Week 1 Tutorial Notes: Introduction to MATLAB

Zeyad Manaa, Mohammed Elbalshy

Aug 26, 2023

Supplementary document for AE 315
Systems and Control course
Fall 2023

Aerospace Department
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

Abstract

This document provides a bird's-eye overview to MATLAB, aiming to familiarize the readers with this software tool. While acknowledging its limitations this document serves as a starting point for the readers to explore and utilize these tools. However, through the course progression, students will delve into a more comprehensive methods that are central to the first course in controls, alongside a taste of advanced topics and techniques, enabling them to a get access to array of skill set to effectively employ them in favour of control theory.

Contents

List of Figures

Listings

1 MATLAB

1.1 MATLAB Installation

We will consider the fastest way to install MATLAB by using the internet connection method provided in [1] as following:

1. Sign in to Your MathWorks Account from here or sign up if you do not have account. Use your university email in the sign up process in order to get the academic licence.
2. Download and run the installer from Downloads page.
3. In MATLAB installer, sign in using the email you have set in step 1.
4. Accept the licence agreement, and choose the setup location.
5. Include all necessary packages in the installations such as:
   (a) Signal Processing Toolbox.
   (b) Curve Fitting Toolbox.
   (c) Optimization Toolbox.
   (d) Symbolic Math Toolbox.
   (e) Control System Toolbox.
   (f) System Identification Toolbox.
   (g) Model Predictive Control Toolbox.
   (h) Fuzzy Logic Toolbox.
   (i) and so on ...
6. Complete the installation instructions following the prompts.

For more detailed installation instructions please refer to [1].

1.2 MATLAB Interface

Upon opening MATLAB, you will be faced with a unique window known as the MATLAB desktop. This desktop can be considered as a central hub, including various other windows and providing access to important tools and features as shown in figure 1. For the purpose of this document, as a notation simplification, we will utilize the `>>` operator to distinguish the code. This operator is adopted from MATLAB command window.

