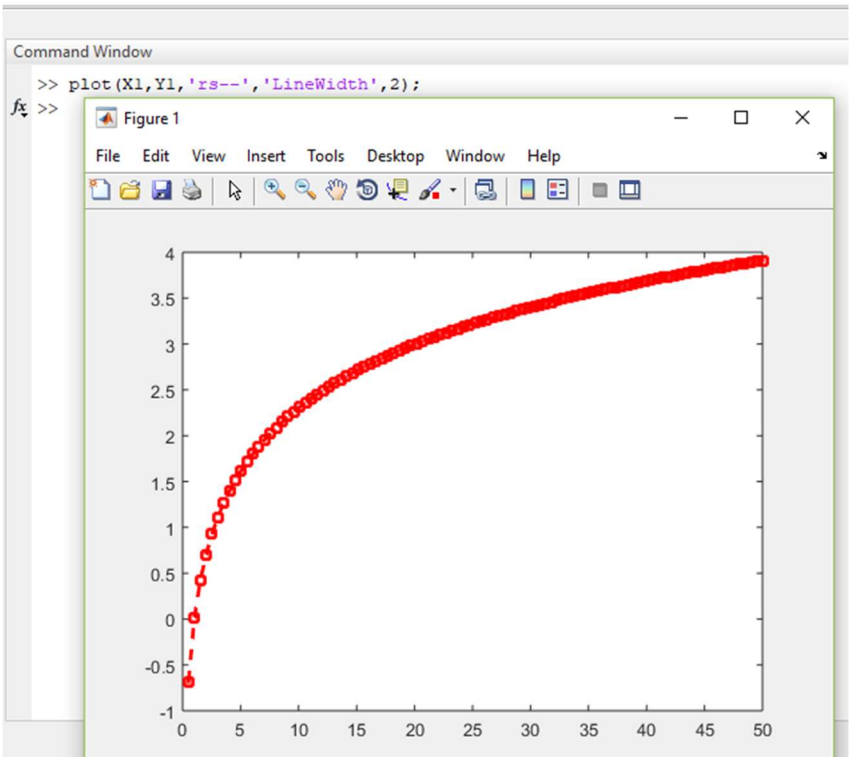


Figure 3.6 Example of using annotation in plot

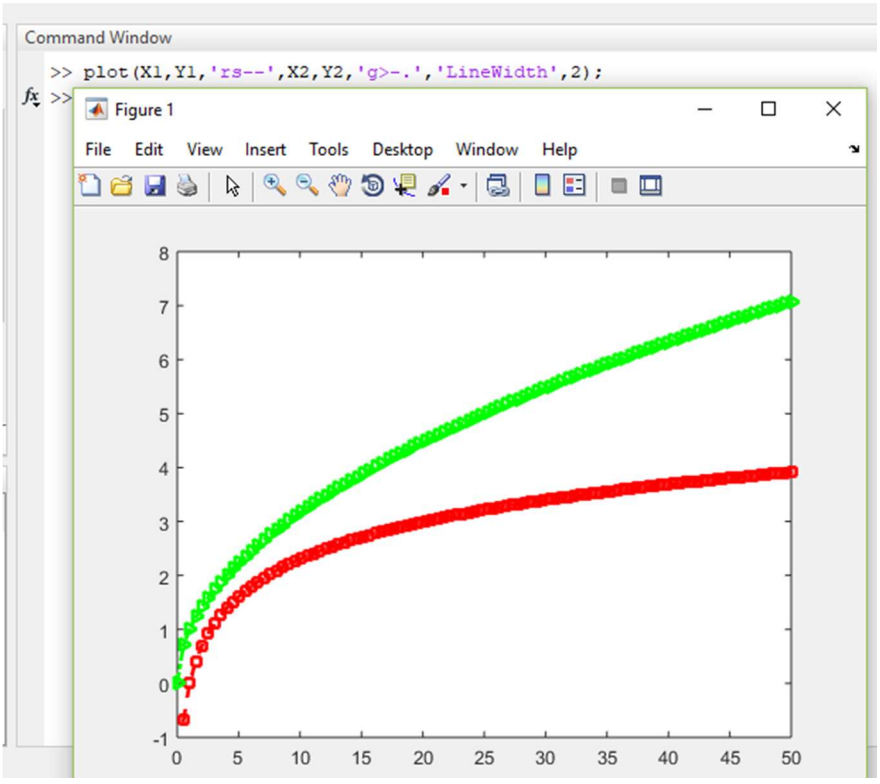


Figure 3.7 Example of using annotation for different plots in same figure

The “Grid” is used to put up the major grid lines (see figure after),

legend(‘firstName’,’second_name’) is used to creates a box where ‘firstName’ specifies the first plot, ’second_name’ specifies the second plot,…etc. The “title” is to write titles.

Also, “ xlabel”, “ylabel” are used to poster label axis. And “ xlim”, “ylim” to limit axis.

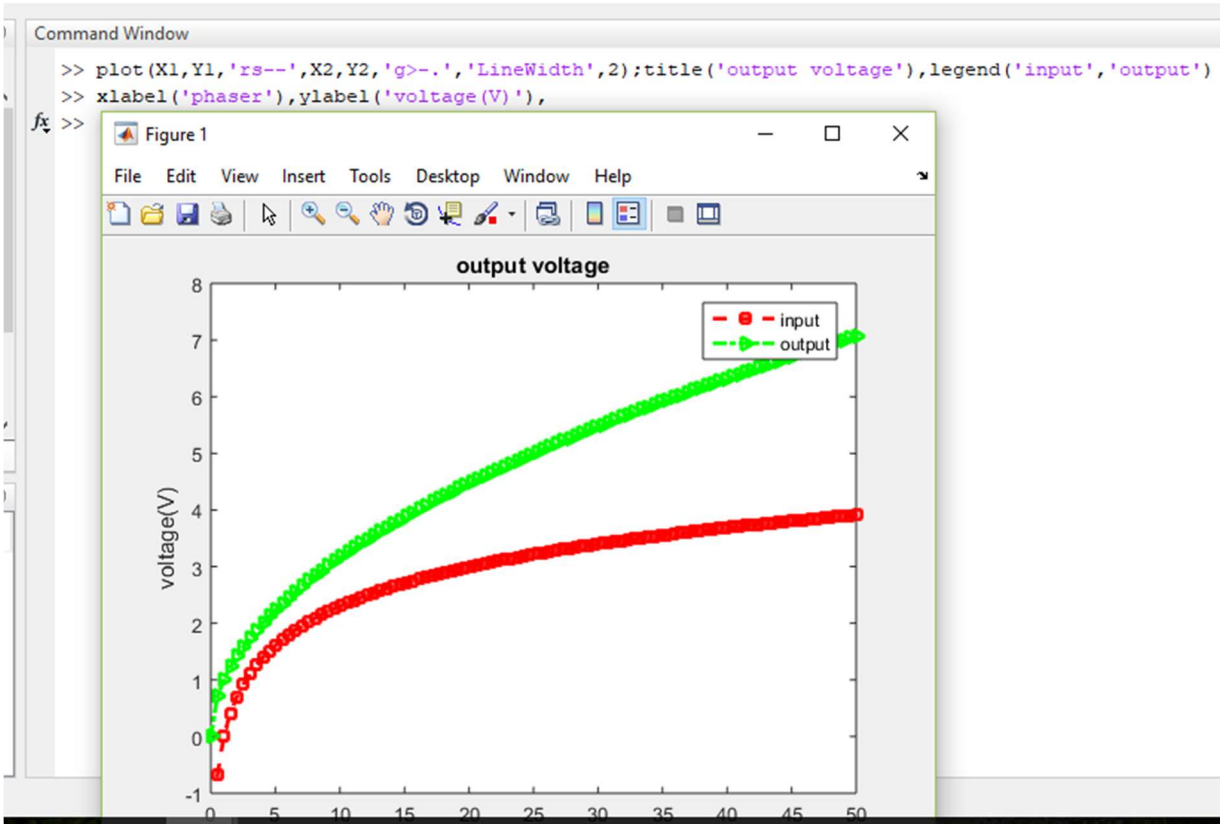


Figure 3.8 Adding Title, legend ,xlabel/ylabel to the figure

Another manner to revise the default figure is using “gca”/ “gcf” proprieties. These commands allow you to enter the figure structure and correct all the characteristics (fields) of figure/axis. But if there are several figures, the command is applied to the **current** once: It handles the current figure/axis. An example is given after.

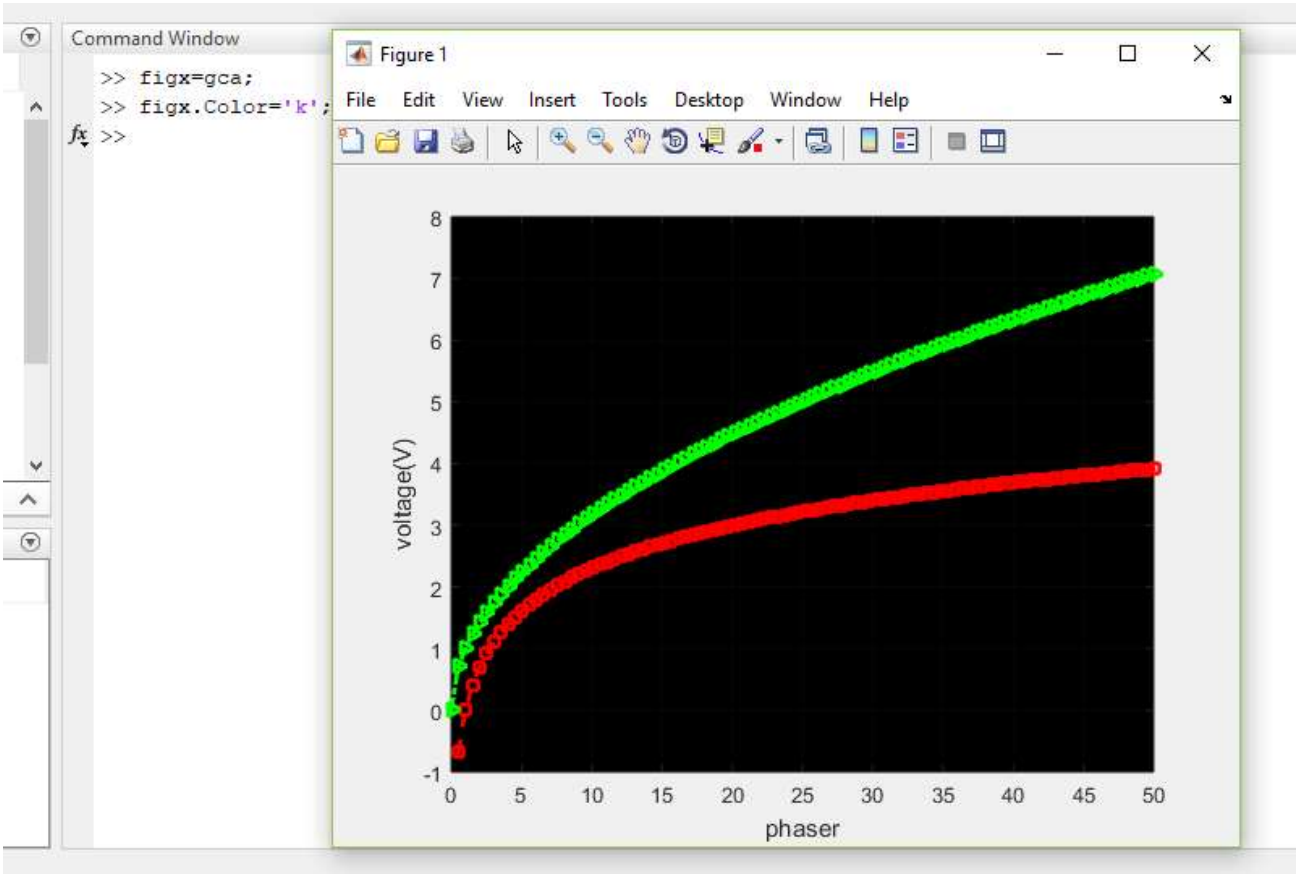


Figure 3.9 Example of using "gca"

III.2.4 Subplot, plotyy, loglog and semilog

subplot(m,n,p) function divides the figure into n×m. The plot just after takes the position p. plotyy is used to plot Y1 versus X1 on the left and plots Y2 versus X2 on the right, loglogis to plot using a base 10 logarithmic scale for the both axis and semilogx is to plot using a base 10 logarithmic scale for the x-axis.

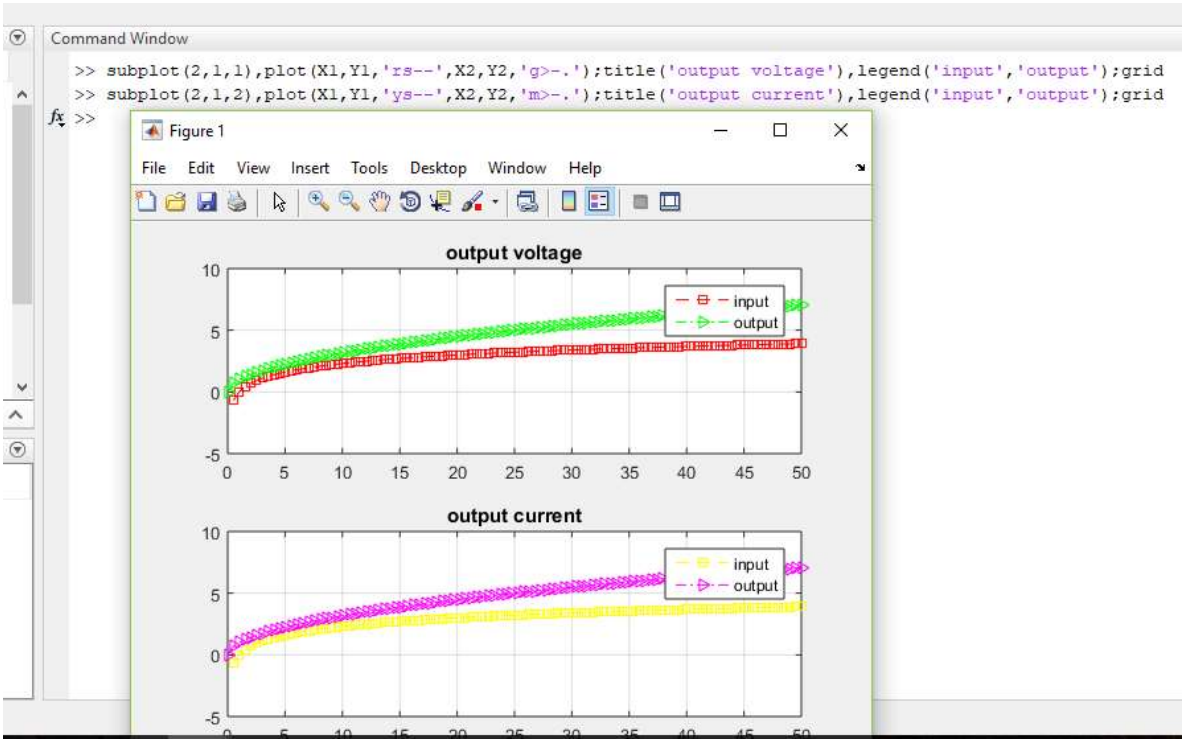


Figure 3.10. Example of using "subplot"

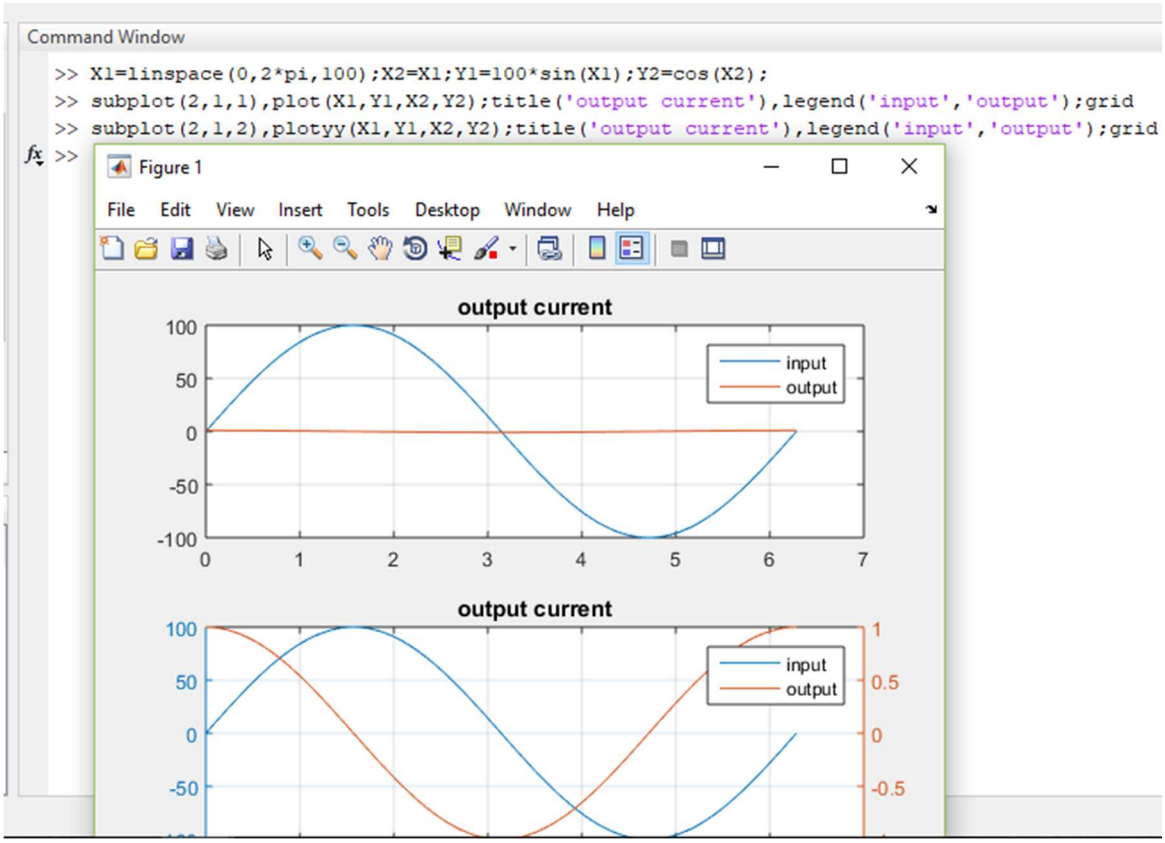


Figure 3.11. Example of using "plotyy"

