School Of
Electrical, Computer &
Telecommunications Engineering
UOW

Training Lab

Introduction to MATLAB

Version: 1.1
Document Owner: Laboratory Manager, SECTE

Author: Dr. Muawiyath Shujau

Document Control:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>29/01/2013</td>
<td>M. Shujau</td>
<td>New resource for student learning</td>
</tr>
<tr>
<td>1.1</td>
<td>25/02/2013</td>
<td>S. Nikolic</td>
<td>Edited content</td>
</tr>
</tbody>
</table>
# MATLAB Training Guide

M. Shujau

## DOCUMENT CONTROL

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION TO MATLAB</td>
<td>8</td>
</tr>
<tr>
<td>1.1. GETTING TO KNOW MATLAB</td>
<td>8</td>
</tr>
<tr>
<td>1.2. MATLAB BASICS</td>
<td>8</td>
</tr>
<tr>
<td>1.2.1. Errors in Matlab</td>
<td>9</td>
</tr>
<tr>
<td>1.2.2. Managing the work space</td>
<td>10</td>
</tr>
<tr>
<td>1.2.3. Saving a Matlab session</td>
<td>10</td>
</tr>
<tr>
<td>1.2.4. Hierarchy of Operation</td>
<td>10</td>
</tr>
<tr>
<td>1.2.5. Matlab help</td>
<td>11</td>
</tr>
<tr>
<td>1.3. WRITING SCRIPTS AND FUNCTIONS</td>
<td>12</td>
</tr>
<tr>
<td>1.4. VECTORS AND ARRAYS</td>
<td>14</td>
</tr>
<tr>
<td>1.4.1. Creating a Vector</td>
<td>14</td>
</tr>
<tr>
<td>1.4.2. Indexing a vector</td>
<td>15</td>
</tr>
<tr>
<td>1.4.3. Creating a Matrix</td>
<td>16</td>
</tr>
<tr>
<td>1.4.4. Deleting rows and columns in a matrix</td>
<td>17</td>
</tr>
<tr>
<td>1.4.5. Concatenating two matrixes</td>
<td>17</td>
</tr>
<tr>
<td>1.4.6. Changing the dimension of a matrix</td>
<td>18</td>
</tr>
<tr>
<td>1.4.7. Matrix dimension</td>
<td>18</td>
</tr>
<tr>
<td>1.4.8. Matrix arithmetic operations</td>
<td>18</td>
</tr>
<tr>
<td>1.5. ARRAY ARITHMETIC OPERATIONS</td>
<td>19</td>
</tr>
<tr>
<td>1.6. MATRIX INVERSION</td>
<td>20</td>
</tr>
<tr>
<td>2. LOGICAL STATEMENTS, LOOPS AND MATLAB FUNCTIONS</td>
<td>21</td>
</tr>
<tr>
<td>2.1. IF – ELSE – END STATEMENTS</td>
<td>21</td>
</tr>
<tr>
<td>2.2. LOGICAL OPERATORS</td>
<td>23</td>
</tr>
<tr>
<td>2.3. “FOR LOOP” STATEMENTS</td>
<td>23</td>
</tr>
<tr>
<td>2.3.1. Nested “for loops”</td>
<td>24</td>
</tr>
<tr>
<td>2.4. THE “WHILE LOOP”</td>
<td>24</td>
</tr>
<tr>
<td>2.5. THE SWITCH STATEMENT</td>
<td>25</td>
</tr>
<tr>
<td>2.6. BREAK STATEMENTS</td>
<td>26</td>
</tr>
<tr>
<td>2.7. MATLAB FUNCTIONS</td>
<td>27</td>
</tr>
<tr>
<td>2.7.1. M Files Functions</td>
<td>27</td>
</tr>
<tr>
<td>2.7.2. Debugging M Files</td>
<td>29</td>
</tr>
<tr>
<td>3. CHARACTER STRINGS AND CELL ARRAYS</td>
<td>30</td>
</tr>
<tr>
<td>3.1. CHARACTER STRINGS</td>
<td>30</td>
</tr>
<tr>
<td>3.1.1. Concatenating, slicing and flipping Strings</td>
<td>30</td>
</tr>
<tr>
<td>3.1.2. Arithmetic and Logical Operations</td>
<td>32</td>
</tr>
<tr>
<td>3.1.3. Format conversion</td>
<td>32</td>
</tr>
</tbody>
</table>

---

MATLAB Training Guide

M. Shujau

Page 3
3.2. CELL ARRAY AND STRUCTURES ................................................................. 34
  3.2.1. Cell Array ..................................................................................... 34
  3.2.2. Structures ................................................................................... 35