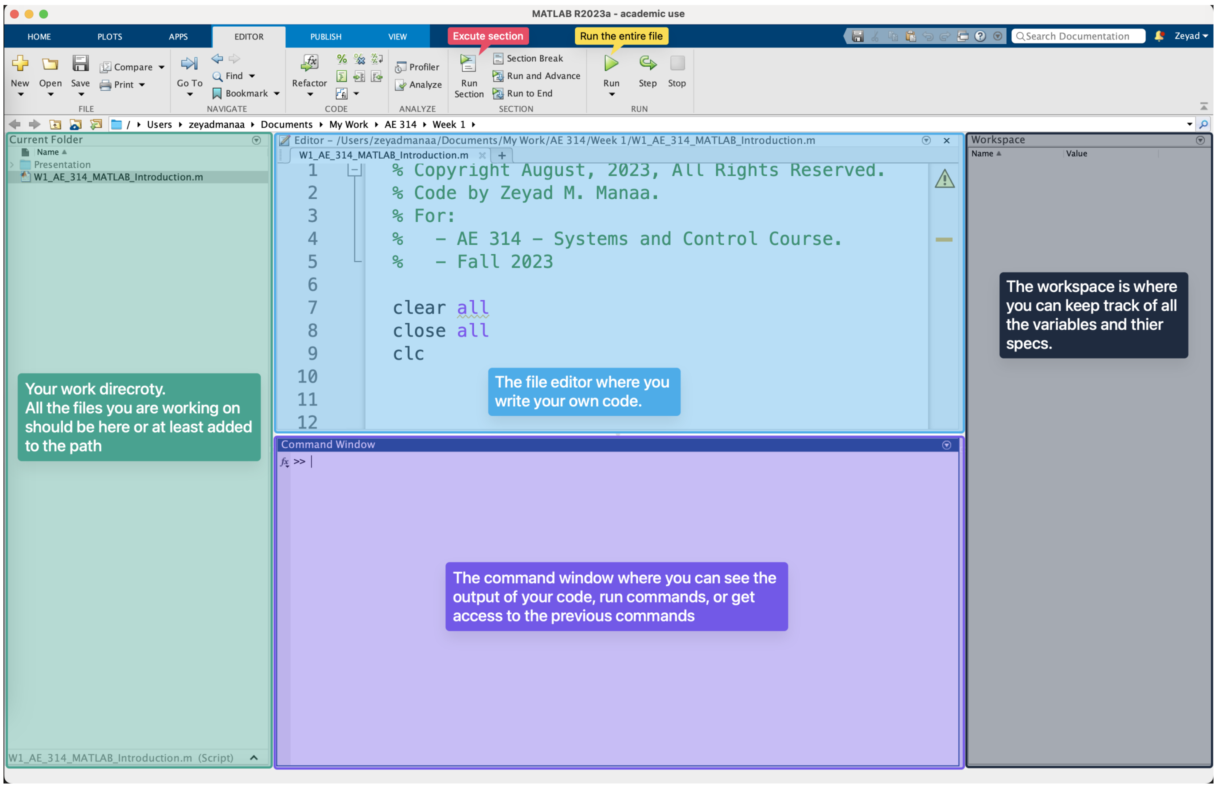
1.3 Basic Mathematical Operations

1.3.1 Using MATLAB as a calculator

For example, as a basic arithmetic operation, suppose you want to evaluate the expression $3 + 2 \times 4$, we will going to use the simple command window introduced earlier, just write down as follow,

```
>> 3+2 * 4
    ans =
        11
```

You should have noticed that MATLAB has stored the value for `ans` as an abbreviation for



```
% Copyright August, 2023, All Rights Reserved.
% Code by Zeyad M. Manaa.
% For:
%    - AE 314 - Systems and Control Course.
%    - Fall 2023

clear all
close all
clc
```

Your work direcroty.
All the files you are working on should be here or at least added to the path

The file editor where you write your own code.

The workspace is where you can keep track of all the variables and thier specs.

The command window where you can see the output of your code, run commands, or get access to the previous commands

Figure 1: MATLAB desktop

`answer`. This is typically the case when you did not assign a *variable*[1] (will come to this later) name to the operation. Note that this value (`ans`) is assigned if it does not exist from the first place, and if existed, MATLAB overwrites it with the new value. So, suppose you conducted another arithmetic as follows,

```
>> (4-2) * 4
   ans =
      8
```

Notice that the new value for `ans` is overwritten by the new value which is 8. That means if you are going to use it in the future it will have the value of 8 not the previous value of 11.

To avoid this, you can easily assign a variable name to the operation instead of just overwriting the `ans` every time as follow,

```
>> x = (4-2) * 4
   x =
      8
```

This variable can be always used in future operations. For example, you can calculate the value of `4*x`. It will result in,

```
>> 4*x
   ans =
      32
```

To suppress the answer from appearing in the command window, one can add a semicolon (;) after each command.

```
>> 4*x;
```

[1]It is advised to give your variables a meaningful names. If you are calculating, for example, the control input for a system, you should avoid naming your variable as `u = <<something>>`. Instead, use a proper naming like `controlInput = <<something>>`. This will make your code more readable as you go on, as well as, more searchable.

As a concluding remark for this subsection, table 1. summarises the operations mentioned earlier and some more.

SYMBOL	OPERATION	EXAMPLE	MATH	OUTPUT
$+$	Addition	3+2	3+2	`5`
$-$	Subtraction	12.3 - 5	12.3 - 5	7.3000
$*$	Multiplication	0.45 * 972.503	0.45×972.503	`437.6264`
/	Division	5 / 98.07	$\frac{5}{98.07}$	`0.0510`
ˆ	Power	$4^{7.1}$	$4^{7.1}$	`1.8820e+004`
`sqrt()`	Square root	`sqrt(15)`	$\sqrt{15}$	`3.8730`

Table 1: Basic arithmetic operators and other.

1.4 Matrices and Vectors

The basis of MATLAB is matrices. Consequently, we must learn about matrix creation and manipulation. There are various methods for creating matrices.

1.4.1 Vectors

A matrix's special case is a vector. This section's goal is to demonstrate how to build vectors and matrices in MATLAB. Following the math notation, any vector will be denoted by a bold symbol. A vector is typically, an array of numbers, ordered in a *row* or a *column* structure. An array of dimensions $1 \times n$ is a *row* vector. On the other hand, an array of dimensions $m \times 1$ is a *column* vector. In MATLAB, the elements of a vector are enclosed by a square brackets and are separated by *spaces*, *commas*, or *semi-colons*. For examples, let $\boldsymbol{x}$ be a vector that has the elements

$$\boldsymbol{x} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

and we want to define in MATLAB. One can note this a vector of size 1×5, which is a row vector. Two ways of doing this is

```
>> x = [1 2 3 4 5];
```

or

```
>> x = [1, 2, 3, 4, 5];
```

Also, let $\boldsymbol{y}$ be column vector defines as,

$$\boldsymbol{y} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^\top$$

where $\top$ is the transpose operator. This can be written in MATLAB language as,

```
>> y = [1; 2; 3; 4; 5];
```

also, the transpose operator turns a *row* vector into a *column* vector. An apostrophe or single quote (') indicates the transposition operation.