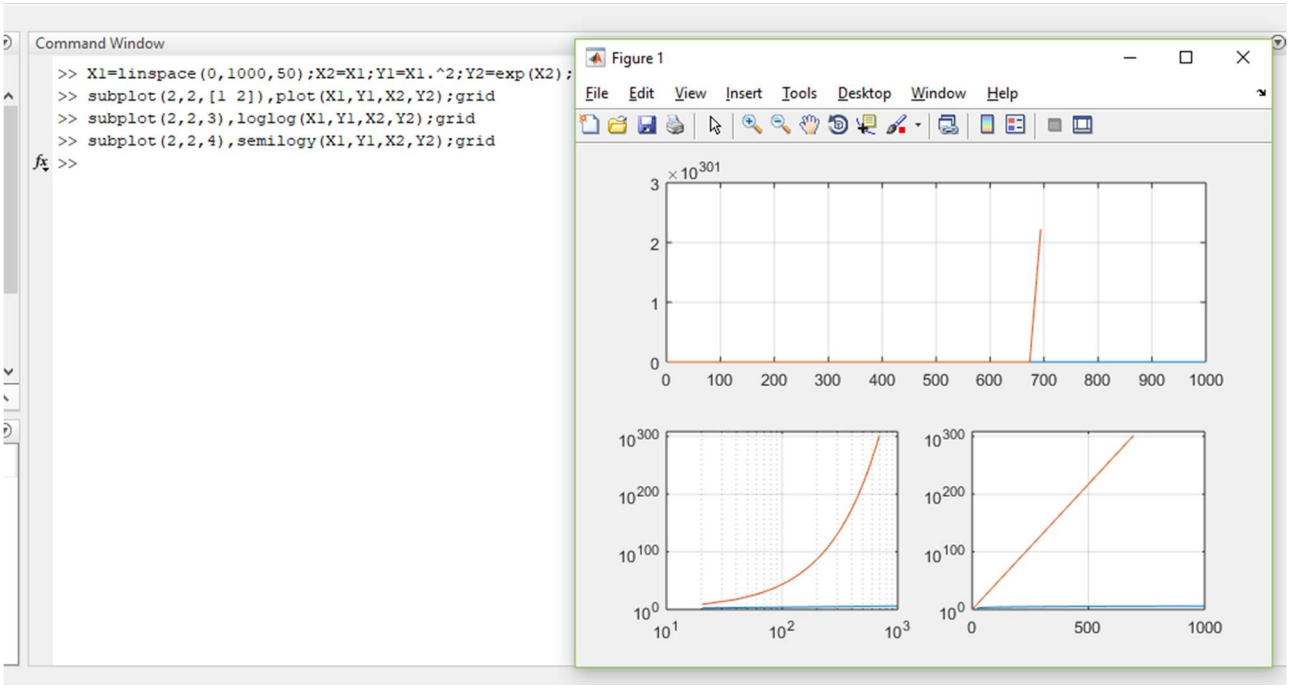


Figure 3.12. Example of using "loglog" and "semilog"

III.2.5 Animate figures

1)for.. end (pause)

```
>> x=0:pi/10:4*pi; y=3*sin(5*x);
>> for k=2:length(x)
plot(x(1:k),y(1:k)); pause(0.001);
end
```

2) comet:

It shows an animated graph where the circle-shaped 'comet head' follows the points of y on the screen. The 'comet body' is therefore defined as the trailing portion that comes after the head.

```
>> x=0:pi/10:4*pi; y=3*sin(5*x);
>> comet(x,y)
```

3) animatedline, addpoints…drawnow...(pause)

Together with each other these commands operate: The expression "addpoints(h,x,y)" joins the points defined by x and y to the animated line that h specifies. Then, we use drawnow to show the updates. Previous points are immediately linked to upcoming ones.

```
>> h = animatedline('Marker','o');axis([0,4*pi,-1,1])
>> for k = 1:length(x)
addpoints(h,x(k),y(k));
drawnow limitrate;
end
```

III.2.6 Function derivation

Finding the derivative is simple if the function is recognized. Instead, one can utilize the Matlab function "diff" in real-world scenarios when he has a file of points. Once applied to a (n×1) vector x, this function calculates the differences between neighboring members, yielding a (n-1) ×1 vector. It can be used to calculate a derivate of function f(x): $\dot{f}(x) = \frac{df}{dx}$. Another manner to calculate derivation can be seen in chapter 6.

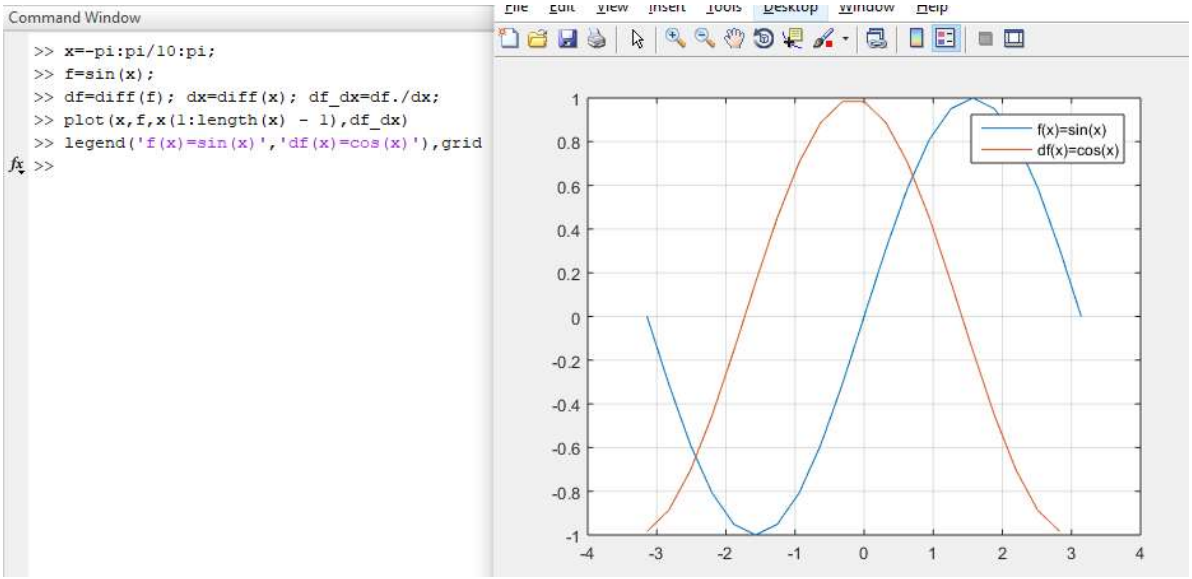


Figure 3.13. Example of calculating a derivate of function

III.3. Plotting in 3D: Plot3, mesh, surf plot:

- Plot3 (X,Y,Z),

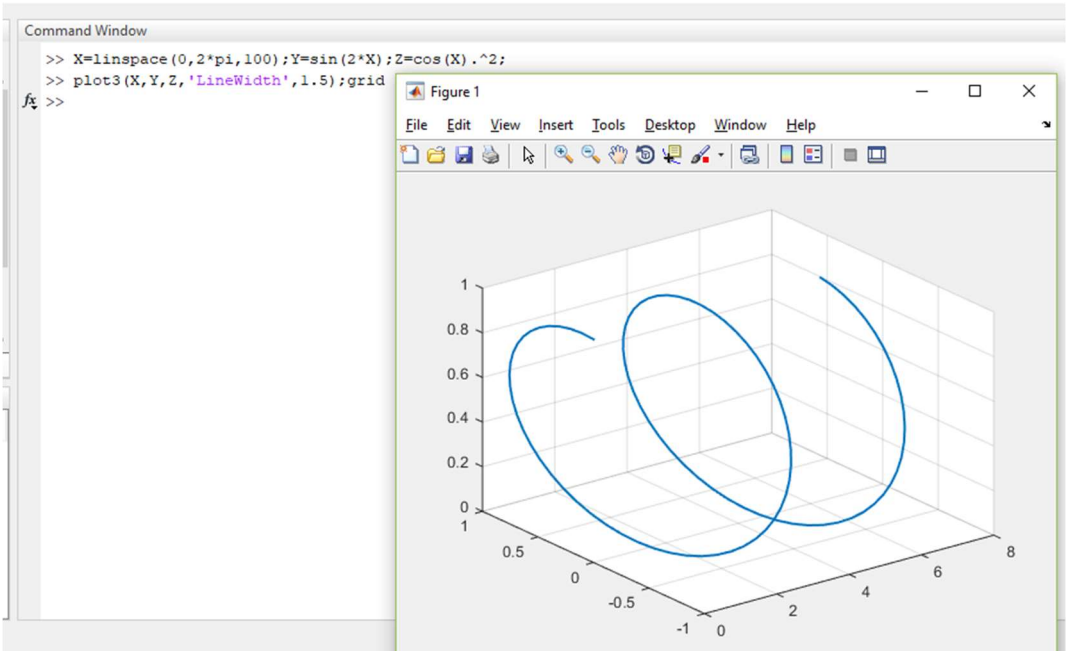


Figure 3.14. Example of using "plot3"

- mesh(Z)→ wireframe mesh with color.

Example:

Let's take x and y as values from $-\pi$ to $+\pi$ with the step$\frac{\pi}{10}$. So we write, (in command window):

>>x=-pi:pi/10:pi; y=x;

Then we create tow matrices X and Y that define domain of calculation of z → meshgrid.

>>[X,Y]=mehgrid(x,y);

Then, we can calculate z and save it on the variable Z:

>> Z=sin(X.*Y)./(X.^2+Y.^2) ;

To plot we use mesh(Z).

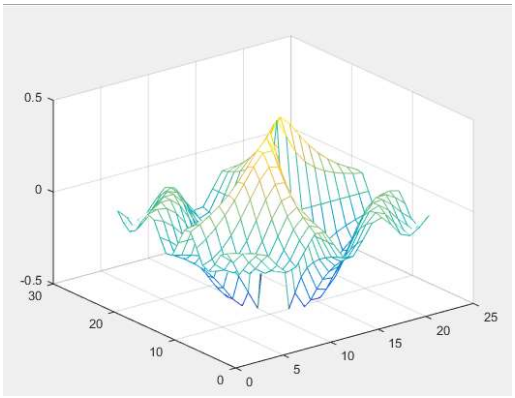


Figure 3.15. Example of using "mesh" command

When we use “surf”, the 3-D surface will be shaded. And “surfc” allows to have the contour plot under.

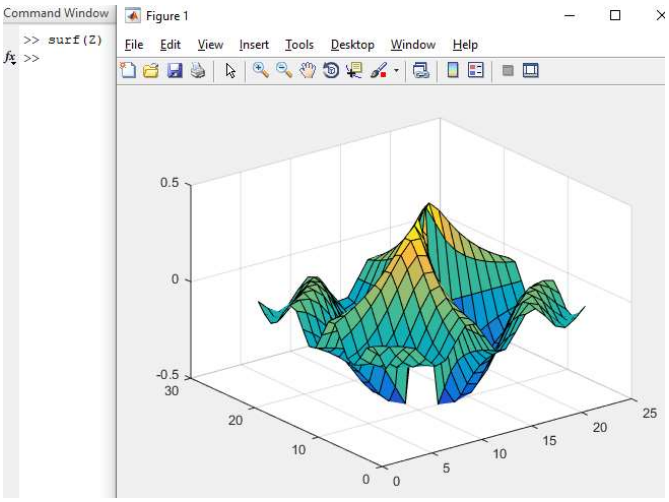


Figure 3.16. Example of using ”surf” function

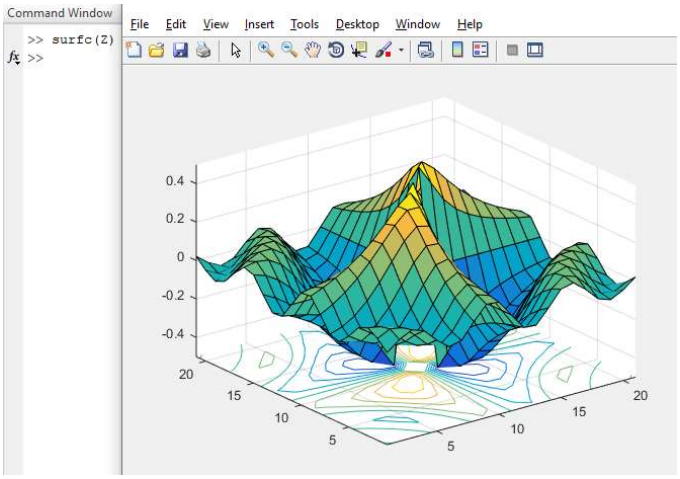


Figure 3.17. Example of using “surfc”

III.4. Saving figures in Matlab

To save figure drawn in Matlab: go to edit. Click on copy Figure

Example:

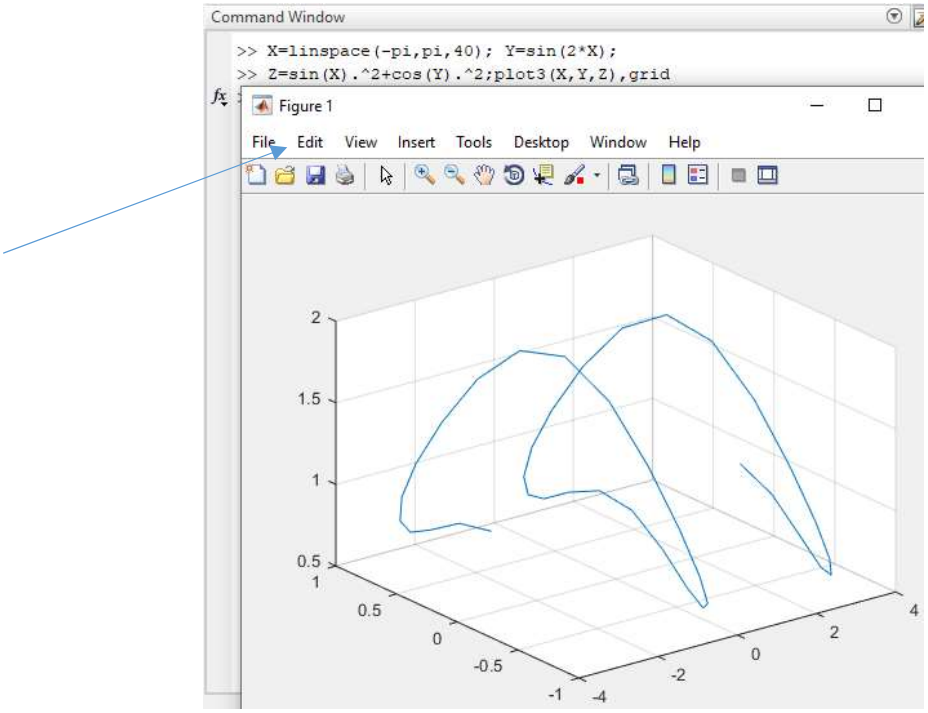


Figure 3.18. Saving figures

Then, in doc file just make past. You get the figure as drawn in Matlab form and characteristics. In your doc file.

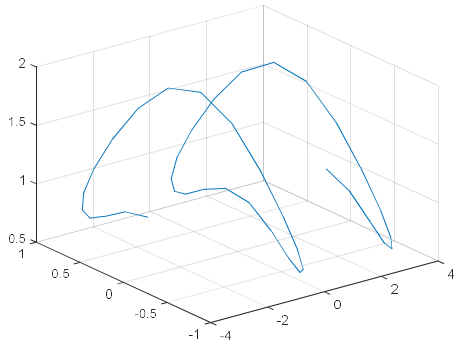


Figure 3.19. The previous saved figure seen out of Matlab

III.5 Conclusion

This chapter concentrated on the Matlab drawing package. We illustrated some commands with examples and details. This chapter's reader can obtain plotting information directly from Matlab. One can draw results without looping several instructions.

Chapter IV: Programming using M-file

IV.1 Framework of program in Matlab:

m-file script/m-file function

IV. 2 Programming with conditions:

if… end, if… else… end,if ... elseif …else… end, Switch case

IV.3 Programming with loops:

for …end , while … end

IV.4 Debugging programs

IV.5 Conclusion

This chapter demonstrates the conventional Matlab programming technique. The program organization, loops, conditions, etc. Debugging can then be used to fix incompyled errors.

IV.1. Introduction

An overview of programming language in Matlab was will be exhibited in this chapter. Student can at the end of this chapter write his own MATLAB m-files or functions. With different kinds of classical commands: affectation, loops, conditions moreover, he can correct hidden errors using debugg mode.

IV.2 Programming in Matlab: m-file script/m-file function

Programming in Matlab uses m-file scripts or m-file functions. It's about putting language code in a file that has the extension ".m" to execute it as needed. Both functions and scripts store variables in work space and variables defined are local to it unless declared as global using the function 'global'.Function. Not as script, function has input arguments and gives output onces.

- The first line in function has always keywords:

```
function [out1,out2,…,outn]=func_name(in1,in2,…,inn)
```

- The second line in function: a commentaries (% definition) which guides the Matlab command 'lookfor'
- Just after one can use commentaries (%definition)→ what can we see using 'help'
- Then body of function. Which takes several commands.

IV. 3 Programming with conditions

IV.3. 1 if …. end

```
if condition (true)
        task
end
```

Condition is a logical variable. To define it, often it has to use to define relational operations (see table below) [1, 2].

Table 4.1. Relational Operations. It returns logical array of Matlab [see help**]**

symbol	<	>	<=	>=	~=	==	&	|	&&	||
meaning	Less than	Greater than	Less or equal	Greater or equal	Not equal	equal	And (element)	Or (element)	And (condition)	Or (condition)

Example

```
x= 3;
if   (x>10)
   disp(x) ,
end
```

In this example the result x is not displayed but if we change x to be x=13 and we repeat the condition, we will get x displayed.

IV.3.2 Condition: if … else …. end

```
if condition (true)
            task1
      else
            task 2
      end
```

Example

```
>> if (x>10) disp('admitted student') ,else disp(' not admitted') ,end
```

In this example, one can change the x values and each time the result flow the condition.

IV.3. 3 Conditions: if elseif else end

```
if condition (true)
            task1
      elseif (cond1 false &cond2 true)
            task 2
      else (cond.1&2 are falses)
```

```
            task3
        end
```

Example

```
%solving a second degree equation ax^2+bx+c=0
delta=(b^2-4*a*c)/(2*a);
 if delta<0
disp('no solution in R')
     elseif (delta==0)
% delta is not <0 and delta is equal to 0
    sol_1=(-b)/(2*a);
    disp('double solution x_0='),sol_1,
else
%delta is not <0 and delta is not equal to 0
sol1=(-b-sqrt(delta))/(2*a);sol2=(-b+sqrt(delta))/(2*a);
     disp('solution1 x_01='),sol1,
     disp('solution2 x_02='),sol2,
 end
```

Complicity sometimes gives way to the use of nested conditions. The program becomes unreadable in addition multiple lines of ends take up space. One can use a Boolean variable in its stead to minimize this. For example, y=(cond)*value is used in place of "if condition then instruction." This means that when the bolean variable "cond" is true, cnd takes the value "1," therefore y=value.

Example 2 using only relational operations

In this example a signal limitation up and down the for loops if for max and min will be remplaced by one line in which it is compared the value of y with max and min

```
% Polting noisy signal with the original With a limit to output.
t=0:100; nois=2*randn(size(t));y=10*sin(0.1*t)+nois;
```

you can add limitation max/min and then y_lim

If y<= y_min ➔ y_lim=y_min

If y>=y_max➔y_lim =y_max

If y >y_min and <y_max ➔ y_lim=y

Using logical operation one can replace the previous three line by one expression:

y_lim=(y>=max).*max+((y<max)&(y>min)).*y+(y<=min).*min;

The expressions : ‘ y<= y_min’, ‘y>=y_max ’ and ‘y >y_min & <y_max’ returen values of ‘0’ or ‘1’

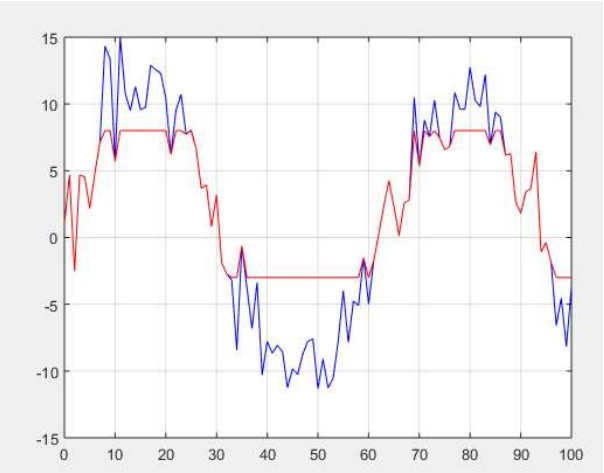


Figure 4.1. Example of Conditions without using “if… end”

IV.3.4 Conditions: switch ….case…otherwise… end

The syntax of this command is:

```
switch (var)
            case var=var1
    task 1
             case var=var2
    task 2
       ..
        ..
    otherwise  task_ default
    end
```

Example 3

The categorization of the student's mention based on his mark X is another illustration of the application of relational relations. Four conditions are produced by this classification, but only one line remains.

```
c=1; %must be in integer;
x=… % student chooses and play with changing values
c=(x<10)*1+((x<12)&(x>=10))*2+((x<16)&(x>=12))*3+((x<20)&(x>=16))*4;
switch (c)
case (1)
disp (‘Not admit’)

case (2)
disp (‘Admit’)
case (3)
disp (‘Admit good’)
case (4)
disp (‘Admit very good’)
otherwise
disp('error')
end
```

IV.4 Programming with loops

IV.4.1 for ….end

```
for vect
    task
        end
```

Example 1

One simple application of loops remains the matrices calculation. The example is just to clarify since one can get, in Matlab, the same result with fewer commands. In this example, we inter

the matrix $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ and we calculate squares of her elements using 'for loop'. One can realize that it can get the same result using M_ sqr =M.^2

```
M=[1,2,3;4,5,6;7,8,9];
[n_row,n_col]=size(M);
 for i=1:n_row
    for j=1:n_col
       M_sqr(i,j)=M(i,j)^2;
    end
end
disp('M square='), M_sqr
```

IV.4.2 Loops: while…. end

```
while condition (true)

        task
                end (false)
```

Example 2

We repeat the previous example of 'for loop' her to make a distinction the difference.