4. FILE INPUT / OUTPUT AND PLOTTING .................................................. 38
  4.1. WORK SPACE I/O ........................................................................... 38
  4.2. TEXT FILES ................................................................................. 38
  4.3. READ AND WRITING DATA TO SPREAD SHEETS IN MATLAB ........................................... 40
  4.4. READING AND WRITING AUDIO AND IMAGE FILES ......................................................... 42
  4.5. PLOTTING ...................................................................................... 43
    4.5.1. Creating a simple 2D plot .......................................................... 43
    4.5.2. 3D plots ............................................................................... 48
Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Plot of Sin(x)</td>
<td>43</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Example of Plot Color and Markers</td>
<td>44</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example of Title and Axis Labels</td>
<td>44</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Example of Hold Function</td>
<td>45</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Example of Four Subplots</td>
<td>46</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Example of a Single Plot</td>
<td>46</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Example of Two Plots</td>
<td>47</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Example of Four Plots</td>
<td>48</td>
</tr>
<tr>
<td>Figure 9</td>
<td>3D Line Plot Example</td>
<td>49</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Mesh Plot Example</td>
<td>50</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Surface Plot Example</td>
<td>50</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Contour Plot Example</td>
<td>51</td>
</tr>
</tbody>
</table>
## Table of Tables

Table 1: Order of arithmetic operation ................................................................. 11  
Table 2: Matlab pre defined constants ................................................................. 12  
Table 3: Command for Array operations ............................................................ 19  
Table 4: Logical Operators Available in Matlab ................................................. 23  
Table 5: Command useful in loops ..................................................................... 27  
Table 6: Differences between functions and scripts ........................................... 28  
Table 7: Commands for Data Type conversion .................................................... 33  
Table 8: Commands for Reading and writing Image files .................................... 42  
Table 9: Command for Reading and writing audio files ...................................... 42
Introduction

This guide is an introduction to the Matlab Programming environment which is an essential tool in most courses in SECTE. It is expected that through these notes students will become familiar with the Matlab programming environment and will gain necessary skills needed for SECTE subjects which require Matlab.

Acknowledgements and thanks are given to Pearson Education Inc. and the authors of Engineering Computation with Matlab for the permission to use the text book resources for education purposes.

Resources

Software
MATLAB software can be found in all computers labs within SECTE and many computer laboratories around the university.

Student version of the MATLAB software can be purchased from the MathWorks website:
http://www.mathworks.com/academia/student_version/
1. Introduction to MATLAB

Matlab is a powerful language for technical computing. The synonym MATLAB stands for MATrix LABoratory. Matlab combines computation, visualization and programming environments all in one platform. It has a complex data structure and has its own debugging and editing tools and supports object oriented programming. It has built in routines which enable fast and easy computation of otherwise complex calculations such as Fast Fourier Transforms (FFT). Results can be graphically visualized easily with easy to understand and easy to use graphics commands. These sets of routines are grouped and packaged into tool boxes; there are tool boxes for signal processing, control theory and many others from different areas of engineering.

The guide presented here is designed for students with no prior experience with Matlab to become familiar with Matlab and learn to use Matlab to solve application problems. The objective of this guide is to enable students to quickly learn the basics steps. It is important that students understand that learning a new language requires practice, hence the best way to learn fast is "trying it yourself".

1.1. Getting to Know Matlab

When you open Matlab by double clicking the Matlab icon on your desktop the Matlab desktop will appear. The Matlab desktop has four other windows in it which are:

1. Command window – to enter the commands for Matlab to process
2. Command history – history of commands entered (shows date and time)
3. Workspace – shows the variables being used by the user
4. Current folder – the folder where the current work is saved and retrieved from

Let us start using Matlab to do some simple tasks, the commands are entered in to the command window at the command prompt (>>).

1.2. Matlab Basics

Let us start with a simple calculation, if we want to add two numbers let’s say 5 and 4, at command prompt we type

```
>> 5+4
ans =
 9
```

Since we didn’t assign an output variable, Matlab assigned the output to the default variable ans which is answer for short. If we decide to assign the output for the same operation to variable x then
>> x = 5+4
x =
   9
Since variable x is assigned the value 9 from previous operation, 2x will result in
>> 2*x
ans =
  18.0000
The variables in Matlab can be re-assigned and in this example the output from the operation 2x will be
assigned to x
>> x = 2*x
x =
  18.0000
It is helpful to use meaningful names for variables as this would make it easy to follow when writing scripts.
Let’s try another example, this time we will create a variable called “number” and assign the value 10 and
then divide the value by 2 and assign the value to another variable “output”.
>> number = 10
number =
   10
>> output = number/2
output =
   5
Here, every time a step is completed a result is seen. If we do not wish to see the intermediate results we
can suppress the numerical output by putting a semi colon (;) at the end of the line.
>> number = 10;
>> output = number/2
output =
   5
This time the output from assigning the value 10 to “number” is suppressed. If you wish to see the value
assigned to “number” it can be seen in the workspace window. All the variables that are in use are displayed
in the workspace window.

