CHAPTER 1

Introduction to MATLAB

In this chapter, we will talk about the basics of MATLAB and how to get started with it. We expect the reader to have basic programming skills, and therefore we will not cover any MATLAB programming concepts in detail. When discussing any topic, we will try not to go into such specifics that cause us to deviate from our main goal. Instead, we will provide proper references which can be consulted for more information.

Introduction

MATLAB is a programming language created with the basic goal of providing a simple intuitive platform for engineering design applications. However, as well as being a programming language, it can also be classified as software for computation and visualization. On one hand, you can call sophisticated programming routines in MATLAB, while on the other hand you can just open a simple graphical user interface to fill in the specifications, click OK, and MATLAB will perform computations and visualize it for you.

As we will see in later chapters, most engineering tasks require processing of matrices, for example image processing or data regression. MATLAB provides an excellent platform and routines for matrix operations. In fact, MATLAB is short for MATrix LABoratory because its main feature is to provide direct operations on matrices and to avoid complicated loops.

Current engineering applications use numerical simulations extensively, which require specific functions along with a programming language. For example, a signal processing application will need filter, fft and similar basic functions. In typical programming environments such as C/C++, you need to write these basic functions yourself. MATLAB provides all such basic and advanced functions in built-in packages known as toolboxes. MATLAB currently has a wide range of toolboxes for different engineering fields, *e.g.* it has toolboxes devoted to signal processing, image processing, communication, control systems and finance. In addition, some engineers are more inclined towards graphical models and want to avoid writing programming routines. MATLAB also features Simulink, which provides a platform for building and simulating graphical models without knowledge of programming.

MATLAB provides an interactive environment to perform engineering tasks for most current fields. You can also use your own toolboxes or use extra toolboxes built by others, which are freely available at the MATLAB File exchange at the following address:

http://http://www.mathworks.com/matlabcentral/fileexchange.

With its intuitive language, minimalistic programming commands, capability of one shot matrix operations, a wide range of toolboxes for most engineering fields, interactive platform and excellent visualization capabilities, MATLAB is regarded as an excellent and preferred tool for academic research and industrial applications.

Interface

After you have successfully installed MATLAB, you can open it by double clicking the icon or typing `matlab` in the terminal/run window. Depending on your machine, you will see something like Figure 1-1. The whole interface is known as the MATLAB Desktop. It consists of many components and windows which can be reorganized and enabled/disabled by mouse actions or via Desktop ➤ Layout menu options. Here are some of the important components one can find in a standard MATLAB Desktop:

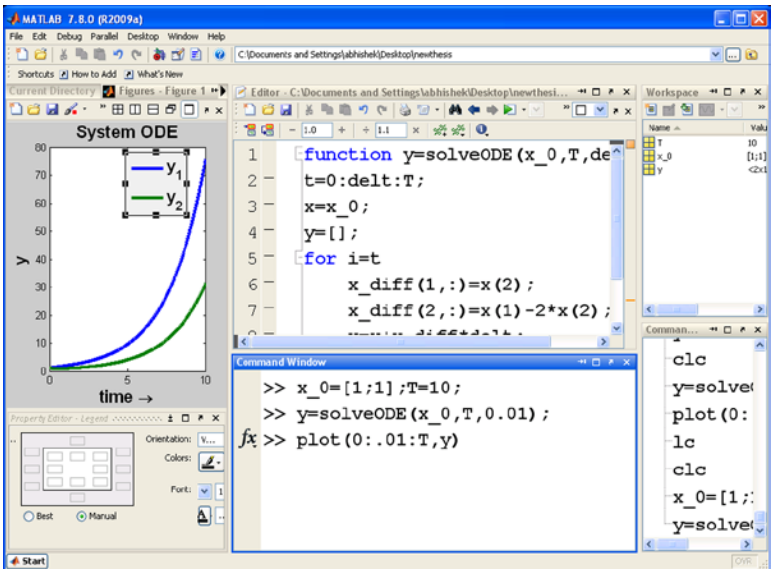

Figure 1-1. *MATLAB Desktop*

Command Window

The command window can be considered as a terminal. It is shown at the bottom of the middle column in Figure 1-1. Here, you can type your commands and run MATLAB programs (known as scripts). MATLAB displays >> as the command prompt.

Current Directory

The Current Directory in MATLAB represents a directory in which MATLAB looks for the functions and program files. In order to run a program, it must be in the current directory. The Current Directory window shows all the files and folders in the current directory. You can change the current directory or even add some folders to the MATLAB search path.