So it appear clearly that in more of for loop, one have to take iterations (i=i+1;j=j+1;) the an end condition at the top.

```
M=[1,2,3;4,5,6;7,8,9]; [n_row,n_col]=size(M);
 i=1;j=1;
while   i<n_row
    while  j<n_col
       M_sqr(i,j)=M(i,j)^2,
       j=j+1;
    end
    i=i+1;
end, disp('M square='), M_sqr
```

Example 3

Write an m-file to display a matrix with dimension (n,m) such that each element on the main diagonal has the value " 2" , each element on the adjacent diagonals -1 , and 0 everywhere else.

Solution

```
for m = 1:m0
   for n = 1:n0
      if m == n
         A(n,m) = 2;            % main diagonal
      elseif abs(m-n) == 1     % adjacent diagonal
         A(n,m) = -1;
      else
         A(n,m) = 0;
      end                    % of if
   end                       % of for n
end                          % of for m
A                            %displaying A
```

IV.4 Debugging programs

Bug means error

Debugging omitting errors

The term is used for errors which appear after execution (value=Inf, nan, complex number not expected results)

Example

```
v1=[0,12,20]
v2=[2,3,14].*v1
V3=[8,17,6]./v1  ➔   Inf  1.4167   0.3000
```

Debugging process

We can debug the M-file using Editor Debuggrer or from command window.

It consists of:

1. Preparing for debugging

   Open the file, be sure the file you run is in the current directory

Figure 4.2-a. check the current directory

2. Setting breakpoints if it is necessary

   Breakpoint shows where the problem occurs.

   Means the debugging mode 'end' of the programme so the intermediates values can be visualized and changed. One can change calculate value in breakpoints. Also it can one or more breakpoints in the program. To create it breakpoints:

- Click in the left-hand column (standard breakpoints) of the file (program)

Figure 4.2-b. Breakpoint setting

After creating breakpoints a red circle appears in the beginning of the line.

In command window the result of run appears "k>>" not ">>".

Figure 4.2-c. Break point result

- Sometimes by running if there is error which produce a specified warning (error breakpoints)
- Under specified condition (conditional breakpoint)

Now, on can change the values and run to detect error…

If we change variable value, the red circle becomes gray.

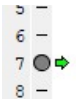


Figure 4.2-d. Break point after changing values

Figure 4.2. Debugging procedure

3. Running M-file with breakpoints
4. Steeping (through an M-file or throw command window)
5. Examining values and correcting problems

   Do not make changes to the M-file in the debug mode. We change values and correct.
6. Ending debugger.
7. Select Exit Debug mode from the debug menu.

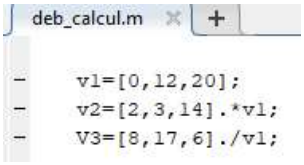
Note

These procedure can be done using command window, some command used are illustrated in the table 4.2.

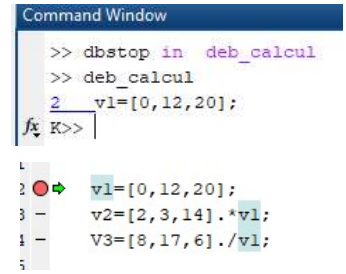
To use them, one have to precise the name of the file (program) which must be in the current directory.

Example 4:

Let's take create a file containing the previous example

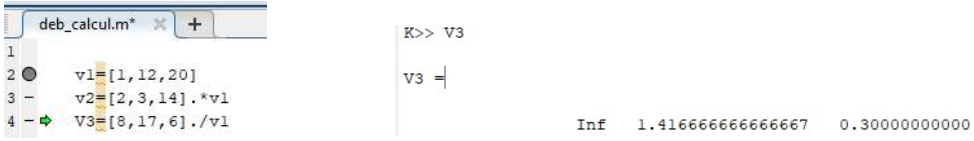


Then in command window, one can create a breakpoint by using 'dbstop' then run (call) the file. It appears a breakpoint in the file:



After changing V1 and using 'step in' which can be run from a specified line. If for this example, you write: K>> dbstep 1:3

You will get



You can conclude that the error is in the first line!

Here to stop debugging: K>> dbquit

Table 4.2. MATLAB functions' to debug

dbstop/dbclear	Set/clear breakpoint
dbclear all	Clear all breakpoint
dbstep in	step in
dbstep	Single step
dbcont	continue
dbquit	Quit debugging
breakpoints	displays the list of breakpoints with values

Example 5:

>> dbstop in funct2 at 4

After running, the prompt in the Command Window changes to

K>>

-Click " Continue" three times to move through the breakpoints at lines 4 to other breakpoint. Now the program is again paused at the breakpoint at line 4.

Type "dbstep" in in the Command Window to step into funt2 into line 4.

-Continue running or step to the next line. If the result is not as you expect, or a previous line, contains an error. Select the Workspace (use dbstack.)

-You can change the value of a variable in the current workspace to see if the new value produces expected results. Then continue running or stepping through the program…

End Debugging.

IV.5 Conclusion

Plotting package of Matlab has been treated in this chapter in details and examples. Then an overview of programming language in Matlab was exhibited. Student can at the end of this chapter write the own MATLAB m-files or functions. With different kinds of classical commands: affectation, as loops, conditions moreover, he can draw results without looping several instructions.

Chapter V: Building Structure & Graphical User Interface “GUI”

V.1 Introduction

V.2Use of structure in Matlab: Example

V.3 Graphical User Interface “GUI” aim and definition

V.4 First step, Components palette, Layout area

V.2 Second step, property inspector

V.3 Last step:,Callbacks

V.4 Examples

V.5 Conclusion

This chapter offers and introduce the graphical programing: Student will learn how to design, modify and test GUI in Matlab.

V.1. Introduction

In general, a program needs to get input from user and gives output. In older versions of Matlab this is got by text based interfaces. This classical manner to program, text based interfaces, is seen in the previous chapter.

Data visualisation or GUI in MATLAB is based on the “Handle” Graphics System in which the objects organised in a Graphics Object Hierarchy can be manipulated by various high and low level commands. Matlab 7 was the first to introduce the "GUI." Then, new items and categories are added to the Handle with every new MATLAB release. … More and more the idea of graphical interface is developing and since R2016, two ways of doing it have been formed: using GUIs and App. Designer [4,5].

In this chapter, we will see only the graphical “GUIs” programming in Matlab. However, before we start, it is better to revise the structure since there are used in graphical programing.

V.2 Use of structure in

Example (student has name, date, mark)

```
>> M01(1,1,1).name='sss';M01(1,1,1).date='18/12/13' ;M01(1,1,1).mark=[14 13 10];
>> M01(1,2,1).name='ttt';M01(1,2,1).date='10/02/13' ;M01(1,2,1).mark=[17 15 12];
>> M01(1,1,2).name='uuu';M01(1,1,2).date='11/11/13' ;M01(1,1,2).mark=[10 14 16];
>> M01(1,2,2).name='vvv';M01(1,2,2).date='03/05/13' ;M01(1,2,2).mark=[16 13 11];
>> M01

M01 =

1x2x2 struct array with fields:

    name
    date
    mark
```

```
>> L01=struct('name',{'ccc','ddd','fff','ggg'},'date',{'16/02/15','09/04/15','02/10/15','29/03/15'},'mark',{[16 11],[13 10],[11 09],
>> L01

L01 =

1x4 struct array with fields:

    name
    date
    mark
```

It can be noticed that we can apply any Matlab command to the field of a structure

```
>> mean([L01.mark])

ans =

   10.8750
```

```
>> mean([L01(1,1).mark])

ans =

   13.5000
```

V.3. Graphical User Interface "GUI" aim and definition

The aim is to create window's screen in order to make the project (the program) easier adjusted and to visualize parameters or results.

V.3.1 First step

>>guide

Or choose new then graphical user interface

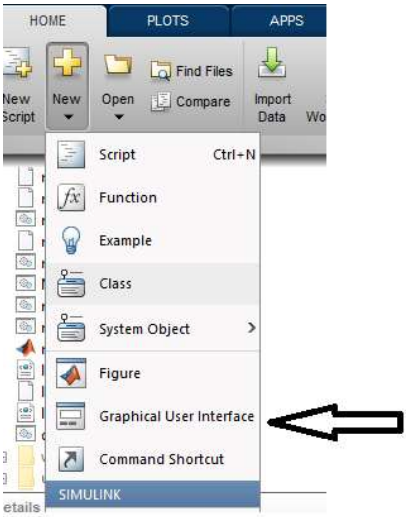


Figure 5.1. Opening graphical user interface without using Command Window

This command creates automatically the following (Guide Quick Start) window and an M-file that controls how the GUI operates.

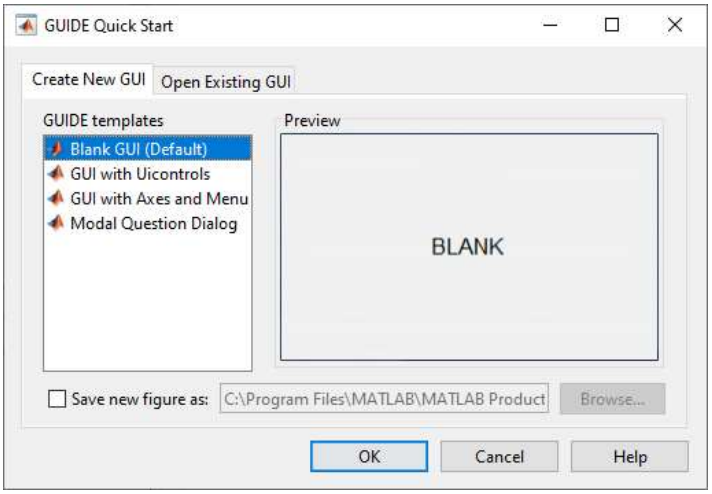


Figure 5.2. Result of opening graphical user interface

By selecting "Blank GUI (Default)", we will have:

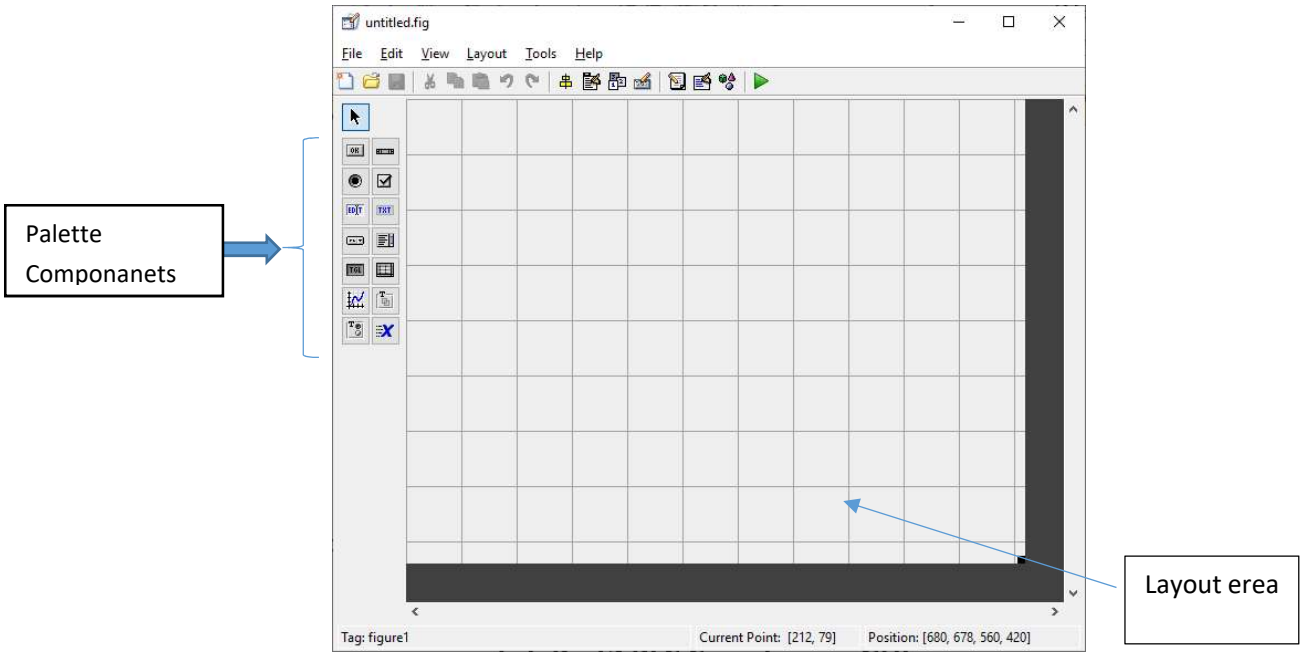


Figure 5.3. Result of click on "Blank GUI (Default)"

V.3.1.1. Layout area

In a way, it defines the body of the future window, and this is the area in which we place the components that we have chosen (from the Components palette). mentioned that dragging the right down coin will resize it.

V.3.1.2 Palette components

Display names of component that can be used in the GUI window witch we want to create.

V.3.2 Second step

In this step, we select the needed components to the layout area, we adjust them by using property inspector. The latter, can be selected from: view menu or by click in the component in the layout area.

Example

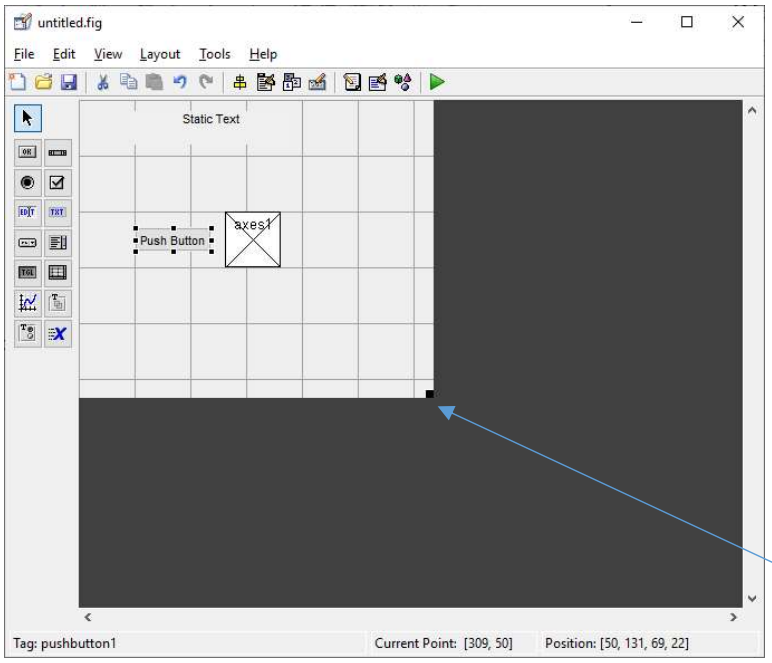


Figure 5.4. Example of first step to create GUI

In this example, we select three component: static texts, push button and axes. We reduce the size of the window.

The push button now is selected if we go to property inspectors from view we will deal with its property not the other components' property.

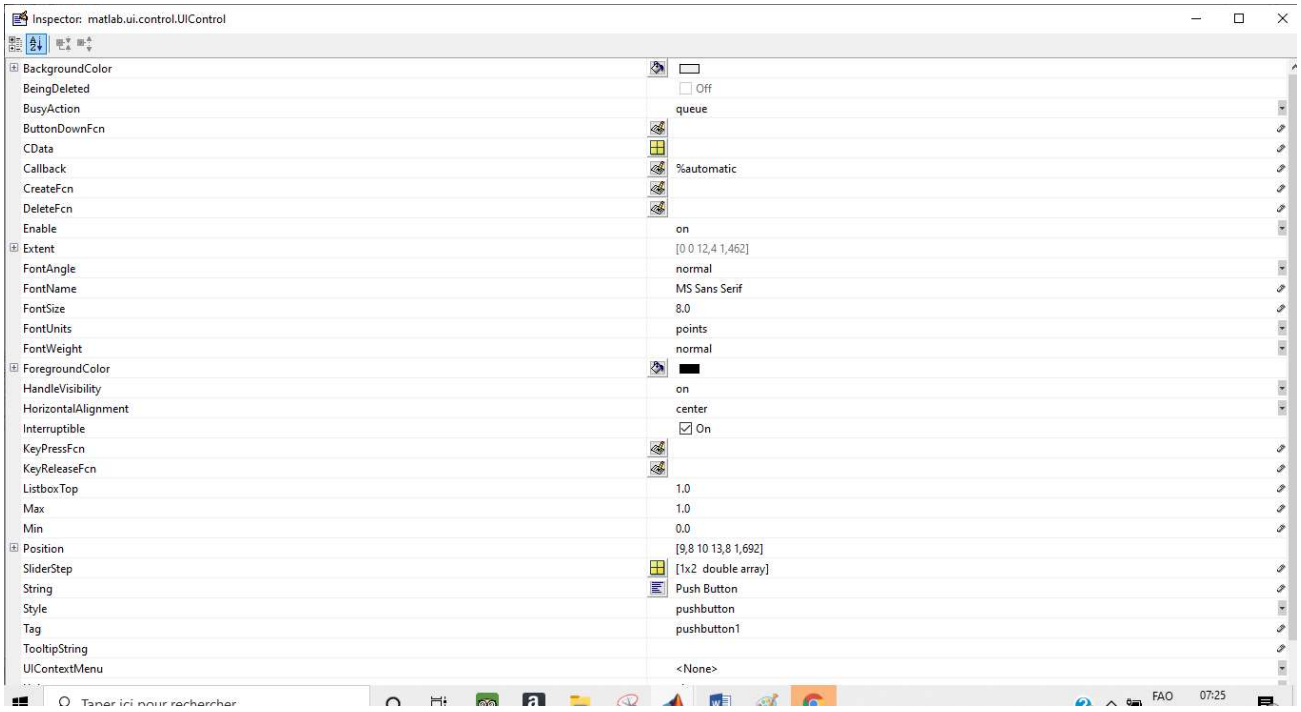


Figure 5.5. Property inspectors for push button

In this example we change string property to “Draw” and “FontSize” to 12.

