CHAPTER 1

Introduction to MATLAB

1.1 Introduction to Numerical Computing

With the advent of computers in the post World War II era, the need to simulate physical problems using this new tool led to the invention of numerical computing. Whereas analytical computation required pen, paper, and the human mind, numerical computation required a calculating device too. Successful implementation of computing devices to solve problems (especially involving repeated tasks) over a large array of data points was observed in many fields of science and engineering. For example, breaking enemy's secret codes, simulating nuclear reactions before nuclear explosions, etc. The scope further expanded to civilian purposes, such as designing and simulating waterways, dams, electric power stations, town planning, etc. All of these applications need to use an equation or systems of equation for a physical model representing a physical problem. There are two ways that one can approach these equations—using analytical and numerical techniques. We concentrate only on the numerical methods of solving equations using MATLAB in this book.

S. Nagar, *Introduction to MATLAB for Engineers and Scientists*,
https://doi.org/10.1007/978-1-4842-3189-0_1

As time progressed, various schemes to define mathematical functions—differentiation, integration, trigonometric, etc.—were written for digital computers. This involved digitization, which certainly introduces errors. Knowledge of errors and their proper nullification could yield valuable information quicker than analytical results. Thus, it became one of the most actively researched fields of science and continues to be one. The search for faster and more accurate algorithms continues to drive innovation in the field of numerical computing and enables humanity to simulate otherwise impossible tasks.

1.2 Tools for Numerical Computing

As the numerical methods progressed as an alternative to analytical methods, computer programming languages were increasingly being used to codify them for programmed investigations of simulations. A number of options [1] exist to perform numerical computation. Programming languages written to handle mathematical functions like FORTRAN, C, Python, Java, and Julia, to name a few, can be used to write algorithms for numerical computation.

1.2.1 The Need for Specialized Software

While all problems can be coded in programming languages, it's necessary to change the approach to computing, file management, etc. when the microprocessor platform or operating system changes. This hinders interoperability. Modern programming languages address some of these issues, but the need for specialized software for numerical computing—where predefined tools of numerical methods can be simply *called* as and when required and customized tools can be developed—was being felt in academia. A number of attempts were made in this direction.

1.2.2 The History of MATLAB

MATLAB was one such program and it was developed by Cleve Moler [2], who was a math professor at the University of New Mexico, teaching numerical analysis and matrix theory. As a PhD student, he initially wrote a lot of code in FORTRAN to solve systems of simultaneous linear equations involving matrix algebra, which ultimately he called MATrixLABoratory (MATLAB). As a professor he wished his students could use the new packages without writing FORTRAN programs.

Hence, in late 1970s, the first version of MATLAB came out (written in FORTRAN). There were 80 functions for performing calculations involving linear algebra problems. Further down the line, Jack Little and Steve Bangert reprogrammed MATLAB in C with additional features for producing a commercial version of the software. Together, all three of them founded The MathWorks [3] in California in 1984, which develops, maintains, and distributes MATLAB and its products worldwide. MATLAB has proven to be an excellent tool for numerical methods [4].

Over a period of time, so many tools and features have been added to the base package of MATLAB that, along with this rich set of libraries, the installation requirements run it is many GBs of data. MATLAB became tremendously popular in the scientific community. It is used by more than 5,000 universities worldwide. It is sometimes rightly termed the “language of engineering”. Cheap availability of digital computing resources propelled its usage in industry and academia to such an extent that virtually every lab needs MATLAB now.

1.3 Installation Requirements

MATLAB should be purchased from the official web site of MathWorks [3] or from an official distributer. The computer system requirements depend on the type and number of optional tools [5] installed with the base

MATLAB package. This book discusses the usage of the base MATLAB package. Hence, to have a good experience with your MATLAB software, use a laptop or workstation with 1GB RAM and any of operating systems—Windows, Linux, or MacOSX. Installation instructions are given with the product. The MATLAB environment is similar on all systems, so you need not worry about this while practicing with the book. This book has been tested for MATLAB R2017a version on the MacOSX 10.12 operating system.