Workspace

The MATLAB workspace contains all the variables present and is shown at the top right of Figure 1-1. MATLAB programming differs from conventional programming languages in the sense that any variables created during a program execution remain, even after the program has executed, until you explicitly clear them or close the MATLAB session.

Figures

MATLAB uses figure windows to display any plot or visualization. This is shown in the leftmost column of Figure 1-1. We can open multiple figure windows using the figure command.

Command History

The command history window stores and displays all previous MATLAB commands issued in the command window. You can rerun any command by double clicking on the command window. We can also access the command via the keyboard, using the up and down arrow keys.

Editor

MATLAB provides an integrated editor to write commands and programs and execute them. The editor also can be used to debug programs. It is shown at the top middle of Figure 1-1.

Help Browser

MATLAB provides excellent documentation for all its functions with sufficient examples. To open MATLAB help, you can type doc or go to Help ➤ Product Help. To directly open help about a function (for example sin), we would type

```
doc sin;
```

Getting Started

In our first example, we will create a simple program to multiply two numbers. MATLAB programs are called scripts or also M-files because their extension is '.m'. To create a script named `myfirstprogram.m`, let us go to the command window and type

```
edit myfirstprogram
```

MATLAB will open an editor window with a blank file name. Type the following code

```
a=4;
b=3;
c=a*b;
disp(c);
```

Save this file and run the code by entering the following in the command window

```
myfirstprogram
```

When you press enter, you get the following output

```
c=
   12
```

Let us spend some time in understanding this simple program. We don't need any sophisticated program definitions or imports to run a simple program such as multiplication in MATLAB. In this program, we have defined two variables a and b. To define/assign a variable, we simply need to type

```
variablename= variablevalue;
```

We then multiplied them to store the answer in c and displayed it using the `disp` command.

The semicolon used to terminate each line is not necessary. If the semicolon is omitted, MATLAB will also display the value computed in the right-hand side of the statement. For example, typing

```
c=a*b;
```

will result in the computation of c without displaying the result, while typing

```
c=a*b
```

will result in the computation of c and also result in the display of the output in the command window. We can change the value of any variable by reassigning its value. For example

```
c=7;
```

will change the value of c to 7.

The first question which comes to our mind concerns the type of a. Is it stored as a double or an integer? First of all, all the variables stored in MATLAB are matrices. Secondly, all the numerical values are treated as double. So 4 is a 1×1 double numerical matrix. So if we compute the division of a by b using

```
c=a/b
```

we will get 1.33. However, the thing to remember is that even if a is a double number, MATLAB still knows that it can also be treated as an integer. This will be better understood in later chapters, when we try to index the matrix by providing integer indices.

Creating a Matrix

To create a matrix, we enclose all the elements inside [ ]. For example, to create a row vector containing the six elements 1 3 5 6 3 2, we can write

```
A= [1 3 5 6 3 2]
```

or

```
A=[1,3,5,6,3,2]
```

The comma or space here separates the elements into a single row and instructs MATLAB to construct a new column for the next element. Similarly we can construct a column vector with the same elements by typing

```
A=[1;3;5;6;3;2]
```

Here, each semicolon tells MATLAB to start a new row for the next element. Both comma and semicolon operators can also be used to join two matrices, for example

```
A=[4;5];
C=[2;A];
```

or

```
C=[2; [4;5]]
```

We can also mix these two operators to form a rectangular matrix.

```
A=[2 3 ; 4 5 ; 6 7]
B=[[2;4;6] , [3 ;5;7]]
C=[2 [3] ; [4;6] [5 7]]
```

All three of the above operations give the same 3×2 matrix. There is another operator ':', known as the colon operator, which means 'to'. For example, to create a matrix with elements from 0 to 100 with unit difference, we can type

```
A=[0:100];
```

To create a matrix with elements from 0 to 2π with step 0.01, we type

```
t=[0:.01:2*pi]
```

which can be interpreted as 'from 0 to 2*pi with step .01'. Remember that pi is a pre-defined constant in MATLAB.

Now you can write these commands in the editor window and save/run it as an M-file. But you can also type these commands directly into the command window. As we press enter after each command, that command will be executed and the result will be the same as if it were executed as a script in the editor window. The benefit of writing the commands in a script file is that it can be saved for future use, and it can be transferred among devices. However, the command window can be used to test or view variables quickly. There are some cases where we want to run a small command over some variables and we know that we will never use this command in the near future. In those cases, an M-file is not needed and the command window can be used instead.