Then we select (one click on) the static text component, we change the string property to “first example”, FontSize to 16 and “ForegroundColor” to blue we get:

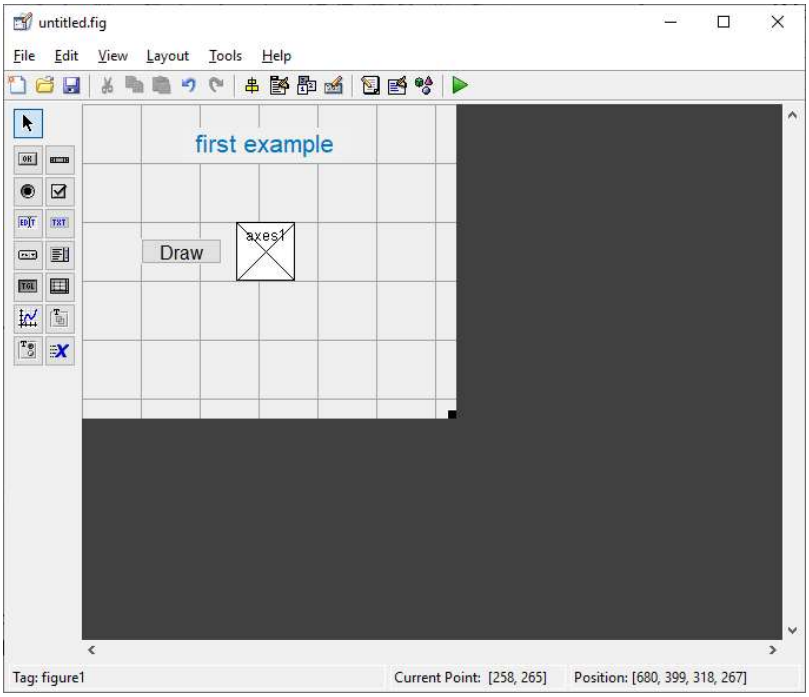


Figure 5.6. The first example figure seen throw Command Window

The tag property is as string priority: we can write a text. But in tag names the component in the program (callback) and string name it in the future window.

If in this example we change tag propriety of push button to draw, in the program the name changes.

```
% --- Executes on button press in Draw.
function Draw_Callback(hObject, eventdata, handles)
% hObject    handle to Draw (see GCBO)
```

Figure 5.7. The first example “Callbacks” from the “push button” part

V.3.3 Last step

The window now is created we have to “program it” in response to some actions of the user (click on for example push button). The tool is the “callbacks” that we found in the M_file created.

This M_file can be seen in 2 manners:

- Go to editor select menu. It will ask to name the program. After saving the name it appears the callbacks' M_file.

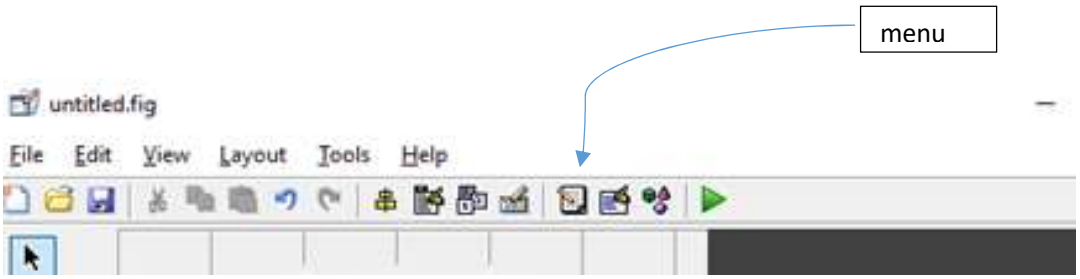


Figure 5.8. How to get menu from editor

- We run the program the first time, the callbacks' M_file appears automatically

V.3.4 Examples

Example 1 (Axis & push button components)

We work on the previous example. We name the program "exp1". The callbacks is such that:

exp1.m

```
function varargout = exp1(varargin)
% EXP1 MATLAB code for exp1.fig
%      EXP1, by itself, creates a new EXP1 or raises the existing
%      singleton*.
%
%      H = EXP1 returns the handle to a new EXP1 or the handle to
%      the existing singleton*.
%
%      EXP1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in EXP1.M with the given input arguments.
%
%      EXP1('Property','Value',...) creates a new EXP1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before exp1_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to exp1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
```

.....

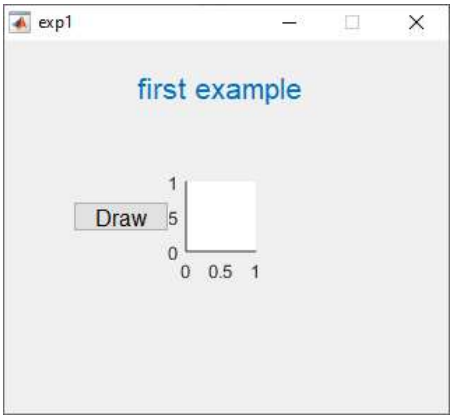
Figure 5.9. The "Callbacks" ot he second example

If we want to see a sine plot after a right click on the push button (to have the callback part of push button) we write the command on the push button part in the callbacks script.

```
% --- Executes on button press in Draw.
function Draw_Callback(hObject, eventdata, handles)
% hObject    handle to Draw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1)
t=0:0.01:2*pi;
plot(t,sin(t));
```

Figure 5.10. Example of writing commands in Callbacks

After run we get:



Then when we click on "Draw" we get

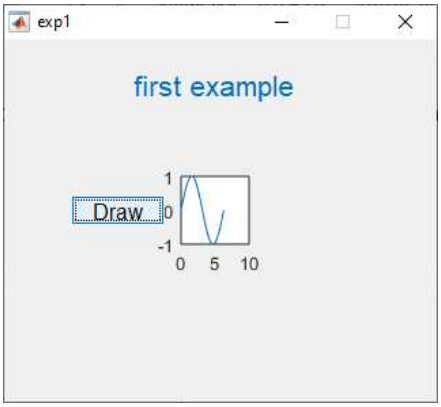


Figure 5.11. Result of the previous example

Example2 (Edit Text to calculate)

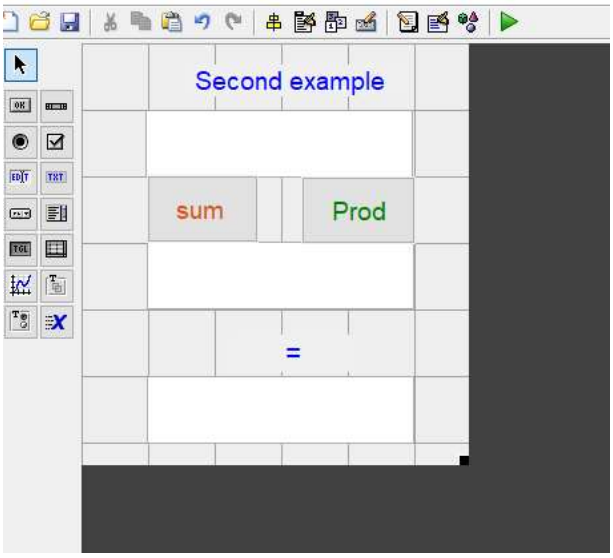


Figure 5.12. Result of example 2

To get this figure, we adjust companent caracteristics in proprty inspucture as it have been indiucated in example1 and in the manner of figure (Fig4.12). Then, in callback of sum we write:

```
function operate_Callback(hObject, eventdata, handles)
% hObject    handle to operate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
x1=str2num(get(handles.edit1,'string'));
x2=str2num(get(handles.edit2,'string'));
x3=x1+x2;
set(handles.edit3,'string',num2str(x3));
```

In callback of prod :

```
function push_Callback(hObject, eventdata, handles)
% hObject    handle to push2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
x1=str2num(get(handles.edit1,'string'));
x2=str2num(get(handles.edit2,'string'));
x3=x1*x2;
set(handles.edit3,'string',num2str(x3));
```

Example3 (Pop-Menu & Table)

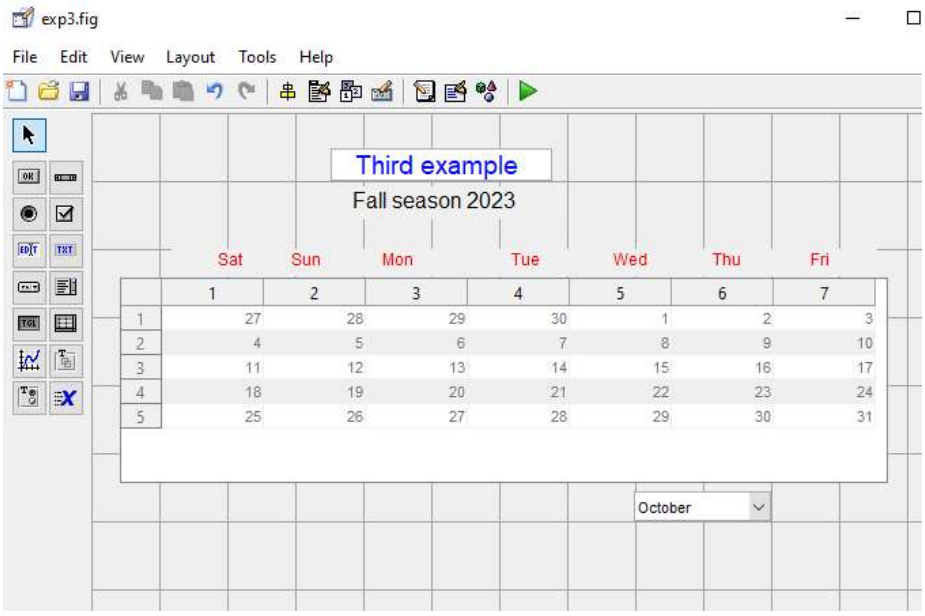


Figure 5.13. Result of example 3

In callback of Pop-up-Menu we write:

```
ch=get(handles.popupmenu1,'value');

switch ch
case 1 % User selects October.
oct[24:30;1:7;8:14;15:21;22:28;29:31,1:4];
set(handles.days,'data',oct)
case 2 % User selects November.
nov=[29:31,1:4;5:11;12:18;19:25;26:30,1,2]
set(handles.days,'data',nov)
case 3 % User selects December.
dec=[27:30,1:3;4:10;11:17;18:24;25:31];
set(handles.days,'data',dec)

   otherwise
   set(handles.days,'data',zeros(5,7))

end
```

V.4 Conclusion

Graphical User Interface are explained with simple examples in this chapter. It is showed how to catch the correct component and adjust it, the how use handles and to program a small tasks.

Chapter VI: Polynomial & Curve fitting

VI.1 Introduction

VI.2 Polynomials in Matlab

VI.2.1 Roots and poly

VI.2.2 Evaluation

VI.2.3 polynomial arithmetic: addition, subtraction, multiplication & division

VI.2.4 Differentiation & integration

VI.3 Curve fitting

VI.4 Conclusion

One of the most important applications with real data is curve fitting. We will not discuss all curve fitting in this chapter, since there are blocks in the menu of Matlab with several functions. We introduce the principle in command window level.

VI.1. Introduction

Since polynomial is important and it appear in several situations, it is shown more easily in Matlab: this chapter discusses polynomial calculations, differentiations, and finding zeroes. Polynomials are later used to interpolate data and fit curves [1, 3, 6, 7].

VI.2. Polynomials in Matlab

A polynomial is a mathematical function defined with polynomic form.

VI.2.1. Roots and poly

$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots . a_1 x^1 + a_0$ in Matlab to define it use: >> P=[an, …..,a1,a0]

Then

k=roots(p) %roots of the polynomial defined by p

p=poly(k) % coefficients vectors of the polynomial defined by the roots in k

Example

$p(x) = x^4 - 13x^3 + 59x^2 - 107x + 60 = (x-1)(x-4)(x-3)(x-5)$

p=poly(k) ➔ [1,-13,59 ,-107, 60]

k=roots(p) ➔ [1;4;3;5]

VI.2.2. Evaluation

P is a polynomial defined by its coefficients.

We compute a polynomial's value at x polynomials in Matlab use: Polyval(p,x)

Example

```
>> p=[1 -2 1] , x=3
>> y=polyval(p,x)
y =     4
>> x=0:6; y=polyval(p,x)
y =     1     0     1     4     9    16    25
```

In this example the polynomial function $p(x) = x^2 - 2x + 1$ is introduced then it is evaluated for a scalaire vartiable x. it gives y=p(3).

The same command 'polyval(p,x)' can be used to calculate several data by introducing x as a vector. It will self-looped to give result as a vector.

VI.2.3. Polynomial arithmetic

Let's take: $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots . a_1 x^1 + a_0$

$k(x) = b_n x^n + b_{n-1}x^{n-1} + \cdots . b_1 x^1 + b_0$

We know that

$p(x) \pm k(x) = (a_n \pm b_n)x^n + (a_{n-1} \pm b_{n-1})x^{n-1} + \cdots . (a_1 \pm b_1)x^1 + (a_0 \pm b_0)$

$p(x) * k(x) = (a_n * b_n)x^{2n} + (a_{n-1} * b_n)x^{2n-1} + \cdots + (a_0 * b_n)x^n + (a_n *$

$b_{n-1})x^{2n-1} + \cdots \quad \ldots + (a_n * b_0)x^n + (a_{n-1} * b_0)x^{n-1} + \cdots (a_0 * b_0)$

In matlab, we define p and k then one can write polynomial multiplication by using the function 'conv' and polynomial division by using 'deconv'. The summation and subtraction are given by '+' and '-'.

$p(x) \pm k(x)$ ➔ p±k , $p(x) * k(x)$ ➔ conv(p,k), $\frac{p(x)}{k(x)}$ ➔ $deconv(p,k)$

Example

P=[1 0 -1]; k=[1 -5 4];

Command Window

```
>> pk_sm=p+k

pk_sm =

     2    -5     3

>> pk_sub=p-k

pk_sub =

     0     5    -5

>> pk_prd=conv(p,k)

pk_prd =

     1    -5     3     5    -4

>> pk_prd_k=deconv(pk_prd,k)

pk_prd_k =

     1     0    -1

>> pk_prd_p=deconv(pk_prd,p)

pk_prd_p =

     1    -5     4
```

Figure 6.1. Result example of polynomial arithmetic

VI.2.4. Differentiation & integration

The derivative of the polynomial p, one can use the Matlab function ‘polyder’ and the integral of polynomial p by ‘polyin’

Example

p =[1 0 -1]

>> dp=polyder(p)

dp =

  2 0

>> int_dp=polyint(dp,-1)

int_dp =

  1 0 -1

In this example, you can see with a simple case, the use of polyder and polyint: the polynomial $p(x) = x^2 - 1$ is derivated to give $dp(x) = 2x + 0$ and integrated to give:

$int\_p(x) = x^3 - 1$.

Example: Function derivation

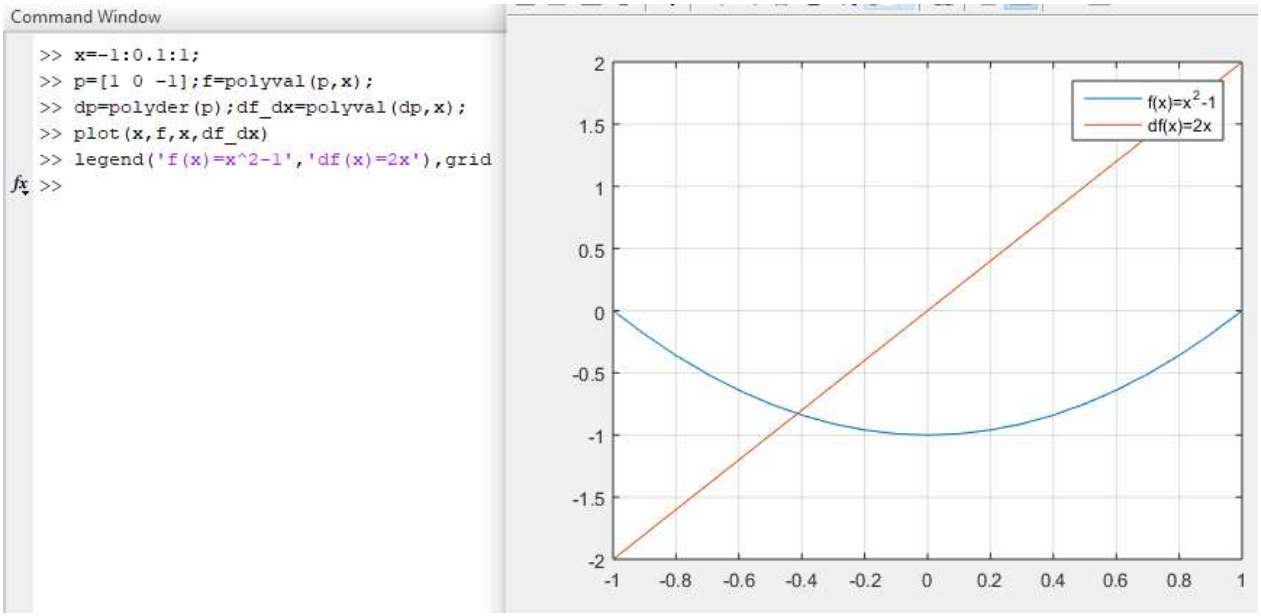


Figure 6.2. Example of calculating a derivate of function

This example shows the use of polyder to calculate and plot the derivation of a function.

VI.3. Curve fitting

Given data X=[……]

Y=[…...]

P=Polyfit(X,Y,n) %n is the order of the fitted polynomial

Means looking for coefficients of the polynomial p(x) that best fits to data in y

Example 1

The aim at this point, that the model is supposed unknown and then it will be fitted to a polynomial of degree2. Let's lake x=0:0.4:8;Y=3*x.^3-6.*x+5; this function is supposed unknown just to generate y. We use the x and y vectors in the function polyfit. This gave the coefficients of the fitted polynomial then we evaluate it for x. the figure (fig6.3) shows y with red and the plot of the fitted polynomials with black.

```
p=polyfit(x,y,2);
      plot(x,Y,'r-',x,polyval(p,x),'ko'),grid
y2=p(1)*x.^2+p(2)*x+p(3);
y1=polyval(p,x);
```

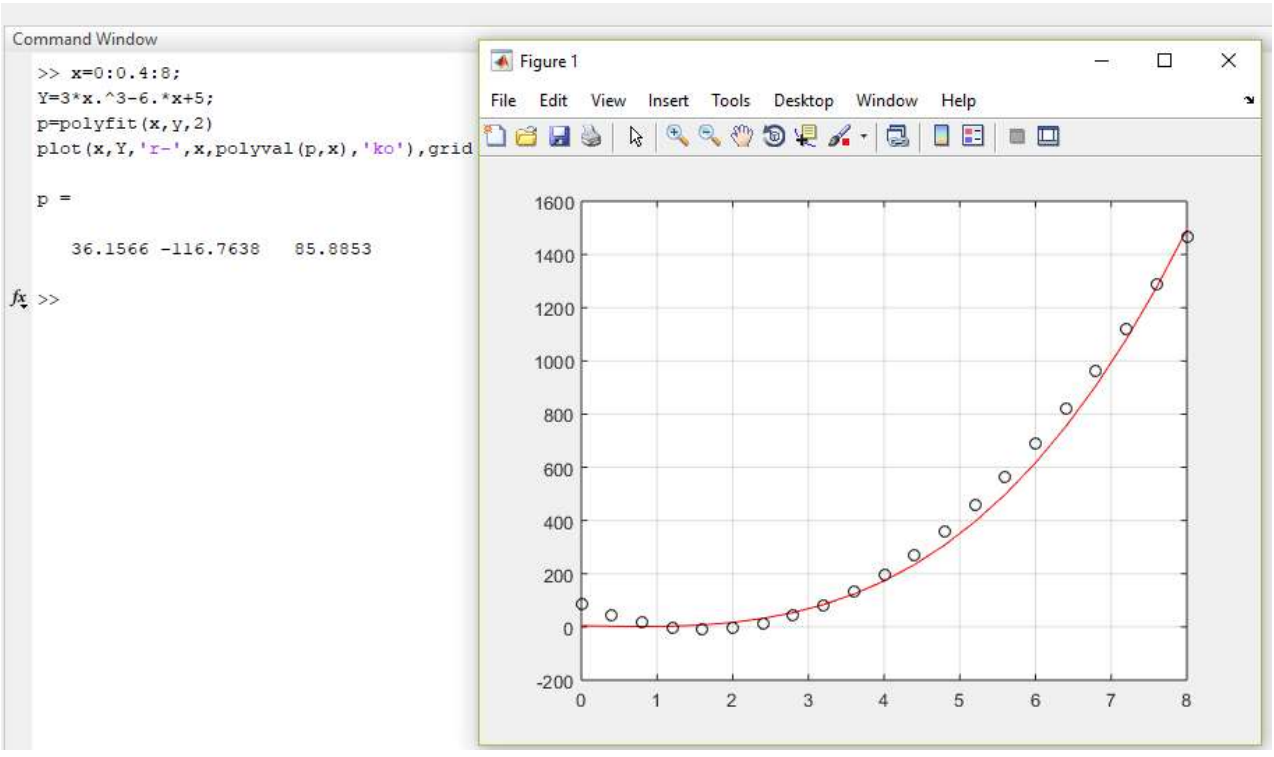


Figure 6.3. Example result of function "polyfit"

Example 2

Given x =[-3 9 41 93] and t=[0 1 2 3]

1. Find the equation which permit to calculate the change in position for a one-dimensional motion (Hint: $x(t) = at^2 + v\,t + x_0$)
2. What will be the position at t=6 (s)
3. At what time it arrive to x=170(m)

Solution

1. We know that is of degree 2. We use "polyfit" to identify a , v and x_0

```
>> p=polyfit(t,x,2)
p = 10.0000    2.0000   -3.0000
```

2. In this question just we evaluate p at t=6: >> x6=polyval(p,6) %= 369(m)
3. We interpolate graphically or by reading values near 170)➔ =4.01(s).