1.4 Workspace

There are two ways to work within MATLAB. The first way is to work at the command line by writing one command at a time. The second method is to write a script (an `.m` file having a set of commands in a sequence) and run it from the command line by simply typing its name. For example, to run the `a.m` script file, you simply write the following at the command prompt:

```
>>a
```

The command prompt is represented by the symbol `>>` by default. You enter a command at the command prompt and then press the Enter key to execute the command. See Figure 1-1.

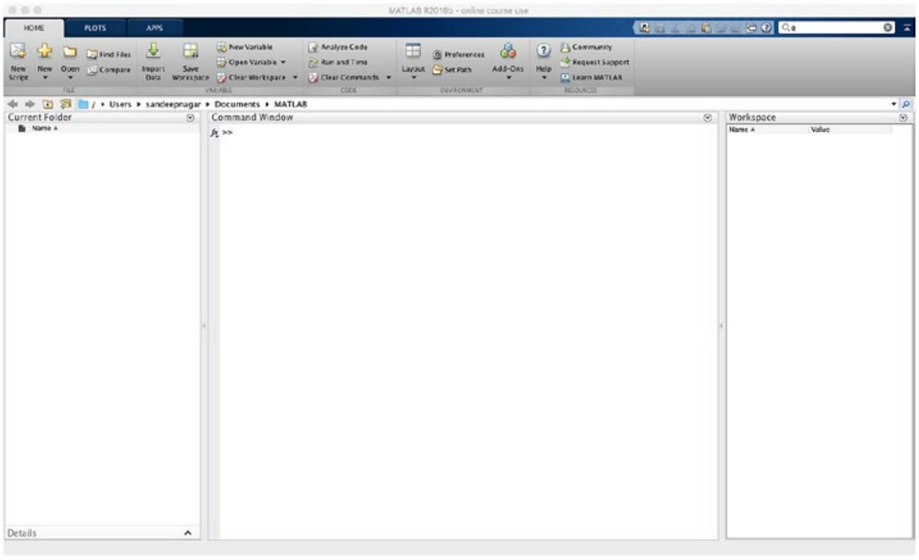

Figure 1-1. *MATLAB in action*

1.4.1 The REPL Principle

The MATLAB command line works on the principle of *REPL*, which stands for Read-Evaluates-Prints-Loop. When input is fed into the MATLAB command prompt, the Julia language:

- *Reads* what the user types
- *Evaluates* what it reads
- *Prints* out the return value after evaluation
- *Loops* back and does it all over again

All MATLAB commands are treated as expressions to be evaluated at REPL. Many programming environments, such as Python's interactive shell as well as the Jupyter notebook format, share the same approach. The new language called Julia also has a REPL and works in a similar fashion.

1.4.2 Calculator

In the simplest view, MATLAB works as a calculator with mathematical operators like multiplication (*), division (/), addition (+), subtraction (-), and exponentiation (^):

```
>> 3 + 5
ans = 8
>> 2 - 3
ans = -1
>> 3.0 * 5
ans = 15
>> 2 / 3
ans = 0.6667
>> format long
>> 2 / 3
ans = 0.666666666666667
>> format short
>> 2 / 3
ans = 0.6667
>> 2 % 3
ans = 2
>> 2 ^ 3
ans = 8
```

As seen in the previous example, when a command is fed into the command prompt `>>`, it is executed and an answer is given by displaying the results in the next line as `ans =`. To display more decimal digits in the result, you can use the `format long` command. By default, MATLAB works with the `format short` command.

1.4.3 Predefined Constants

```
>> pi
ans = 3.1416
>> i
ans = 0.0000 + 1.0000i
>> j
ans = 0.0000 + 1.0000i
>> Inf/Inf
ans = NaN
```

A number of physical constants are defined: `pi`, `e` (Euler's number), `i` and `j` (the imaginary number $\sqrt{-1}$), `inf` (Infinity), `NaN` (Not a Number, which results from undefined operations, such as `Inf/Inf`).