Functions

A function is a simple set of instructions which accepts some inputs and returns outputs. There are a wide range of inbuilt functions in MATLAB for different application and they can be called anywhere with proper inputs. A function call generally has the following syntax

```
[output1 output2]= functionname (input1, input2)
```

Consider the function sin which takes a vector and computes the sine of each of its elements, returning a vector consisting of the values in the same order as the input. To view the syntax of the sin function we can type:

```
help sin
```

which displays a short description of the sin function and its syntax with some examples.

```
Sin:
Sine of argument in radians
Syntax
Y = sin(X)
Description
Y = sin(X) returns the circular sine of the elements of X. The sin function operates element-wise on
arrays. The function's domains and ranges include complex values. All angles are in radians.
```

After reading this, we can easily guess the format of the `sin` function. Now we can use this function as follows:

```
x=0:.01:2*pi;
y=sin(x);
```

Similarly, there is a wide range of functions such as log, exp, etc., which can be found at http://www.mathworks.com/help/matlab/functionlist.html. We will encounter many of these functions as we go. Two of the most interesting features of MATLAB are that these functions have very intuitive names and syntax, and their syntax can be easily found just by going to MATLAB help.

In MATLAB, we can also define our own functions. Suppose we want to define a function `myfun` which computes `y=log(x)*sin(x/4)` given the input x. We first create an M-file by typing

```
edit myfun
```

and then we type the following

```
function y= myfun(x)
y=log(x)*sin(x/4);
```

saving it to a file. Such a file is called a function file and has the same extension .m as a script. The first word of the first line of this code tells MATLAB that it is a function file. Everything before the equals sign tells MATLAB what variables to output and the word immediately after the equals sign is the function name by which MATLAB remembers this function. This name should match the name of the file. The name is followed by a list of all the input parameters inside parentheses. The rest of the code contains instructions to compute output variables from input variables.

To call this function, we write

```
x1 = 5
y=myfun(x1);
```

in any script, function or command window. Similarly, we can define a multiple input and output functions as

```
function [z p]= myfun(x,y)
z=log(x)*sin(y/4);
p=exp(x)*cos(z/4);
```

and this function can be called by

```
[a1 b1]=myfun(x,2);
```

If a function has no inputs, you can call this function by

```
out=myfunction()
```

or

```
out=myfunction
```

The Difference Between Functions and Scripts

Functions and scripts are both M-files, but there are a few differences between them. As we saw, the first line defines whether the M-file is a function or a script. A function has a particular set of inputs and outputs, while a script doesn't. A script can use all the variables created in the MATLAB workspace, but a function can only use the variables specially passed to it. After the execution is over, a script returns all the variables to the workspace while a function only returns the variables listed as outputs and deletes the rest of the variables. Scripts use the same copy of a variable in the MATLAB workspace and modifying any variable in any script will affect the original copy in the workspace while a function creates a new copy of all the variables when they are called and any modification to these variables doesn't affect the original variables. In other words, when a function is called from a workspace, the function call creates a new workspace, copies variables to it, computes outputs and returns these variables to the calling workspace, deleting its own workspace. You can also call a script from a function. In that case, the script will use the calling function's workspace and the variables created by this script call will also get deleted when the calling function finishes its execution.

As we saw, scripts, variables and functions (with no input) are called in the same way. For example, to display a variable `finalsum` we write

```
finalsum
```

and to run a script named `scrfinal` we write

```
scrfinal
```

Similarly, to multiply the variable `finalsum` by 2, we write

```
x=2*finalsum;
```

and if there is a function named `mypiconst` defined as

```
function x=mypiconst
x=3.14;
```

we can multiply the output of this function by 2 by writing

```
x=2*mypiconst;
```

What if there is a variable named `finalsum` and a script file with name `finalsum.m`? How does MATLAB know which one to use? The answer is that MATLAB first searches for the variable in its workspace and if it cannot find any matching variable then it searches for scripts or functions in the current directory. In other words, a variable shadows a script or function with the same name. This is also true for inbuilt functions. For example, we saw that sin is used to compute the sine of a variable. Let us define our own variable sin and then try to call the sin function

```
sin=4;
d=sin*3
y=sin(3);
```

We will get `d=12`, but the next line will cause an error because sin is treated as a variable here and it is trying to access the third element of the sin matrix. Here, (.) denotes indexing, which we will learn more about later. To see this explicitly, we can ask MATLAB about any variable name/function to determine which version MATLAB is using by typing

```
which sin
```

to which MATLAB displays

```
sin is variable.
```

To use the function sin again, we need to delete the sin variable first by typing

```
clear sin
```

To clear all the variables, we type

```
clear all
```

Now typing `which sin` results in

```
sin is inbuilt function.
```

This is crucial and we need to remember it, otherwise we would constantly run into errors or unexpected outputs. For example, in the above code, if we use

```
sin=4;
d=sin*3
y=sin(1);
```

it will run without error because y will be assigned the first element of the sin variable, which is 4. So instead of getting 0.8414, you will get `y=4`, and all subsequent computations will be incorrect, possibly without us realizing the mistake.