VI.4 Conclusion

In this chapter, polynomial calculations is introduced. One can get to evaluate a polynomial function, to calculate, differentiate or integrate polynomials; in more, one can use to find their zeroes. Following that, data will be interpolated (to estimate the intermediate values) using polynomials.

Chapter VII: Control System Toolbox

VII.1 Introduction

VI.2 Building Models for LTI:
Continuous time model: Transfer function, Continuous time model: State-space, Discrete time Transfer function/ State-space, Conversion between Transfer function / State-space, Combining LTI Models

VII.3 Response Analysis:

Transient Response Analysis, Frequency Response Analysis

VII.4 Introduction to control designing

VII.5 conclusion

The use of 'Control System Toolbox' simplify techniques of linear system analysis. Student can easily analyse and design control systems. This chapter includes both classical and state-space models design methods. We get also, root-locus plots, frequency-response analysis, and system performance. At the end, It will be introduced the proportional-integral-derivative control, and pole placement design.

VII.1. Introduction

Control System toolbox consists of tools specifically developed for control applications. It provides algorithms for systematically analysing, designing linear control systems. One can specify system as a transfer function, state-space, zero-pole-gain; Analyse model using step response plot and Bode plot. ie: visualize system behaviour in time / frequency domain. One can design compensator using automatic PID controller tuning, LQR/LQG design …etc. In this chapter, we limit to introducing the most important commands used in the Control System toolbox [2, 8].

VII.2. Commands used to build Models for LTI

In this part emphasis on the Matlab commands, that allows the open loop study for a given output of the system. First, you must have the transfer function then, the time/frequency responses.

VII.2. 1. Commands used to get continuous time model: Transfer function

To define a transfer function with Matlab:

$$\frac{y(s)}{u(s)} = \frac{a_m s^m + a_{m-1} s^{m-1} + \cdots . a_1 x s^1 + a_0}{b_n s^n + b_{n-1} s^{n-1} + \cdots . b_1 s^1 + b_0}$$

- We use two polynomials. The first for nominator the second for denominator:

$$num = [a_m \quad a_{m-1} \quad \cdots \quad a_1 \quad a_0], \quad den = [b_n \quad b_{n-1} \quad \cdots \quad b_1 \quad b_0]$$

  - Then we use the Matlab function "tf". This function allows you to create a model (see workspace) from its numerator and its denominator.

```
>> num=[1 -6];den=[1 4 1 -6]; h=tf(num,den)

h =

        s - 6
  -------------------
  s^3 + 4 s^2 + s - 6

Continuous-time transfer function.
```

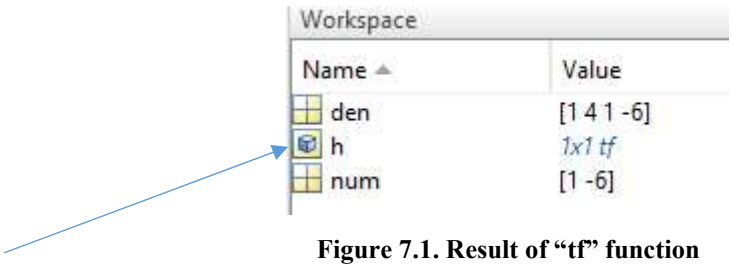


Figure 7.1. Result of "tf" function

- If the transfer function is defined by poles and zeros $\frac{y(s)}{u(s)} = \frac{k(s-p_1)(s-p_2)\cdots(s-p_m)}{(s-z_1)(s-z_2)\cdots(s-z_n)}$, we add the gain factor "k" and we use also two polynomials but **in the roots form**.

The Matlab function "zpk" is defined using the first vector for poles and the second vector for zeros..

- If the transfer function contains a delay $\frac{y(s)}{u(s)} = e^{-val\,s}\ \frac{a_m s^m + a_{m-1}s^{m-1} + \cdots a_1 x s^1 + a_0}{b_n s^n + b_{n-1}s^{n-1} + \cdots b_1 s^1 + b_0}$, we use tf(…,'inputdelay',val)

Examples:

```
Command Window
>> num=[1   -6];den=[ 1  4   1   -6];tf(num,den)

ans =

         s - 6
  -------------------
  s^3 + 4 s^2 + s - 6

Continuous-time transfer function.

>> p=[1;-3;-2];z=[6]; k=1;  zpk(z,p,k) % p=roots(num),z=roots(den),

ans =

        (s-6)
  -----------------
  (s-1) (s+2) (s+3)

Continuous-time zero/pole/gain model.

>> num=[1   -6];den=[ 1  4   1   -6];val=2;tf(num,den,'inputdelay',val)

ans =

                    s - 6
  exp(-2*s) * -------------------
              s^3 + 4 s^2 + s - 6

Continuous-time transfer function.

>>
```

Figure 7.2. Example results of "zpk" and "tf(..'inputdelay'..)" functions

The examples in figure6.2 show different application of 'tf'. In the first case appears transfer function of a system given by the polynomial formulat. Now, if roots and zeros give the 'tf', we are not obliged to convert. One can use the command 'zpk'. Actually systems have a delay to start, if in our mobilization it is taken and not neglected, this can be simulated using the label 'inputdelay' as in the third case.

VIII.2.2. Commands used to get continuous time Transfer State-space

To define the state space

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

The Matlab function "ss" builds an object SYS to represent the state-space model in continuous time.

To see state space we need to define A, B, C, D matrices.

Sys1 = ss(A,B,C,D); (see figure 7.3 –a) creates a state-space model object (see figure 7.3-b) representing the continuous-time state-space model.

It can also converts a dynamic system model sys to state-space form. The output Sys1 is an equivalent state-space model (state-space realization).

If the mode is an identified model represented by an input-output equation of the form

$\{y = Gu(t) + He(t)$

where u(t) is the set of measured input and e(t) represents the noise

On can use:

Sys1 = (sys, 'measured') to convert the measured component into the state-space form.

Sys1= ss(sys, 'noise') to convert the noise component into the state space.

To label y, u and x vectors, on can use: printsys(A,B,C,D,ULABELS,YLABELS,XLABELS). (See figure 7.4)

Example:

```
Command Window
>> A=[2 6; 8 3];B=[3; 4];C=[0 3];D=0;sys1=ss(A,B,C,D)

sys1 =

  a =
       x1  x2
   x1   2   6
   x2   8   3

  b =
       u1
   x1   3
   x2   4

  c =
       x1  x2
   y1   0   3

  d =
       u1
   y1   0

Continuous-time state-space model.
```

Figure 7.3-a. Example results of stat space functions: command window

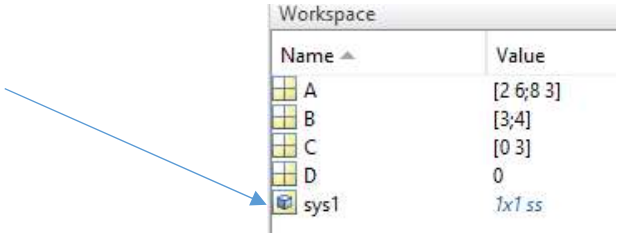


Figure 7.3-b. Example results of stat space functions: workspace

```
Command Window
>> A=[2 6;7 2];B=[4;0];C=[0 3];D=0;u_labl=['input volt'];y_labl=['speed'];x_lab=['x_1 x_2'];
>> printsys(A,B,C,D,u_labl,y_labl,x_lab)

a =
                x_1          x_2
       x_1      2.00000      6.00000
       x_2      7.00000      2.00000


b =
              input
       x_1    4.00000
       x_2          0


c =
                x_1          x_2
     speed        0      3.00000


d =
              input
     speed        0
```

Figure 7.4. Example of adding annotation in stat space

VII.2. 3 Discrete time Transfer function/ State-space

To create a discrete time Transfer function

$$H(z) = \frac{y(z)}{u(z)} = \frac{a_m z^m + a_{m-1} z^{m-1} + \cdots . a_1 z^1 + a_0}{b_n z^n + b_{n-1} z^{n-1} + \cdots . b_1 z^1 + b_0}$$

Or a discrete time State-space

$$\begin{cases} x[n+1] = Ax[n] + Bu[n] \\ y[n+1] = Cx[n] + Du[n] \end{cases}$$

Just we add the sampling time

- tf(num,den,TS)
- ss(A,B,C,D,TS)

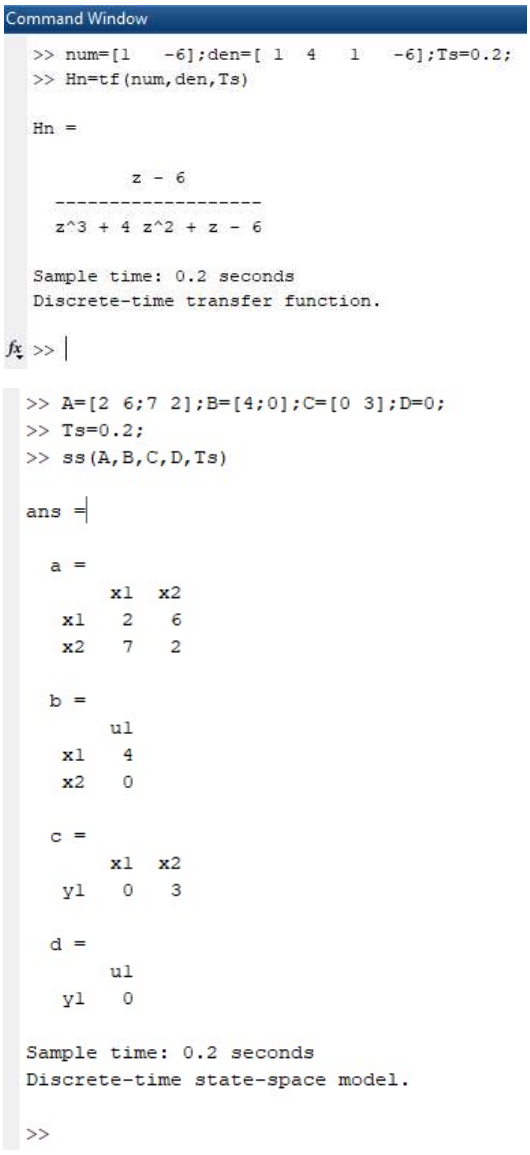


Figure7.5. Example of Discrete time Transfer function/ State-space

VII.2. 4. Conversion Transfer function/ State space

>>[num,den]=ss2tf(A,B,C,D)%state space to transfer function

>> [z,p,k]=ss2zp(A,B,C,D)%state space to zero-pole-gain

One can do the inverse:

>> [A,B,C,D]=tf2ss(num,den) %transfer function to state space

>> [A,B,C,D]=zp2ss(z,p,k) % zero-pole-gain to state space

Or also do:

>>[num,den]=zp2tf(z,p,k) %zero-pole-gain to transfer function

>> [z,p,k]=tf2zp (num,den) %transfer function to zero-pole-gain

VII.2.5 Combining LTI Models

Sometimes, model are given with a block diagram. In order to get the open / closed loop model, we have to simplify and reduce blocks. Using Control System toolbox and when the model is seen as a block with inputs and outputs (block diagram), containing a transfer function or state-space model inside.

There are in Matlab "functions" that can be used to perform basic diagrams and connect 'a model object' such asin figure 7.6.

In the left part of figure (fig6.6-a) one can see the basic cases SISO of two blocs connected in series, in parallel an in feedback. The resulting model will take the transfer function $G_1 * G_2$, $G_1 + G_2$ and feed back of G_1, G_2 respectively.

Notes:

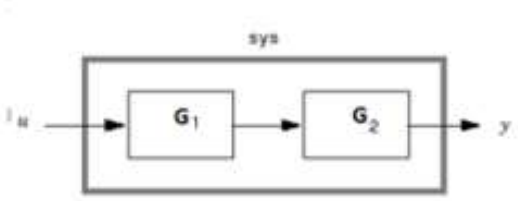
We can use also arithmetic operations such that:

- Addition sys= G1+G2
- Multiplication sys= G1*G2
- Inversion sys= inv(G)

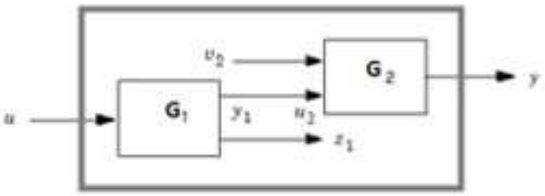
Example:

To illustrate let's take: $G_1 = \frac{2}{s+1}, G_2 = \frac{1}{s+5}$

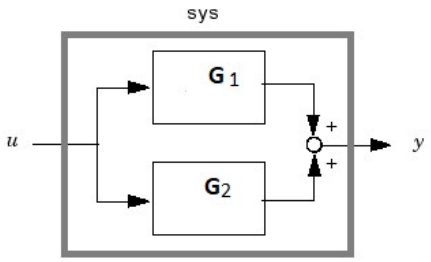
-



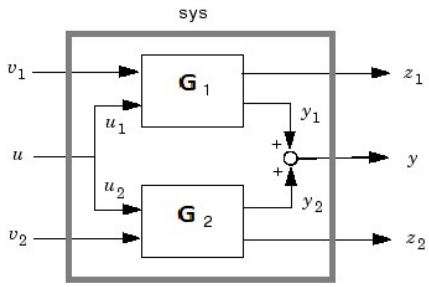
>>Sys=series(G1,G2)



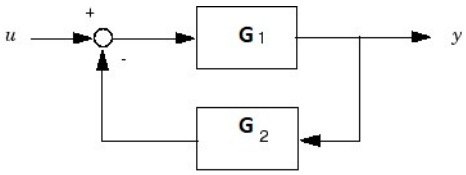
>>Sys = series(G1,G2, 1, 2)



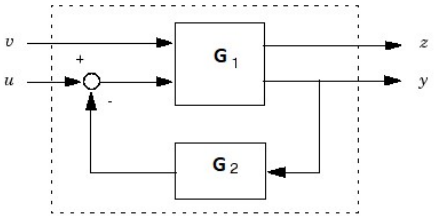
>>Sys=parallel(G1,G2)



>>Sys= parallel(G1,G2,[2 1],[2 1])



>>Sys=feedback(G1,G2)



>>Sys = feedback(sys1,sys2,feedin,feedout,+1)

Figure 7.6-a. Blocks SISO

Figure 7.6-b. Blocks with MIMO

Figure 7.6. Blocks with functions under

```
>> Gls2=series(Gl,G2)

Gls2 =

        2
  -------------
  s^2 + 6 s + 5

Continuous-time transfer function.

>>
>> Glp2=parallel(Gl,G2)

Glp2 =

    3 s + 11
  -------------
  s^2 + 6 s + 5

Continuous-time transfer function.
```

```
>> Glfb2=feedback(Gl,G2)

Glfb2 =

    2 s + 10
  -------------
  s^2 + 6 s + 7

Continuous-time transfer function.
```

Figure 7.7 Example of blocks

In case of multi-inputs (see fig 7.6-b), sometimes not all inputs go to the symmetric destination, with the functions series parallel feedback one can fellow each output to go to the correct input as in his actual model.

In the example fig7.6-b when is connected in series, only the output y1 goes to the model of G2 and from the second input so you have to precise that 'the outpu1 goes to input2). For the parallel case,[2 1] means that the second input/output of the first system an de first input/output of the second system are only connected to resulting system input/output u. In the case of feedback, with the same manner, we can conclude that 'feedout' takes value of 2.

VII.3. Response analysis

VII.3.1. Transient Response analysis

Transient response means the process generated in going from the initial state to the final state It is used to see the time domain characteristics of dynamic systems.

Matlab functions used are mainly:

- impulse response
- Step response

Example

This example shows the impulse response for a system defined with the transfer function $H = \frac{s-6}{s^3+4s^2+s-6}$ .Time t is defined for an interval of 10 s.