So, considering, on can define the $\boldsymbol{y}$ vector by simply transposing the $\boldsymbol{x}$ by the following command,

```
>> y = x'
   y =
      1
      2
      3
      4
      5
```

One can access any element by the following command,

```
>> y(1)
   ans =
      1
```

also, a stack of elements can be accessed by,

```
>> x(1:3)
   ans =
      1 2 3
```

or, from on index to the end as,

```
>> x(3:end)
   ans =
      3 4 5
```

The matrices or vectors we work with have too many elements for us to enter one at a time. As an illustration, let's say we want to enter a vector x with the points $\boldsymbol{x} = \begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 & \ldots & 5 \end{bmatrix}$. Utilization the *colon* operator will facilitate the operation,

```
>> x = 0:0.1:5
```

and the resulting vector will have 51 elements.

1.4.2 Matrices

A matrix is also an array of numbers. In order to define a matrix in MATLAB do the following,

1. Begin with a square bracket ([), as you have seen with vectors.
2. Separate the elements of each row with spaces or commas (,).
3. Separate each row by a semi-colon (;).
4. Close the previously opened square bracket (]).

Consider the matrix A defined as,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

which can be written in MATLAB as,

```
>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
   ans =
      1 2 3
      4 5 6
      7 8 9
```

As done before with vectors, we can access an element of A by,

```
>> A(1,3)
   ans =
      3
```

`A(1,3)` element is an element located in the first row and third row. Its value is 3. Additionally, you can overwrite an element in matrix using,

`>> A(1,3) = 0` This line substitutes the value of 3 by 0. To check, we simply can write A in the command window as,

```
>> A
```

```
   ans =
      1 2 0
      4 5 6
      7 8 9
```

Single elements of a matrix are accessed as `A(i,j)`, where $i \geq 1$ and $j \geq 1$. negative and zero subscripts are not available in MATLAB.

A specific row or column can be chosen using the colon operator. `A(m:n,k:l)`, for instance, defines rows m to n and columns k to l. Parts of a matrix are referred to through subscript expressions. For instance,

```
>> A(2,:)
   ans =
      4 5 6
```

The colon operator can also be used to extract a sub-matrix from a matrix A

```
>> A(:,2:3)
   ans =
      1 2
      4 5
      7 8
```

The matrix transposition operator is the same as done in vectors using (').

Basic matrix generation functions are available in MATLAB. The functions `zeros`, `ones`, and `eye`, respectively, return the matrix of zeros, the matrix of ones, and the identity matrix. Table 2 describes them.

COMMAND	DESCRIPTION
`eye(m,n)`	Returns an $m \times n$ matrix with 1 on the main diagonal
`ones(n,m)`	Returns an $m \times n$ matrix of ones
`zeros(m,n)`	Returns an $m \times n$ matrix of zeros
`rand(m,n)`	Returns an $m \times n$ matrix of random numbers

Table 2: Elementary matrices

As we just mentioned, MATLAB enables the arithmetic operations +,-,*,/ and to be performed on matrices. These operations are valid for matrices as long as they satisfy the mathematical conditions required to make a specific operations. For example, if `A` and `B` are matrices of the same size, we can add them, if `A` has a number columns as the same number of `B`'s number of row, we can multiply them, and so on.

On the other hand, array arithmetic, also known as *array operations*, are carried out element by element. The *array operations* are distinguished from matrix operations by the period character (.). The character pairs (.+) and (.-) are not utilized, however, because addition and subtraction are performed using the same matrix and array operations.

For example, consider

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

we can apply element by element multiplication as,

```
>> C = A.*B
   C =
      10 40 90
      160 250 360
      490 640 810
```

Table 3 summarize the differencs between the matrix and array operations.