1.4.4 Common Mathematical Functions

```
>> abs(-10.034)
ans = 10.034
>> log10(10)
ans = 1
>> sin(10)
ans = -0.5440
>> cos(10)
ans = -0.8391
>> tan(10)
ans = 0.6484
>> asin(1)
ans = 1.5708
>> asin(10)
ans = 1.5708 + 2.9932i
>> acos(1)
```

```
ans = 0
>> acos(10)
ans = 0.0000 - 2.9932i
>> atan(1)
ans = 0.78540
>> atan(10)
ans = 1.4711
```

A number of predefined mathematical functions exist in MATLAB, including:

- Absolute value: `abs()`
- Logarithm: Natural logarithm `log()` and Base-10 logarithm `log10()`
- Trigonometric functions: `sin()`, `cos()`, and `tan()`. Arguments are taken in radians.
- Inverse-trigonometric functions: `asin()`, `acos()`, and `atan()`

When one works on the command prompt, it is often convenient to have a clear screen by getting rid of the previous command written at the command prompt. This is done using the command `clc`, which *clears the screen* by removing all inputs and outputs.

Complex calculations involving these functions and operations can be performed with ease, like the following

$$\sqrt{sin(10)^2 + cos(10)^2}$$

and

$$\frac{sin(10)}{\sqrt{cos(10)}}$$

```
>> sqrt(((sin(10))^2)+(cos(10))^2)
ans = 1
>> sin(10)/sqrt(cos(10))
ans = 0.0000 + 0.5939i
```

1.5 Self Learning and Getting Help

Covering all the functions available with MATLAB is beyond the scope of this book (or any other book!). To understand how a particular function needs to be used, you can use the `help` and `doc` commands. For example, typing `help exp` gives you detailed information about how this function should be used, whereas `doc exp` opens the official documentation page for the built-in function, `exp`.

```
>> help exp
exp     Exponential.
exp(X) is the exponential of the elements of X, e to the X.
For complex Z = X+i*Y, exp(Z) = exp(X)*(COS(Y)+i*SIN(Y)).

See also expm1, log, log10, expm, expint.

Reference page for exp
```

Whereas `help` is typically used by programmers to get a quick overview of usage for a particular built-in command, the `doc` is used to learn about MATLAB structures. The `doc` provides detailed descriptions of usage as well as useful examples. For example, typing `doc exp` on the MATLAB command prompt will open a new window, which will show the documentation for using the `exp` facility.

1.6 Variables

To store values temporarily, we use variables that store the value at a particular memory location and address it with a symbol or a set of symbols (called *strings*). For example, you can store the value of `1/10 * pi` as a variable `a` and then use it in an equation like this:

$$a^2 + 10\sqrt{a}$$

To perform this calculation:

$$\pi^2 + 10\sqrt{\pi}$$

```
>> a=1/10* pi
a =  0.3142
>> a^2 + 10* sqrt(a)
ans = 5.7037
```

Hence, the symbol = works as an assignment operator. It assigns the value on the right side to the variable named on the left side. Multiple assignments can be performed by using the comma (`,`) operator. Also, if you don't want to produce the results on-screen, you can suppress this by using the `;` operator.

```
>> a1 = 1, a2 = 10, a3 = 100
a1 = 1
a2 = 10
a3 = 100
>> a1 = 1, a2 = 10, a3 = 100;
a1 = 1
a2 = 10
>> a1 = 1; a2 = 10; a3 = 100;
>> a1
```

```
a1 = 1
>> a2
a2 = 10
>> a3
a3 = 100
```

1.6.1 Data Types

While assigning data to a variable, it is important to understand that data can be defined as a variety of *objects* defined by their *data types,* as follows:

- `logical`: This type of data stores Boolean values `1` or `0`, which can be operated by Boolean operators like `AND`, `OR`, `XOR`, etc.
- `char`: This type of data stores alphabetic characters and strings (groups of characters written in a sequence).
- `int8`, `int16`, `int32`, and `int64`: This type of data is stored as integers within 8 bits, 10 bits, 32 bits, and 64 bits, respectively. The size of the integer is given by its bit counts.