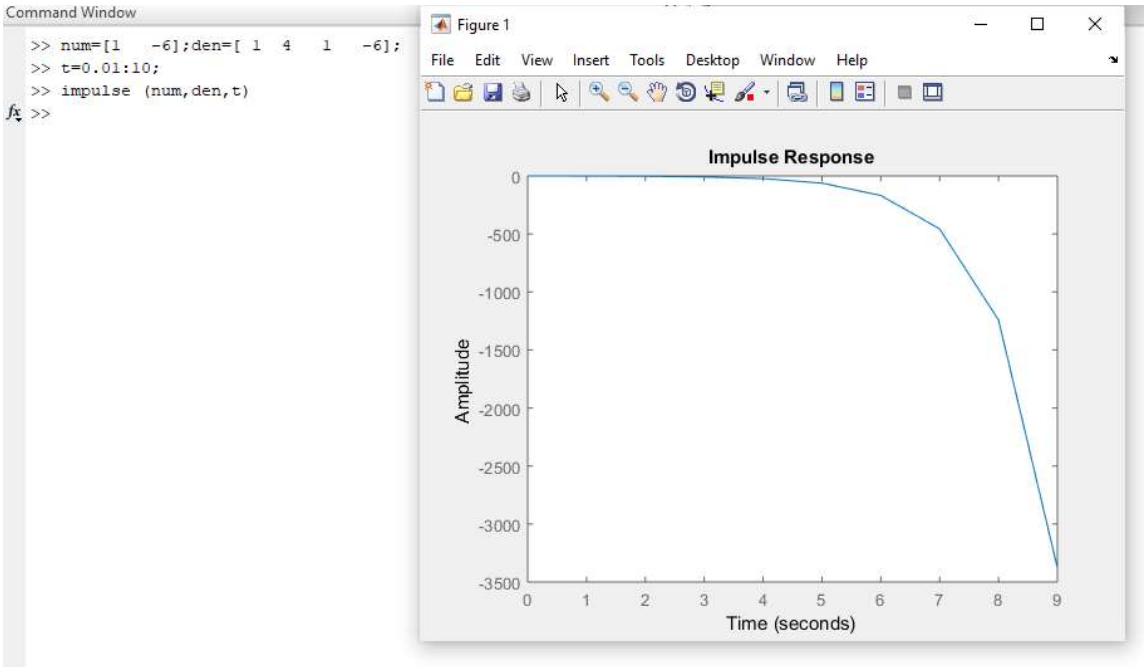


Figure 7.8. Example of impulse response

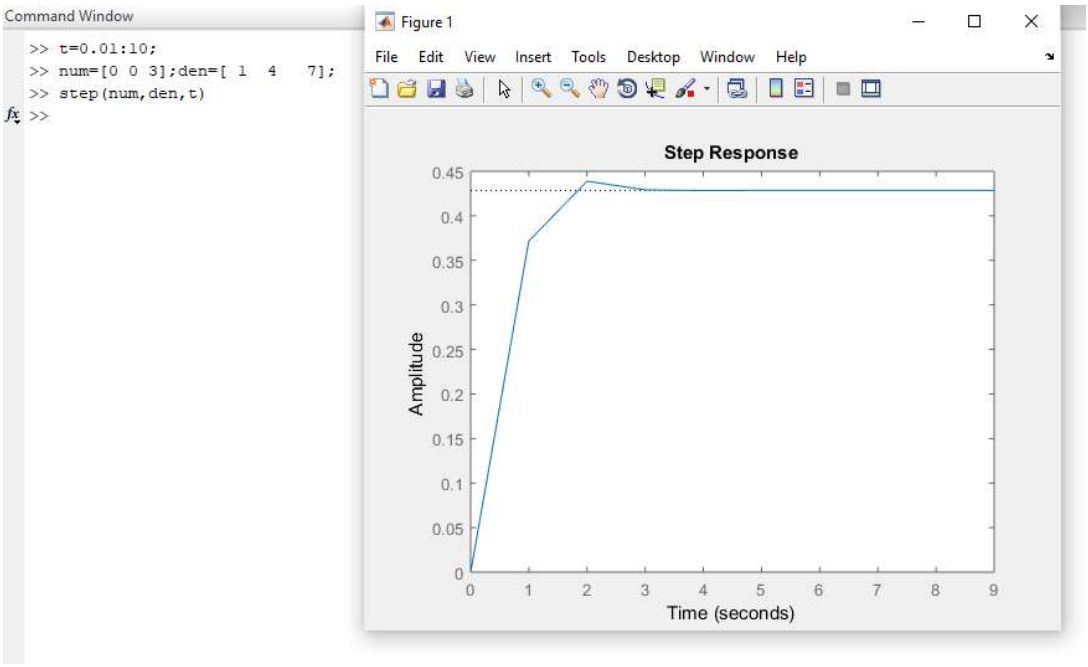


Figure 7.9. Example of step response

Another example to see the step response for the system

$$H = \frac{3}{s^3 + 4s + 7}$$

VII.3. 2. Frequency Response Analysis

A frequency-response model → frequency points with corresponding complex frequency response data obtained either through simulations or experimentally. One can use the data to create a frequency response model:

- bode diagram→>>bode(num,den)
- Nyquist plot→>>Nyquist(num,den)
- Nichols plot → >> Nichols(num,den)

The use of bode(sys1,sys2,...,sysN), nyquist(sys1,sys2,...,sysN) , nichols(sys1,sys2,...,sysN) superimposes the Nyquist plots of several LTI models.

Examples

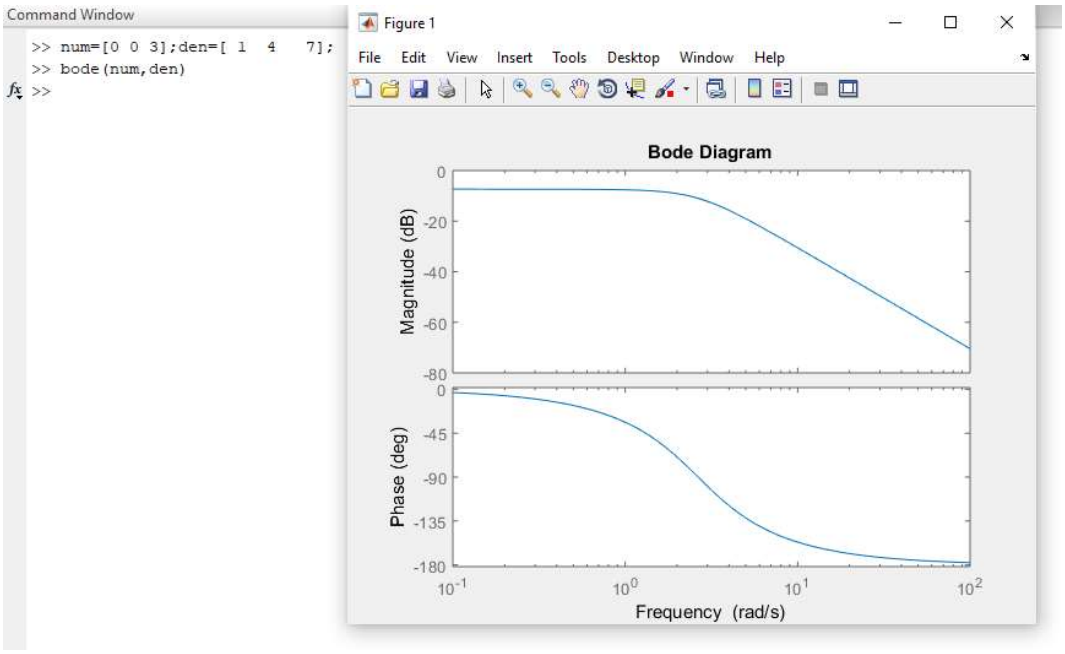


Figure 7.10. Example of Bode function

The Nyquist plots are used to analyze system properties including gain margin, phase margin, and stability when appealed without left-hand arguments. The nyquist function has support for M-circles, which are the contours of the constant closed-loop magnitude. M-circles are defined as the locus of complex numbers and G is the collection of complex numbers that satisfy the constant magnitude requirement.The matlab command 'nyquist' produces a Nyquist plot on the screen. An example is in figure 6.10.

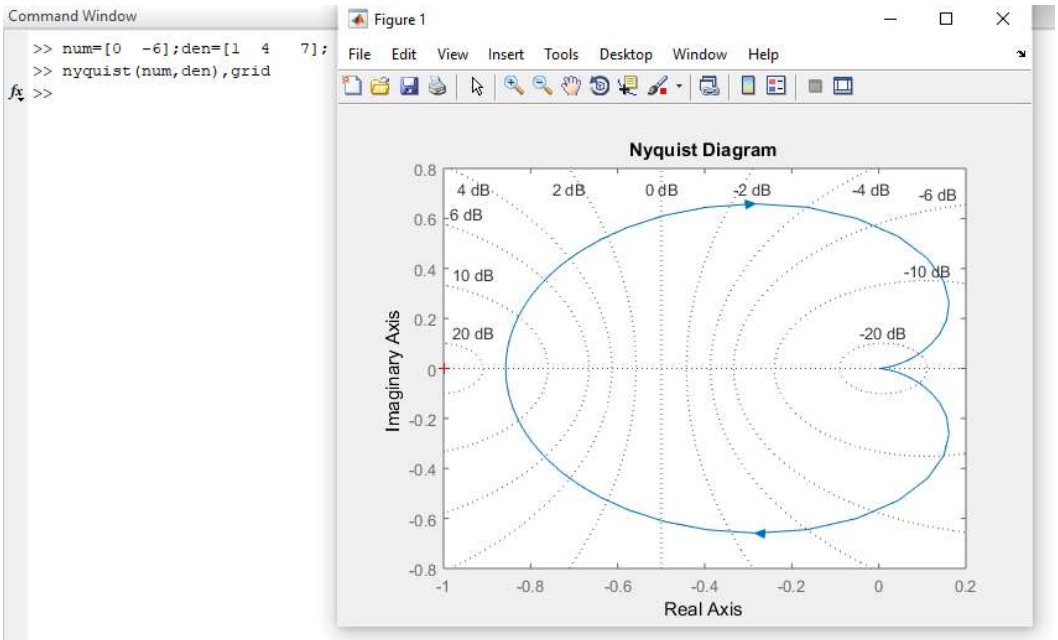


Figure 7.11. Example of Nyquist function

The Nyquist plots are used to analyze system properties including gain margin, phase margin, and stability when appealed without left-hand arguments. The Matlab command 'nyquist' produces a Nyquist plot as in the example of figure 7.11.

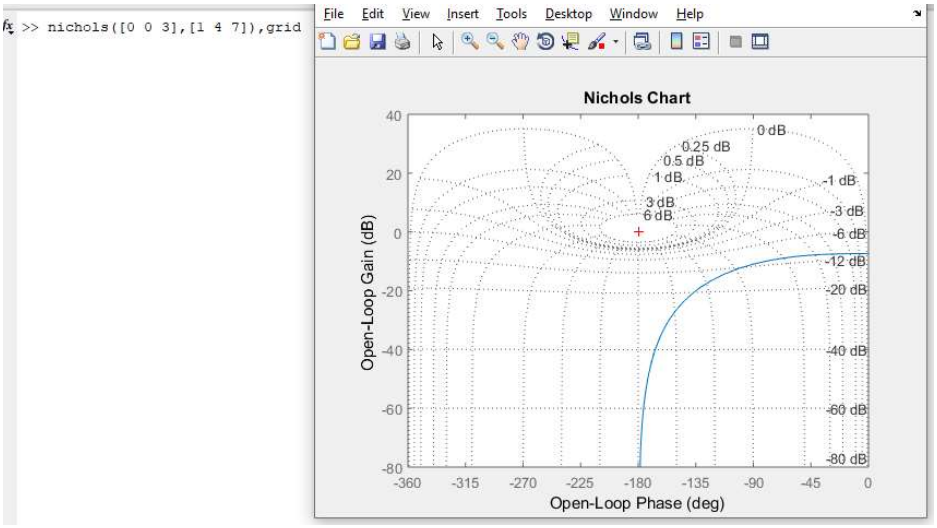


Figure 7.12. Example of Nichols function

nichols(sys) command Matlab creates a Nichols chart of the frequency response for the dynamic system sys. It displays the magnitude (in dB) plotted against the phase (in degrees) of the system response. It is useful to analyze open- and closed-loop properties of SISO systems. An example is shown in fig.7.12.

Bode Nyquist and Nichols diagrams of a discrete model can be obtained and plotted using dbode dnyquist and dnichols functions.

VII.4. Introduction to Control Designing

Several function of control are included in Control System toolbox. In this chapter, we will see some few examples. Such as:

- ✓ PID functions

The transfer function of parallel-form PID controller C:

a) $C = K_p s + \frac{K_i}{s} + \frac{K_d s}{T_f s+1}$

Where Kp is proportional gain, Ki: integrator gain, Kd: derivative gain, Tf: derivative filter time. The parallel-form PID controllers in matlab → C = pid(Kp,Ki,Kd,Tf)

b) $C = K_p(1 + \frac{1}{T_i s} + \frac{T_d s}{\frac{T_d}{N}s+1})$

Ti : integrator time ,Td: derivative time ,N : derivative filter constant.

The transfer function of standard-form PID controller C

in matlab →C = pidstd(Kp,Ti,Td,N)

Example

```
>> Kc1= pidstd(2,10,4,8);  % Standard-form controller
>> Kc2 = pid(Kc1);          %Proportional integral and Derivative Gains
```

```
Kc1 =

                1      1               s
  Kp * (1 + ---- * --- + Td * ------------)
                Ti     s           (Td/N)*s+1

  with Kp = 2, Ti = 10, Td = 4, N = 8

Continuous-time PIDF controller in standard form

KC2 =

                1            s
  Kp + Ki * --- + Kd * --------
                s          Tf*s+1

  with Kp = 2, Ki = 0.2, Kd = 8, Tf = 0.5

Continuous-time PIDF controller in parallel form.
```

Figure 7.13. Example of PID controller using commands

✓ Pole Placement: we use the function « place »

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu} \\ \boldsymbol{y} = \boldsymbol{Cx} + \boldsymbol{Du} \end{cases}$$

For a desired closed-loop pole locations p, place computes the gain matrix K such that the state feedback u = –Kx places the closed-loop poles at the locations p.

Example

Given the state space below, compute the feedback gain matrix K needed to place the closed-loop on poles p =[-5 -2 -4]:

$$\begin{cases} \dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -1 & -5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \\ y = [9 \quad 1 \quad 0] x + 0.u \end{cases}$$

The result using matlab command is:

K=place([0 1 0; 0 0 1;-6 -1 -5],[0;0;1],[-5,-2,-4])

K =

34.0000 37.0000 6.0000

VII.4 Conclusion

In this chapter, it was seen linear system analysis in matlab form. It is shown in frequency models and state-space models in order to introduce analysis and design methods. Then a little control design is given as the proportional-integral-derivative control, and pole placement design.

Chapter VIII: Introduction to Simulink

VII.1 Introduction
VII.2 Simulink: Getting started
VII.3 Building models
VII.4 Simulink related command window
M-file/ command windows' functions
ODE solver function

Mathematics is the basic building block of science and engineering, with Matlab, it is wide easier to do many computations. Therefore, MATLAB is like a powerful calculator that gives access to explore problems in science, engineering, and mathematics. Simulink is one of toolbox in Matlab used for this aim. In this chapter, we will introduce Simulink.

VIII.1. Introduction

Simulink is a tool that allows the modeling, simulation and analysis of linear or non-linear dynamic systems; analogous or discrete. In addition, the parameters can be modified online or during the simulation. It is built around a library of blocks classified by categories (discrete/ continuous systems, linear or not, connection blocks, etc.) [2,7,8].

VIII.2. Simulink: Getting started

- With Simulink student can do simulation and design dynamic systems.
- Simulink has a graphical enVIIronment →design, simulate, implement and test time-varying systems.
- To open Simulink there are manners: from command window by writing the command:

  >>Simulink

  Or using the up icon by click on Simulink Library

  In both case, we get

  Simulink Library Browser→ where one can find all Simulink's blocks

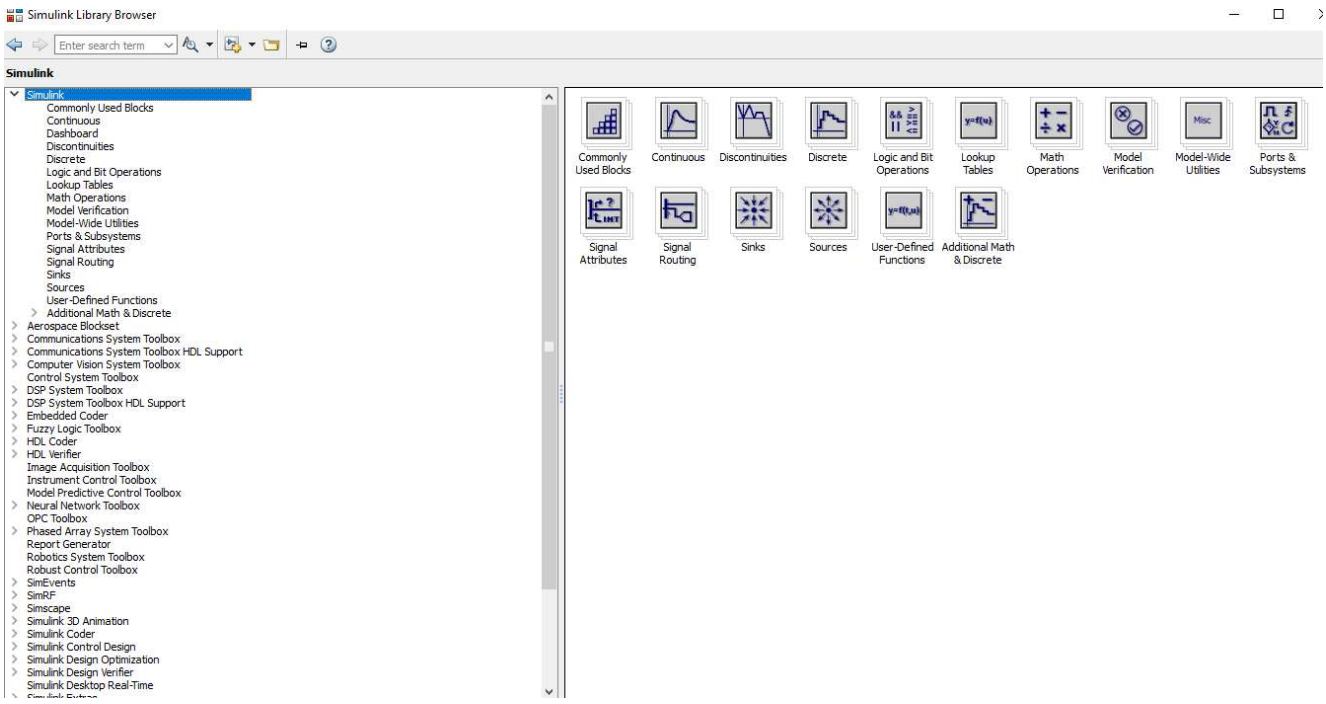


Figure 8.1. Simulink Library blocks

The figure 7.1 contains different types of blocs. They are classified according to their categories. These right side 'rectangles' contain each one a number of blocs. Choose a bloc of categories and click to get devices.

1. Double click on "Sinks" block for example

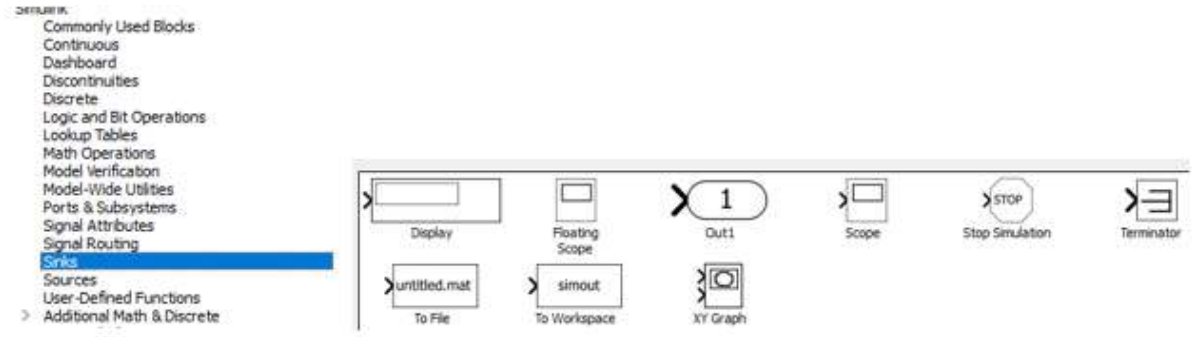


Figure 8.2. Sinks blocks

Sinks contains virtual signal visualization devices. In the figure 7.2 one can see the result of the click on ‘sink’, it appears some blocs that permit to see inputs and outputs values such as ‘display’ or graphs ‘scope’ or ‘XY Graph’. Data can be saved in files ‘To file’, or to ‘work space’. It can be choosen to anther output or Terminators.