Operation	Matrix	Array
Addition	$+$	$+$
Subtraction	$-$	$-$
Multiplication	$*$	$.*$
Division	$/$	$./$
Left division	$\backslash$	$.\backslash$
Exponentiation	^	.^

Table 3: Matrix and array operations

1.5 Visualization and Plotting

1.5.1 Creating 2-D plots

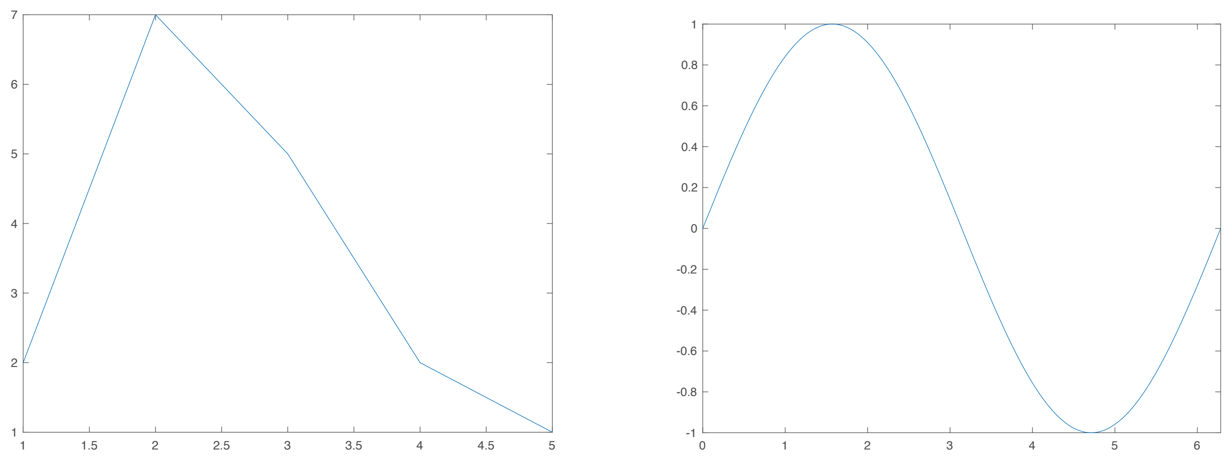
2D plotting is the basic MATLAB visualization procedure. It is done by taking two row or column vectors with the same length, $\boldsymbol{x} = [x_1, \ldots, x_i, \ldots, x_n]$ as a vector of x-coordinates and $\boldsymbol{y} = [y_1, \ldots, y_i, \ldots, y_n]$ as a vector of y-coordinates. Where each i^{th} element of vector $\boldsymbol{y}$ is plotted with its corresponding i^{th} element of vector $\boldsymbol{x}$. The plot command is `plot(x,y)`.

Listing 1: Plot basics example 1

```
x = [1:5];
y = [2,7,5,2,1]
plot(x,y)
```

Listing 2: Plot basics example 2

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```



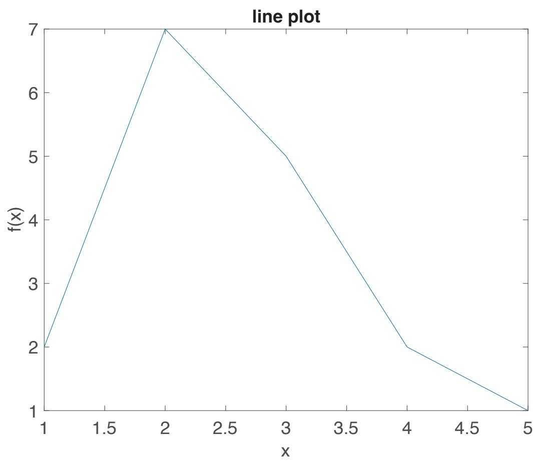
(a) Random vectors plot (b) $y = \sin x$

Figure 2: Plotting basic functions

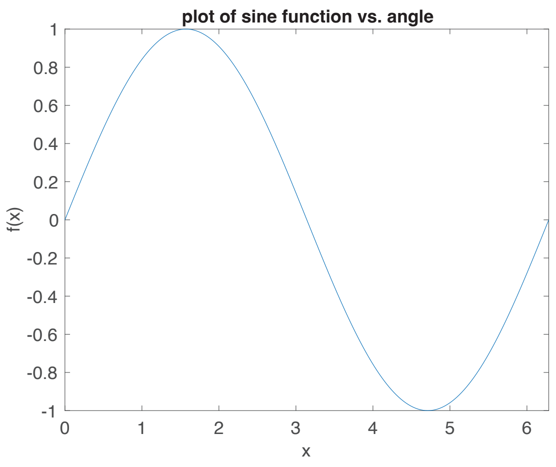
1.5.2 Adding title, axis labels and legend