Special Matrices

We can also create some special matrices via inbuilt functions. For example, to create matrix of size 5×6 all entries of which are ones, we can write

```
A=ones(5,6);
```

Similarly zeros will create an all zero matrix, eye will create an identity matrix and rand will create a matrix whose entries are random values between 0 and 1.

Other Variable Types

We saw that MATLAB creates variables of type matrix. Until now, we have only seen variables containing numerical matrix values. In this section, we will meet a few other types of variables that MATLAB implements.

Character Variables

A character variable is a string containing characters. Remember that a single character is just a 1×1 character matrix. To define a string we write

```
A='strval';
```

which is equivalent to

```
A=['s' 't' 'r' 'v' 'a' 'l'];
```

To define multiline strings, we need to define each row with the same number of columns by including spaces. For example

```
A=['john';'joe'];
```

Cells

When a collection contains elements of different types, they can be represented using a cell array. A multiline string can be better represented using cells since different elements in a cell can have a different number of columns, hence strings with different lengths.

```
A={'name', 12};
A={'john','joe'};
```

Logical Variables

Logical or boolean variables can have only two values, true (1) or false (0). To create a logical matrix, we write

```
A=[true, false, true];
```

or

```
A=logical([1,0,1]);
```

Structures

A structure has many variables attached to it, indexed by fields. For example, a student is attached to variables declaring his name, class and cgpa. To construct a student structure, we write

```
student.name='John';
student.class=10;
student.cgpa=3.5;
```

which results in a structure matrix of size 1×1. To create a second element of the above matrix, we can write

```
student(2).name='Joe';
student(2).class=9;
student(2).cgpa=3.7;
```

Saving/Loading Variables

Since MATLAB deletes all its variables when it is closed, you can save all the variables in a data file for the next session by executing the following

```
save filename
```

Similarly we can load all the variables using

```
load filename
```

Plots

MATLAB also provides easy and simple plotting functions. To plot data, you need two vectors of the same size, one for the x axis and the other for the y axis. When we call plot with these two vectors, MATLAB will create a pair (x,y) by taking corresponding elements from these two vectors, plotting them on a Cartesian plane and connecting them by straight lines.

The following example shows how to plot a rectangle in MATLAB (see Figure 1-2)

```
x=[1 0 -1 0  1];
y=[0 1 0  -1 0];
plot(x,y);
```

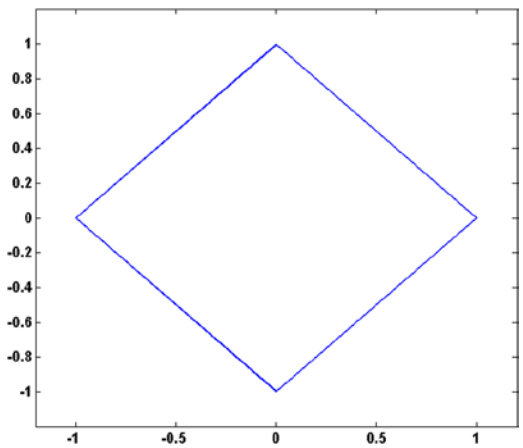

Figure 1-2. *Plotting a rectangle*

If the x values are very close to one another, the plot will look like a continuous plot (See Figure 1-3)

```
t=0:.01:2*pi;
y=sin(t);
plot(t,y);
```

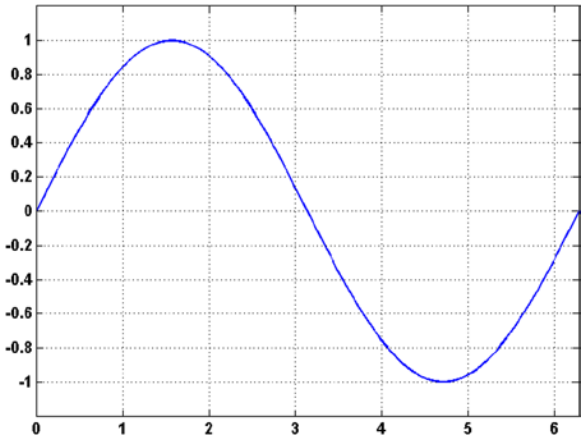

Figure 1-3. *Plotting a continuous function*

In later sections we will see other plotting functions in greater detail.