2. Double click on “commonly used blocks” blocks

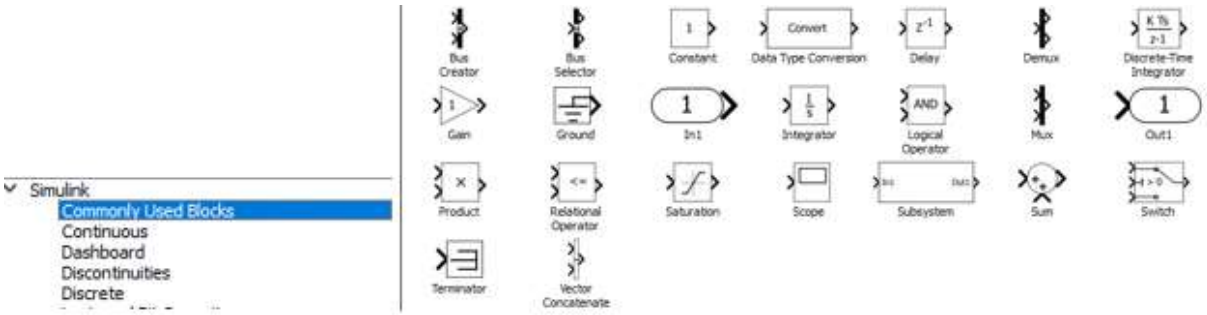


Figure 8.3. Commonly used blocks

This figure (7.3) the commonly used blocs that are used to connect some parts of dynamic models.

3. Double click on “sources” block

‘Sources’ contains simulated inputs to dynamic models we can see here ‘step’, ‘ramp’, ‘sine’, signal generators’ ..etc.

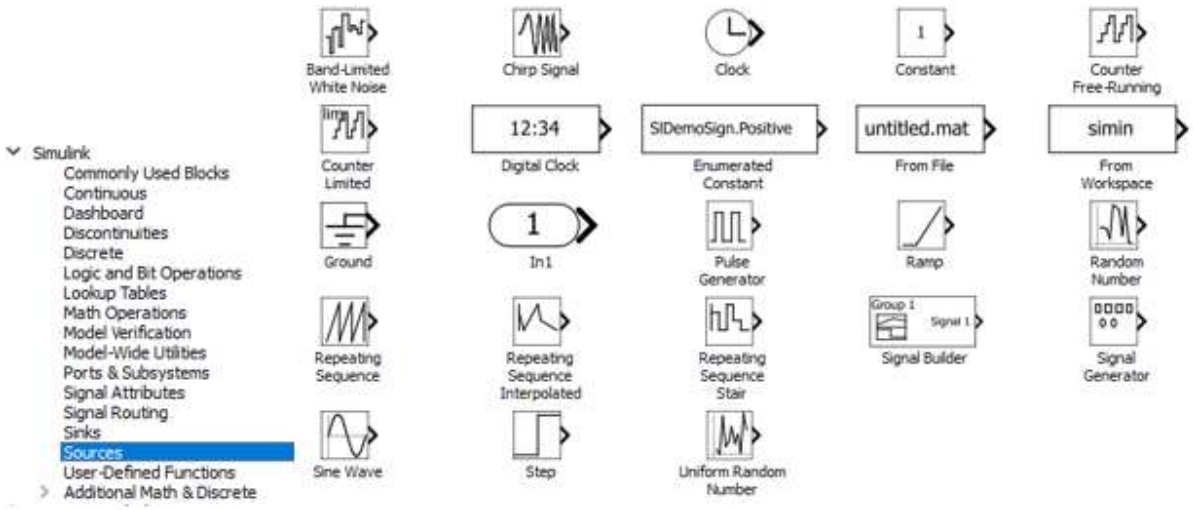


Figure 8.4. Sources blocks

4. Double click on “Simscape” block

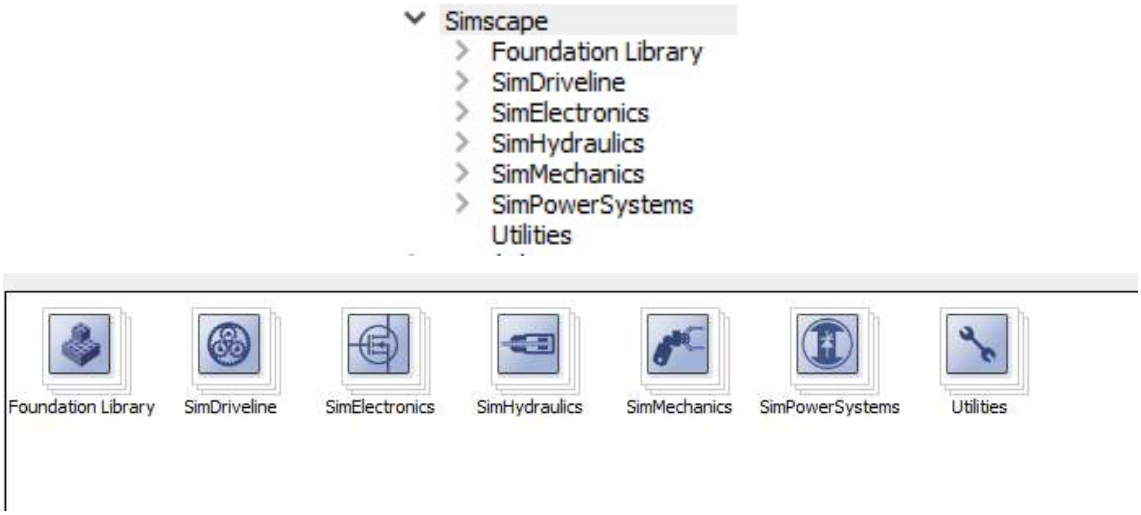


Figure 8.5. Simscape blocks

The block Simscape contains devices simulated with their actual comportment such as diodes, resistances, pumps and valves. They are also classified as in figure 7.5. The blocs ‘Simscape’ need ‘solvers’.

VIII.3. Building Models

Building a model is equivalent to creating a function in M-file.To create or ‘build’ models:

In command window Use icon New➔ Simulink model or in Simulink library browser click on new model.

In Simulink, there is several manners to build a same model: one can use the transfer function (see exemple1), use the mathematical and logical blocks (see example 2) or connect simulated deVIIces (see example 3)…etc.

Example 1: (Using transfer function)

The aim of this example is to create a dynamic model and to see the output behaviour.

The model chosen in this example has a transfer function: $h(s) = \frac{1}{s+1}$, the system is exited with a step input then with a sine once.

First: Go to icon of library browser select New Model (see fig 8.6-a).

You get a new white screen in which you will put all components this is 'model'.

Then select components one by one. You can see that is just enough to right click on the chosen block and use 'add to new model' (see fig8.6-b).

For this example, we need a bloc of transfer function; bloc of input and another for output (see fig 8.6-c).

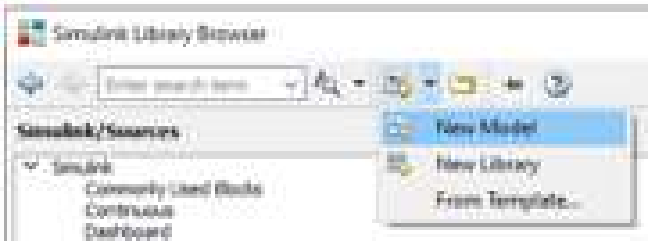


Figure 8.6-a. 1st: open a new model

Figure 8.6-b. 2nd ; add components in Simulink model

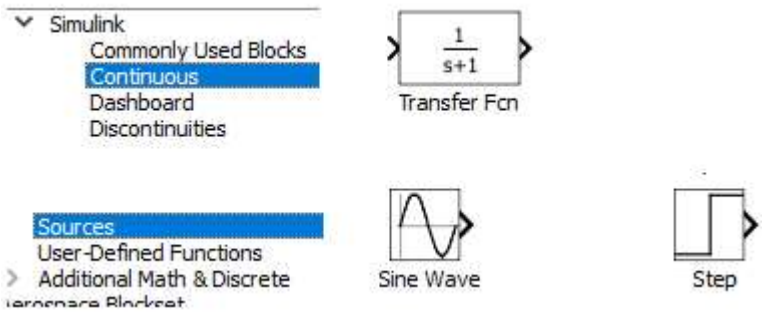


Figure 8.6-c. 3rd: getting components in example1

Figure 8.6 Create a Simulink model using transfer function

After selecting all blocks to the created model. You have to connect them (click on the end of on block, keep the mouse pushed until the next block and so on) then ran.

You can see result by double click on scope (see fig 8.7):

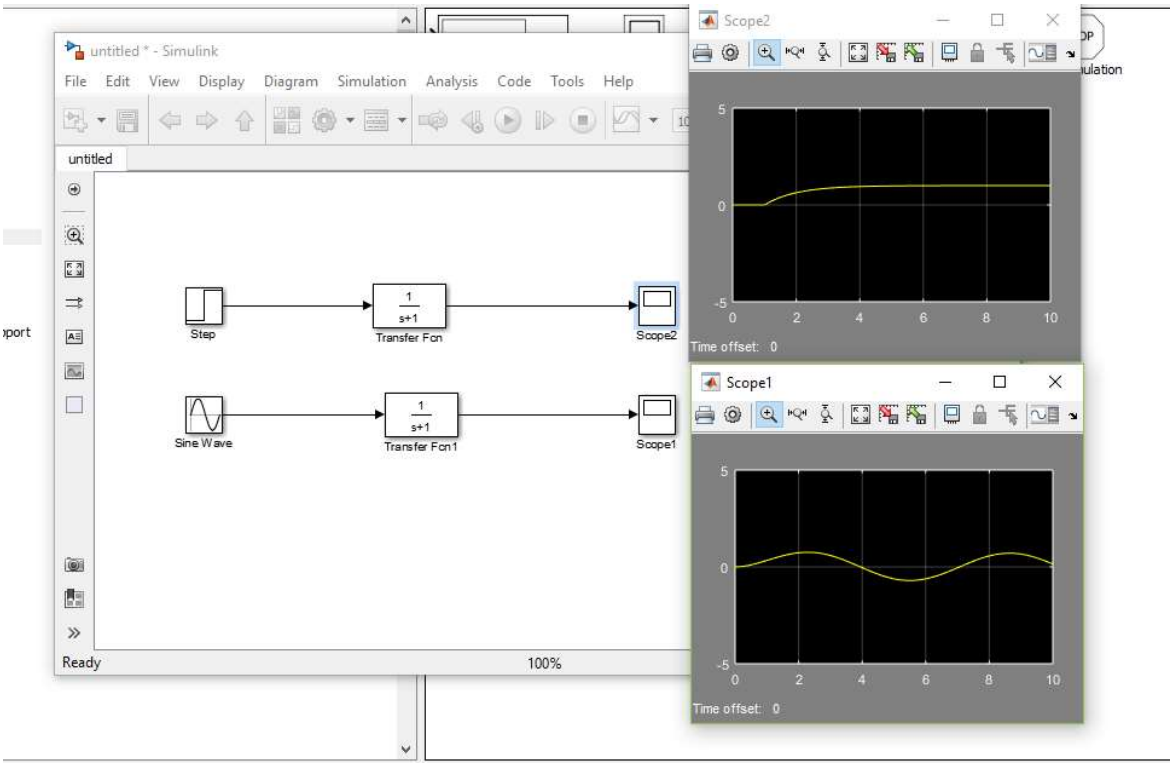


Figure 8.7. Result of the previous example

In this example, it is chosen the default parameters of Transfer function block, for more example, you have to double click on blocks, to adjust parameters:

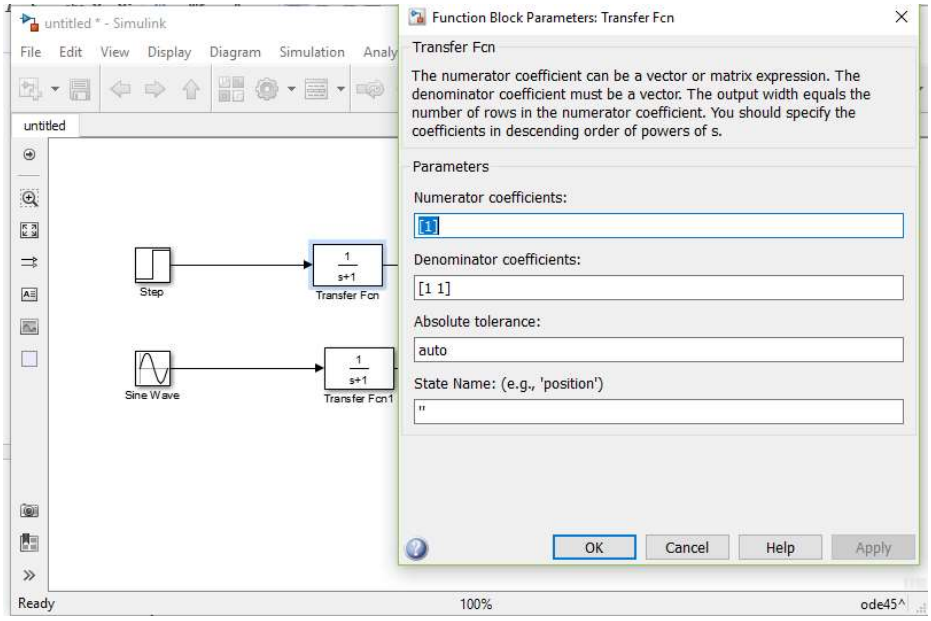


Figure 8.8. Example of changing parameters

Example2: half-wave rectifier by using mathematical and logical model

A second manner to build models is to use mathematical and logical category of blocks. The example of half-wave rectifier can be modelled in such manner.

For this case one can use logic blocs (see fig 8.9-a). Since, the diode effect, as only the positive signal, is easy to do with such blocs.

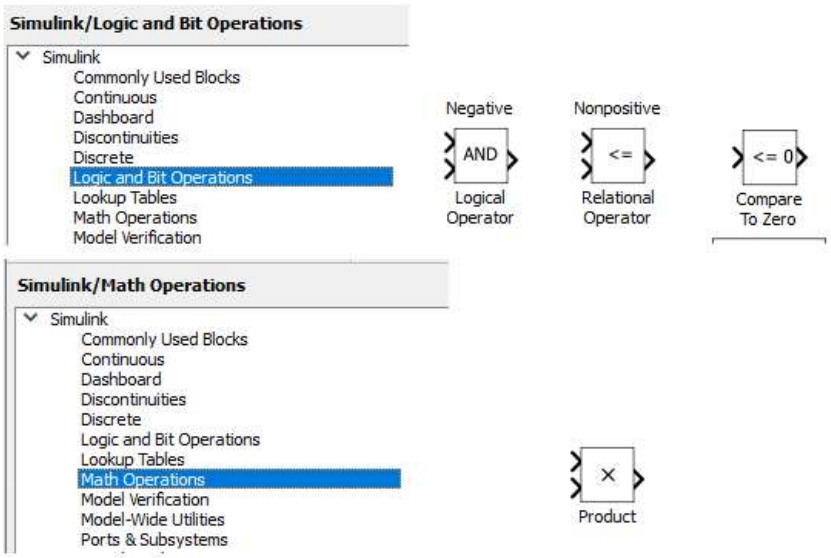


Figure 8.9-a: getting components in example2

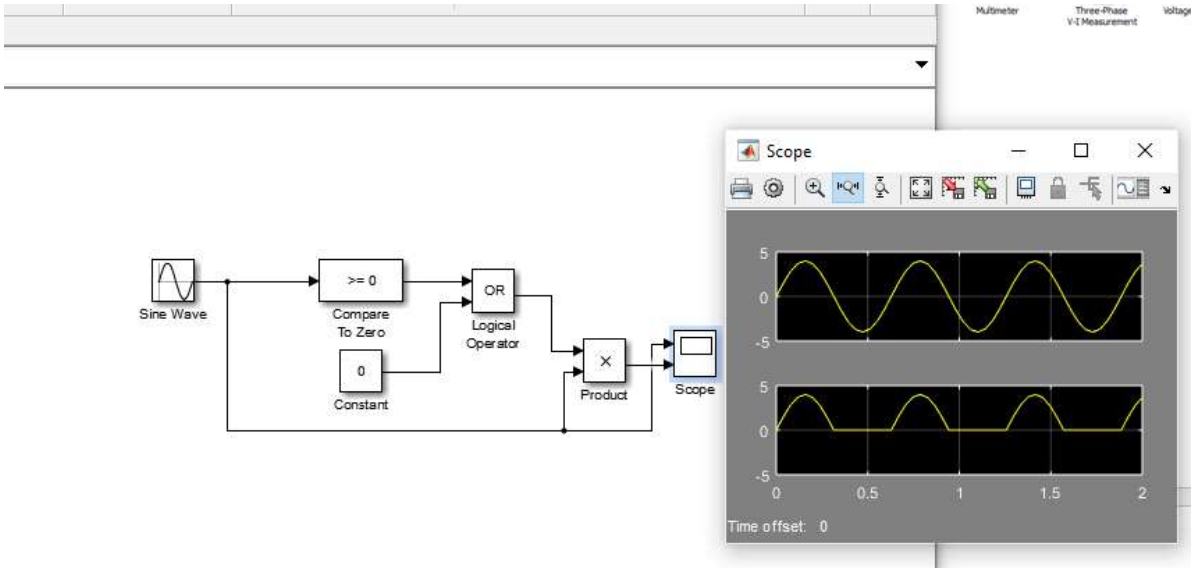


Figure 8.9.-b. The example after run

Figure 8.9.Example of creating Simulink model using mathematical model

Example3: half-wave rectifier Electrical components

In this example, one can see the half –wave model by using diode, resistance and signal generator this means to use simscape block.

Using this block may create a problem to of connections in two levels. First, when going from the physical inputs. In this level, an inner Simulink program will transform to mathematical differential equation to simulate. In order to tolve such problems you need to conncet solvers (see fig 8.10).

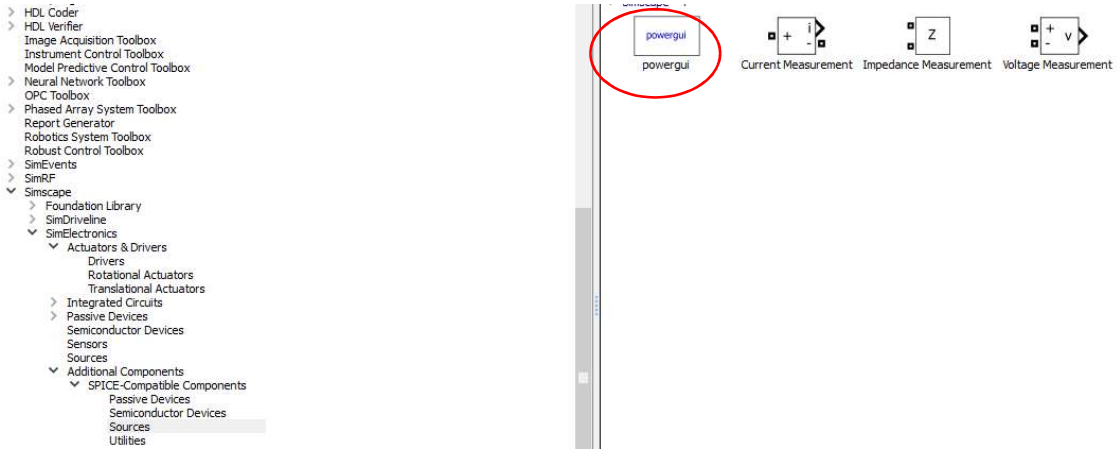


Figure 8.10. Location and choice of powergui

The second problem is when the choice of devices from the library is not fine. You see that the wire cannot be connected the well be in discontinuous and red (see fig 8.11).