MATLAB enable users to add axis label and figure titles. These can be added to Graph no. 3 by the following commands:

```
>> xlabel('angle')
>> ylabel('sin(x)')
>> title('plot of sine function vs.  angle')
```



(a) Random vectors plot



(b) $y = \sin x$

Figure 3: Plotting basic functions

1.5.3 Multiple data sets in one plot

This subsection discusses two ways to plot multiple data sets in one plot. The first one is to include all the data set pairs together in the plot command as

```
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
```

The other way is to use hold on command.

Listing 3: Multiple data sets in one plot example

```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
plot(x,y1,' -')
hold on
plot(x,y2,'-')
plot(x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
```

To be able to distinguish between different plots on the same figure, command legend can be used to name each data set on the plot.

```
>> legend('2*cos(x)','cos(x)','0.5*cos(x)');
```

The names in the command line should be ordered according to the first plotted data sets. The code result can be found in figure 4.

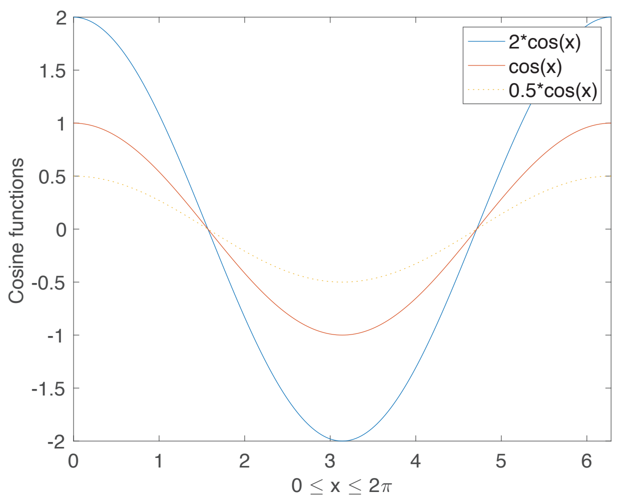


Figure 4: Multiple data sets super-imposed in one plot

1.5.4 Subplots

Subplot command divide the figure into different subplots in the same figure and create axes for each one. `subplot(m,n,p)` divides the figure into $m \times n$ grid. p indicates the position for the subplot figure. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes.

Listing 4: Subplot example

```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
y4 = sin(x);

figure % to open a new figure
title('cosine functions')
subplot(221)
plot(x,y1)
xlabel('0 \leq x \leq 2\pi')
ylabel('2*cos(x)')
subplot(222)
plot(x,y2)
xlabel('0 \leq x \leq 2\pi')
ylabel('cos(x)'')
subplot(223)
plot(x,y3)
xlabel('0 \leq x \leq 2\pi')
ylabel('0.5*cos(x)')
subplot(224)
plot(x,y4)
xlabel('0 \leq x \leq 2\pi')
ylabel('sin(x)')
```

The code outcome is presented in figure 5.

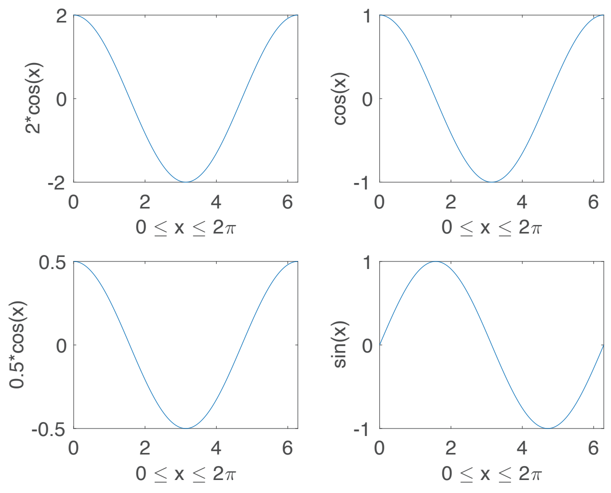


Figure 5: Subplot example

References

[1] "Install Products Using Internet Connection - MATLAB & Simulink," 2023. https://www.mathworks.com/help/install/ug/install-products-with-internet-connection.html.