  Both `logical` and `char` are one byte (8 bits) wide.
- `uint8`, `uint16`, `uint32`, and `uint64`: This type of data stores unsigned integer data in 8, 16, 32 and 64 bits, respectively.
- `double` and `single`: This type of data is stored as double and single precision floating types, respectively. Decimal numbers are represented by floating point data types. Single precision occupies 4 bytes (32 bits) and double precision occupies 8 bytes (64 bits) to store the floating point numbers.

In the single precision system, 23 bits store the fraction bits (i.e., the numbers after the decimal point), 8 bits store the exponent (i.e., the numbers before the decimal point), and the 32nd bit is reserved for storing the sign.

In a double precision system, 52 bits store the fraction bits (i.e., the numbers after the decimal point), 11 bits store the exponent (i.e., the numbers before the decimal point), and the 64th bit is reserved for storing the sign.

Single and double precision matters when the precision of the result matters. In cases like GPS positioning for a projectile flying at high speeds, the results must be as precise as possible for greater accuracy of hit.

- `double complex` and `single complex`: Complex numbers have real and imaginary parts, which are stored separately. These numbers can be stored as single or double precision numbers using these data types.

1.6.2 Naming Conventions for Variables

There are some naming conventions for variables names, which must be respected to avoid errors.

- Names should not start with a number; however, numbers can be used anywhere afterward.
- Variable names are case sensitive.
- *Keywords* cannot be used as names.
- Names can include underscores (_).

While naming a variable, if you need to check that the name given is a keyword first, you can use the built-in function called `iskeyword(name)`. Simply typing `iskeyword()` produces a list of keywords, as shown here:

```
>> iskeyword()
ans =

20x1 cell array

'break'
'case'
'catch'
'classdef'
'continue'
'else'
'elseif'
'end'
'for'
'function'
'global'
'if'
'otherwise'
'parfor'
'persistent'
'return'
'spmd'
'switch'
'try'
'while'
```

1.6.3 List of Variables

While working on a project, it is useful to keep track of all the variables used in the project to avoid errors due to duplication of names. You can obtain a list of all variables by using the `who` and `whos` commands. Whereas the command `who` simply presents the list of variables in the workspace, `whos` presents the same with more information, like the size of the variable, the number of bytes used to store the variable, and the variable type.

```
>> who
Your variables are:

a    a1   a2   a3   ans

>> whos
Name       Size           Bytes  Class      Attributes

a          1x1                8  double
a1         1x1                8  double
a2         1x1                8  double
a3         1x1                8  double
ans       20x1             2462  cell
```

Note that the list of variables produced in this example represents the present state on my computer. If you have been working on projects other than practicing from the present book, all the variables defined in the present session will get reflected when you type `who` or `whos`. By using `who` and `whos`, you can keep track of memory requirements. Remember that judicious use of memory resources is important, especially on Raspberry Pi based systems. To wipe off the stored variables, you can use the `clear` command. It is also important to note that the variables list is session dependent. When you exit the session by closing MATLAB using the icon or using `exit`, the list of variables is erased from memory.

1.6.4 Global and Local Variables

A variable declared globally (i.e., within the main program) is known as a global variable, whereas a variable declared locally within a function (explained in later chapters) is known as a local variable. To define a global variable, you use the `global` declaration statement. Once defined, it remains the same irrespective of any new definition, unless you issue the `clear` command to clear the variable names and values from memory.

As seen, `a = 1` stays the same irrespective of the next definition, `a = 2`. When the command `clear` is issued at the command prompt, all variable names and values are flushed out of memory and the variable name can be used again. This time, if it is not defined as a `global` variable, its value can be changed repeatedly. The `isglobal()` command lets one check if a variable name has been defined as a global variable.