In the example 3, to build a model for a half-wave you need: Resistance, Diode and ac source but the scope cannot ‘measure’. **You have to add voltage measurement**! (See fig8.12-a)

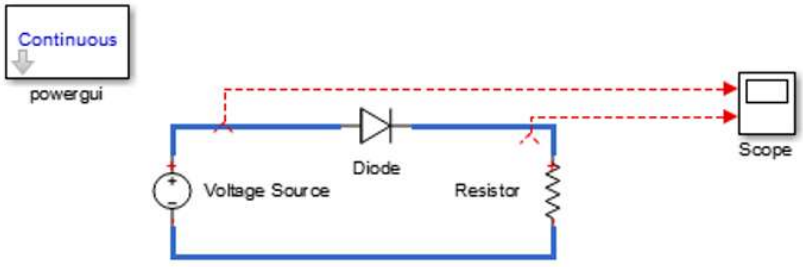


Figure 8.11. Example of no connections problem using Simscape

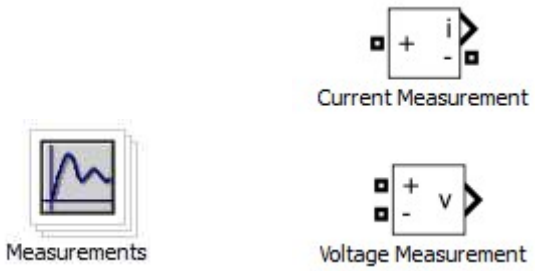


Figure 8.12-a. Choice of voltage measurement in the previous example

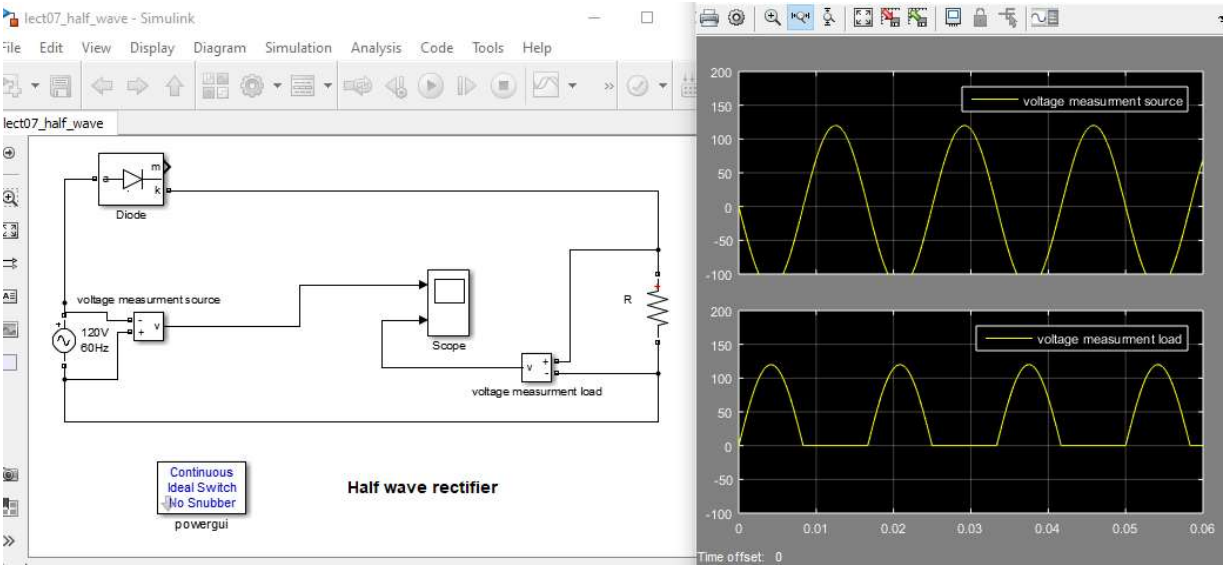


Figure 8.12. –b Result after run

Figure 8.12. Result of the example 3

To see both input and output you can use two scopes one for each one, as you can use one scope (fig 8.12-b) and adjust the scope parameters to get to inputs. These will create automatically two connections the up connection will gave result in the up figure and the down one his figure is down.

Example4: block masking

To allow bringing together a certain number of blocks performing a function into a single block. This by first selecting all the models with the mouse, then choosing the 'Subsystem mask' option from the menu. Then we get the masked block as in figure (fig8.13).

This is useful when the model contains several blocks so it make it difficult to understand the fonctionarity of the model by one look. You 'mask 'subsystems' to give a general look. Then to get details, click on subsystems one by one.

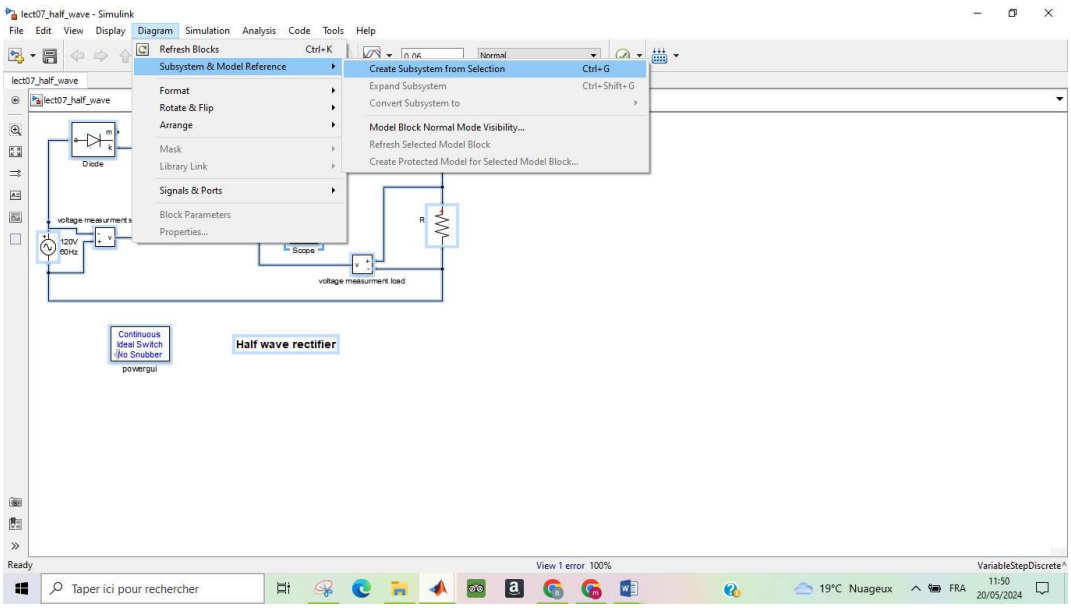


Figure 8.13. Result of the mask operation

VIII.4 Simulink related command window

In the previous section, one can see how to select and connect blocs to get Simulink model then on can visualize the outputs. As a function of M-file one can save and run it again from Simulink or from M-file. In this section, one can see how to call, quit rectify and run Simulink model from an M-file or command window.

VIII.4.1 M-file/ command windows' functions

One can configure and run Simulink throw m-file or command window.

- ✓ Simset → modify simulation setting
- ✓ Sim → simulate dynamic system

Example 5:

```
T_final=100;  % to stop simulation
T_initial=0.1; %
options=simset('solver','ode45','fixedstep', T_initial);
Sim('name_simul1', T_final,options)
```

In this example it is supposed that we had created before a Simulink model and save it with the name ‘'name_simul1'’ (change the default name untitled). Then, in command window (or

in M-file), we write the previous command lines which stand to: precise the final and initial time, to change the default options then call and run our Simulink model.

VIII.4.2. ODE solver function

ODE solvers are used in Simulink to solve differential equations. Then can be used also in command window.

Example 6: Use of ODE out of Simulink

To describe the principal of work of ODE solvers we have to define the differential equation as state function in the script file then we call the function by ODE with the initial conditions

Let's take the following differential equation:

$$\ddot{x} = -3\dot{x} - x + 9$$

To solve it in matlab using ODE functions

-First we create a script file were it contain the state vector

```
function dx=calculation(t,x)
dx=zeros(2,1);
dx(1)=x(2);
dx(2)=-3*dx(1)-x(1)+9;
```

- Then to solve the differential equation we need an interval of time t (t=0:10 for ex) and initial conditions of the vector $[x \quad \dot{x}]$ and to call function in the script.

```
>> [T,X]=ode45(@calculation,[0 10],[0  0]);
```

Exercise

Consider the simple pendulum. Where l denotes the lenth of the rod and m the mass of the bob. We assume that the rod is rigid and has zero mass.

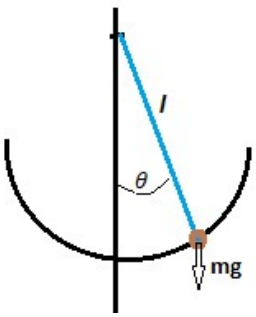


Figure 8.14. The simple pendulum model

θ is the angel from the rod to the vertical axis.

The pendulum is free to swing in the vertical plane.

In order to get a linear model, one can consider only the gravitational force "mg" and the frictional force which is proportional to the speed "$-kl\dot{\theta}$ ".

Using Newton's second law of motion:

$$ml\ddot{\theta} = -mgsin(\theta) - kl\dot{\theta}$$

The equation of motion:

This system is stable at zero for θ .

If we apply a torque T (ie: a push to the bob), it gets:

$$\mathrm{ml}\ddot{\theta} = -\mathrm{mg}\sin(\theta) - \mathrm{kl}\dot{\theta} + \mathrm{T/l}$$

1. Solve the differential equation:

➔ use ode functions

$$\ddot{\theta} = -\frac{g}{l}sin(\theta) - \frac{g}{m}\dot{\theta} + \frac{1}{ml^2}T$$

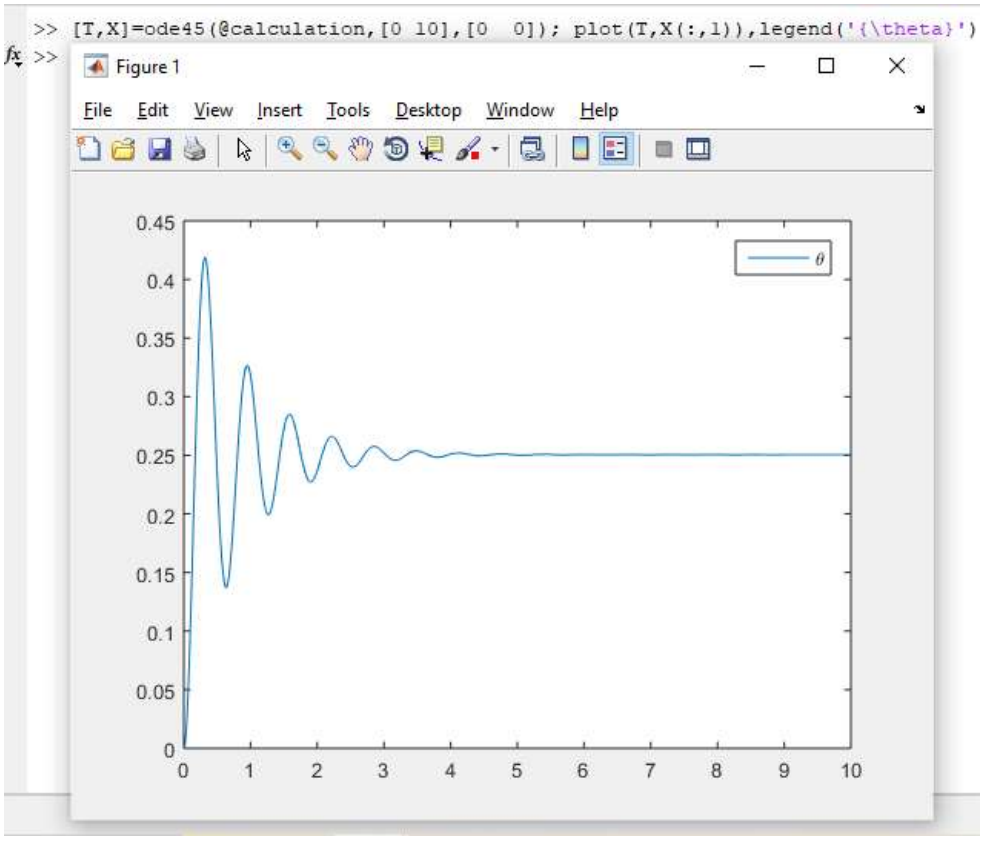


Figure 8.15. Result of application of ODE to simple pendulum model

2. The state model:

We take $x = \theta$. Then

$$\begin{cases} \dot{x} = \dot{\theta} \\ \ddot{x} = -\frac{g}{l} sin(x) - \frac{g}{m}\dot{x} \end{cases}$$

For $\theta \approx 0$ we can write $sin(\theta) = \theta$ then if we take $X = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}$

We get $$\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} X(2) \\ -\frac{g}{l}X(1) - \frac{g}{m}X(2) \end{bmatrix}$$

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & -\frac{g}{m} \end{bmatrix} X$$

The torque (push to the bob) will be viewed as an input control.

Yield to $\ddot{x} = -\frac{g}{l} sin(x) - \frac{g}{m}\dot{x} + \frac{1}{ml^2}T$

$$\dot{X} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & -\frac{g}{m} \end{bmatrix} X + \begin{bmatrix} \frac{1}{ml^2} \\ 0 \end{bmatrix} T$$

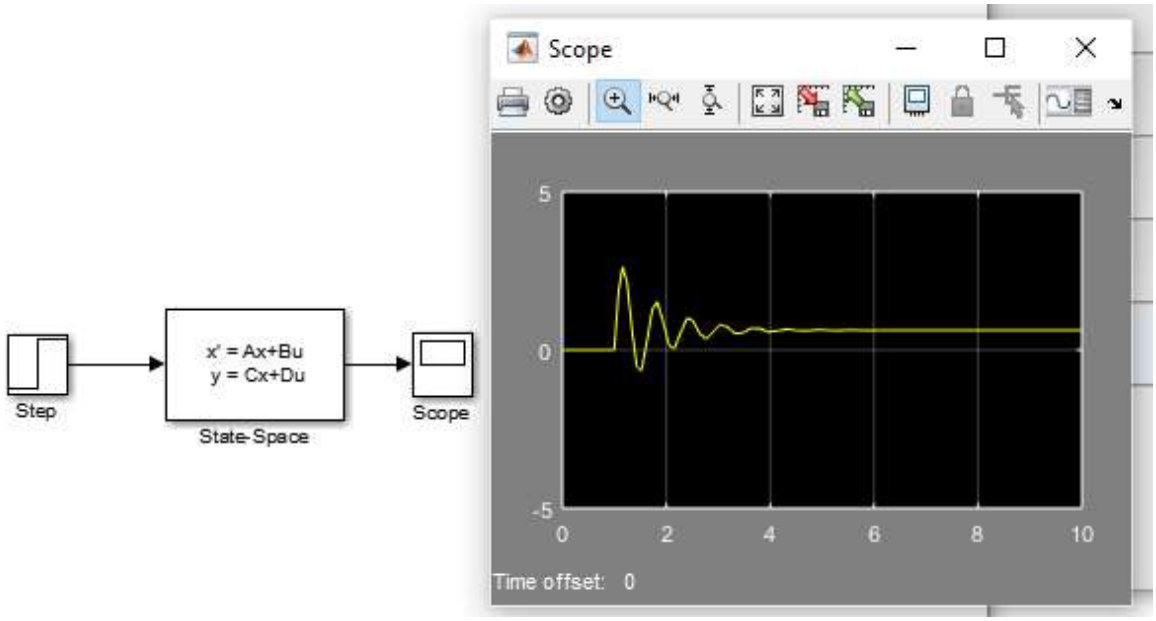


Figure 8.16. Result of Simulink stat space simple pendulum model

3. The transfer function model:

$$ml\ddot{\theta} = -mgsin(\theta) - kl\dot{\theta}$$

$$mlS^2x + mgx + klSx = T/l$$

$$(ml^2S^2 + kl^2S + mlg)x = T$$

$$G(S) = \frac{x}{T} = \frac{1}{(ml^2S^2 + kl^2S + mlg)}$$

```
>> m=4;l=0.1;g=9.98;k=0.6;
>> num=[1]; den=[m*l*l,k*l*l,m*l/g]; G = tf(num,den)

G =

                1
  ----------------------------
  0.04 s^2 + 0.006 s + 0.04008

Continuous-time transfer function.
```

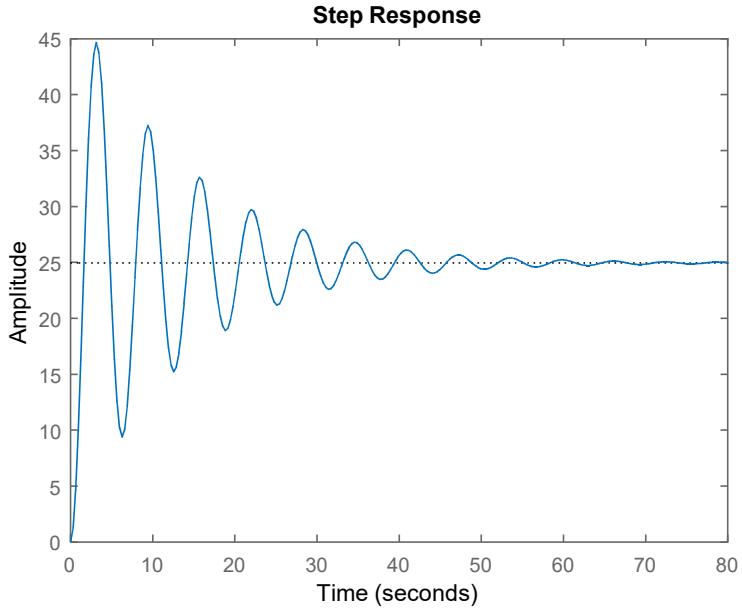


Figure 8.17. Result of using transfer function for the simple pendulum model

VIII.5 Conclusion

This chapter is a brief introduction to Simulink. Student can start using the Toolbox correctly. Therefore, he can further built models that are more complicated or model for a real situations. with basic steps, how he can use Simulink.

General conclusion

In conclusion, this course handout was written to provide students with a basic overview of commands, which are useful in science and engineering.

We have systematically explored how to calculate with Matlab: We began with scalars, vectors then matrices. Delved into linear Algebra calculations then the specifics case of solving linear equations.

Aftermaths, one can learn to Plot graphs in Matlab. We gave some details to simplify graphing for more than one graph and to shecht in 3 dimensions.

Our key outcomes include programming in Matlab. This needs to use control structures such as loops, conditions.

The handout describes moreover the 'Graphical User Interface' with examples. One can learn to clip the correct component and adjust it, and then use handles to program small tasks.

Student is familiarized with polynomial calculations also, he get to evaluate a polynomial function, to calculate, differentiate or integrate polynomials; in more, one can use to find their zeroes. Then he can interpolate data using polynomials.

We explained an example of Matlab's toolbox: the on for linear system analysis in Matlab form and control design 'control system toolbox'. Throughout, we encountered several manners to analyses and control dynamic models in continuous and discrete form.

Additionally, one significant issue in Matlab Toolboxes is 'Simulink'. This includes graphical form to 'model' system. In such way student can start to build dynamic simulation.

Bibliographi

[1] Kharab, A., & Guenther, R. (2018). An introduction to numerical methods: a MATLAB® approach. CRC press.

[2] Bashier, E. B. (2020). Practical Numerical and Scientific Computing with MATLAB® and Python. CRC Press.

[3] Xue, D., & Chen, Y. (2018). Scientific computing with MATLAB. Chapman and Hall/CRC.

[4] Martinez, W. L. (2011). Graphical user interfaces. Wiley Interdisciplinary Reviews: Computational Statistics, 3(2), 119-133.

[5] Goeser, P. T., & Duong, A. (2019). MATLAB Graphical User Interfaces (GUIs) for Problem Solving in Thermodynamics.

[6] Gupta, A. (2015). Numerical methods using MATLAB. Apress.

[7] Yang, W. Y., Cao, W., Kim, J., Park, K. W., Park, H. H., Joung, J. & Im, T. (2020). Applied numerical methods using MATLAB. John Wiley & Sons.

[8] Singh, K. K., & Agnihotri, G. (2012). System Design through Matlab®, Control Toolbox and Simulink®. Springer Science & Business Media.