1.2.1. Errors in Matlab

What happens when an expression is entered incorrectly to Matlab? Let say if we want to calculate 2x from
previous example and we left out the multiplication sign “*”
>>2x
??? 2x
     |
Error: Unexpected MATLAB expression.
Matlab responses with an error message. To correct the error we can recall the previous typed command using the up arrow key “↑” and when the command is displayed in the command prompt it can be corrected.

1.2.2. Managing the work space

It is very important that the variables that are no longer used should be released since every time a variable is assigned, a system memory is assigned to that variable. The command used to clear the variables is “clear” or “clear all”. To clear a specific variable “clear variable name” can be used. Let us consider our previous example

```matlab
>> clear
>> x
??? Undefined function or variable 'x'.
```

Here we have clear all the variables assigned. Hence there is no variable x anymore.

```matlab
>> x = 5;
>> y = 6;
>> clear x
>> x
??? Undefined function or variable 'x'.
>> y
```

Here we have assigned two variables x and y and we needed to clear x only. The variable x is cleared while variable y remains. To display all the variables used, command “who” or “whos” can be used. Command “whos” gives more detailed information which include size and space allocation.

1.2.3. Saving a Matlab session

If we wish to save our Matlab session, we can use the “diary” or “diary filename” command. The later will save the session under the “filename”. This will save all the input and output in the Matlab session. There are two other commands that are used with the “diary” command which are “diary on” and “diary off” which can be used to start recording a session and end recording a session.

1.2.4. Hierarchy of Operation

Let us consider another example,

```matlab
>> 2+3*5
ans =
    17
```

If we now introduce parentheses to the above equation such that the expression reads (2+3)*5 we would get
By adding the *parentheses* the two expressions have given different results: 17 and 25. The order in which Matlab executes arithmetic operation is *exponentials* first, followed by *multiplications* then *division* and finally *addition* and *subtraction*. In this example it is seen that by adding the *parentheses* the order of the operation has changed. Here, the *parentheses* is used to overrule the priority which can be useful in some complex expression to avoid ambiguity. In Matlab the order in which arithmetic operations are evaluated is given in Table 1.

<table>
<thead>
<tr>
<th>Order</th>
<th>Mathematical operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>Parentheses are evaluated first, starting from the inner most parentheses and working outwards</td>
</tr>
<tr>
<td>2nd</td>
<td>All exponentials from left to right</td>
</tr>
<tr>
<td>3rd</td>
<td>All multiplications and divisions from left to right</td>
</tr>
<tr>
<td>4th</td>
<td>All additions and subtractions from left to right</td>
</tr>
</tbody>
</table>

Table 1: Order of arithmetic operation

Let's us look at a different example:

\[
\frac{1}{4+5^2} + \frac{4}{5} \times \frac{6}{7}
\]

In Matlab

\[
\text{>> 1/(4+5^2)+4/5*6/7}
\]

\[
\text{ans =}
\]

\[
0.7202
\]

without the parentheses

\[
\text{>> 1/4+5^2+4/5*6/7}
\]

\[
\text{ans =}
\]

\[
25.9357
\]

The results are very different with and without parentheses. It is very important that the order of arithmetic operation is followed correctly to avoid ambiguity.

1.2.5. Matlab help

Matlab contains many pre-defined mathematical functions which can be seen by typing the command “help elfun” for elementary functions and command “help specfun” for special functions. At this point it would be good idea to look at Matlab help commands. Matlab help command is very useful in finding the syntax of a
function and how a function can be used and what other functions are related to a specific function. As an example we want to find about “cos” function:

```
>> help cos
COS Cosine of argument in radians.
COS(X) is the cosine of the elements of X.
see also acos, cosd.
Overloaded methods:
    sym/cos
Reference page in Help browser
    doc cos
```

Here Matlab has displayed all the information about “cos” function and other functions that are related to “cos” function in Matlab. It is worth noting the “doc cos” link which opens the online Matlab help manual, which is much more comprehensive and gives examples of how to use the function. Another useful help tool is the “lookfor” command which gives quick summary information about the function.