Global variables are used to define constants during numerical calculations. Suppose you want certain variables to change values, so you could make those unchanging values be global variables by giving the name of your choice. The predefined variables, like `pi`, `e`, etc., are defined in a similar manner.

1.6.5 The clear Command

As seen in the previous section, the `clear` command flushes out the variable names and their values from memory. It proves to be much more useful than that. Whereas `clear all` is the same as `clear`, it can also be used to selectively wipe out variables and their values. Simply type `help clear` to see a detailed view of its use, as shown in Listing 1-1.

Listing 1-1. The help clear Command

```
>> help clear
clear Clear variables and functions from memory.
clear removes all variables from the workspace.
clear VARIABLES does the same thing.
clear GLOBAL removes all global variables.
clear FUNCTIONS removes all compiled MATLAB and
MEX-functions.
Calling clear FUNCTIONS decreases code performance and is
usually unnecessary.
For more information, see the clear Reference page.

clear ALL removes all variables, globals, functions and MEX
links.
clear ALL at the command prompt also clears the base import
list.
Calling clear ALL decreases code performance and is usually
unnecessary.
For more information, see the clear Reference page.

clear IMPORT clears the base import list. It can only be
issued at the
command prompt. It cannot be used in a function or a
script.

clear CLASSES is the same as clear ALL except that class
definitions
are also cleared. If any objects exist outside the
workspace (say in
userdata or persistent in a locked program file) a warning
will be
```

```
issued and the class definition will not be cleared.
Calling clear CLASSES decreases code per formance and is usually unnecessary.
If you modify a class definition, MATLAB automatically updates it.
For more information, see the clear Reference page.

clear JAVA is the same as clear ALL except that java classes on the
dynamic java path (defined using JAVACLASSPATH) are also cleared.

clear VAR1 VAR2 ... clears the variables specified. The wildcard
character '*' can be used to clear variables that match a pattern. For
instance, clear X* clears all the variables in the current workspace
that start with X.

clear -REGEXP PAT1 PAT2 can be used to match all patterns using regular
expressions. This option only clears variables. For more information on
using regular expressions, type "doc regexp" at the command prompt.

If X is global, clear X removes X from the current workspace, but
leaves it accessible to any functions declaring it global.
clear GLOBAL -REGEXP PAT removes global variables that match regular
```

```
expression patterns.
Note that to clear specific global variables, the GLOBAL option must
come first. Otherwise, all global variables will be cleared.

clear FUN clears the function specified. If FUN has been locked by
MLOCK it will remain in memory. If FUN is a script or function that
is currently executing, then it is not cleared. Use a partial path
(see PARTIALPATH) to distinguish between different overloaded versions
of FUN. For instance, 'clear inline/display' clears only the INLINE
method for DISPLAY, leaving any other implementations in memory.

Examples for pattern matching:
clear a*                  % Clear variables starting with "a"
clear -regexp ^b\d2{3} $   % Clear variables starting with "b" and  %    followed by 3 digits

clear -regexp \d         % Clear variables containing any digits

See also clearvars, who, whos, mlock, munlock, persistent, import.

Reference page for clear
>>
```

Judicious use of the `clear` command proves to be a very powerful tool in managing memory requirements for a memory intensive numerical calculation.

1.7 Summary

MATLAB is a high performance language for technical computing. By using MATLAB as a simple calculator (using numbers and basic operations) as well as a complex calculator (using variables with complex functions), you can perform numerical calculations with ease. The learning curve for MATLAB is quite flat, owing to its simple and intuitive syntax. Whenever you become confused, the documentation for the particular commands can be easily accessed using the `help` command. The MATLAB GUI (Graphic User Interface) also provides an integrated environment for working with many different kinds of computational tasks, as shall be explored in upcoming chapters.

1.8 Bibliography

[1] https://en.wikipedia.org/wiki/List_of_numerical_analysis_software

[2] https://mathworks.com/company/newsletters/articles/the-origins-of-matlab.html

[3] https://www.mathworks.com

[4] https://in.mathworks.com/discovery/numerical-analysis.html

[5] https://mathworks.com/products/