In addition to pre-defined functions, Matlab has many pre-defined constants such as:

<table>
<thead>
<tr>
<th>Constant</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>π = 3.14159……..</td>
</tr>
<tr>
<td>i and j</td>
<td>±1</td>
</tr>
<tr>
<td>Inf</td>
<td>infinity</td>
</tr>
<tr>
<td>NaN</td>
<td>Not a number</td>
</tr>
</tbody>
</table>

Table 2: Matlab pre defined constants

1.3. Writing Scripts and Functions

A script lets you save and reuse set of commands without retyping them in the command window. A script consists of combinations of executable Matlab commands. The script files have a file name extension “.m” and are called “m” files. The script files can be used as a series of commands or can be used as functions which can accept input arguments and produce one or more output arguments.

Let us look at some examples of scripting in Matlab.

Example 1:

Let the system of equations be:

\[
\begin{align*}
2x + 3y - z &= 4 \\
x + 3y + 4z &= 3 \\
x + y + z &= 1
\end{align*}
\]
We want to find the solution, named \( s \), for this system of equations

- Use the Matlab editor to create a file: File → New → M-file.
- Enter the command statements into the M-file:

\[
A = \begin{bmatrix}
2 & 3 & 1 \\
1 & 3 & 4 \\
1 & 1 & 1
\end{bmatrix}; \\
b = \begin{bmatrix}
4 \\
3 \\
1
\end{bmatrix}; \\
s = A \backslash b
\]
- Save the file, test.m.
- Run the file, by typing:

\[
>> \text{test}
\]

\[
s = \\
-0.2000 \\
1.6000 \\
-0.4000
\]

This is a simple script where we have solved the system of equations to find the value of \( x, y \) and \( z \). The problem here is that there is no way to change any of the input arguments other than to physically edit the script, hence has limited use. Let’s say we want to calculate the roots of quadratic equation and want to make it such that it can be reused by other students and in other “m” files that we write. This requires writing a function in Matlab. The function will have 3 inputs and two outputs. Let’s see how we can achieve this with an example.

Example 2
The general form of a quadratic equation is:

\[
a x^2 + b x + c = 0
\]

with the roots of equation given by

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

We want to create a function that will accomplish this:

- Use the Matlab editor to create a file: File → New → M-file.
- Declare the function header and then enter the command statements
- It is very important NOT to give function names already used in Matlab or function names that may have been used before. Always try to give a unique function name.

```matlab
function [r1,r2] = quadratic_test(a,b,c)
    % function to calculate the roots of x
    % help is here
    r1 = (-b+(sqrt(b^2-4*a*c)))/(2*a);
    r2 = (-b-(sqrt(b^2-4*a*c)))/(2*a);
end
```
- Save the function in the current directory and then call the function in the command line or inside another function as follows:

```matlab
>> [r1,r2] = quadratic_test(5,6,1)
```

```
r1 =
 -0.2000
r2 =
 -1
```

The line below the function header is the description of the function and the lines below it are the help information that you provide for your function. In our case if we say help quadratic_test at the command line we should see these lines.

```
>> help quadratic_test
function to calculate the roots of x
help is here
```

### 1.4. Vectors and Arrays

The most basic element is Matlab is a Matrix. A matrix is 2 dimensional arrays consisting of \( m \) rows and \( n \) columns. First we will look at vectors which are a special case of matrixes. There are two types of vectors, namely row vector which is a matrix of \((1\times n)\) dimension and column vector which is a \((m\times 1)\) matrix.

#### 1.4.1. Creating a Vector

The members of a vector in Matlab are enclosed by a square bracket and separated by spaces or commas (,). As an example let us form a row vector “rvec”:

```
>> rvec = [ 1 2 3 4 5]
```

```
rvec =
 1 2 3 4 5
```

```
>> rvec = [ 6,7,8,9,0]
```

```
rvec =
 6 7 8 9 0
```

A column vector “cvec” can be formed in a similar way but instead of spaces or commas separating the members by semi-colons ;:

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

```
cvec =
 1
 2
 3
 4
 5
```
A row vector can be converted to a column vector using the transpose command denoted by an apostrophe ('). Let us convert our row vector “rvec” to a column vector:

\[ \text{cvec} = \text{rvec}' \]

\[
\begin{array}{ccccccc}
6 & 7 & 8 & 9 & 0 \\
\end{array}
\]

### 1.4.2. Indexing a vector

The members of the vector are indexed according to their position in the vector, rvec(1) is the first element in the vector, rvec(2) is the second element and so forth. It is important to understand that in Matlab the indexing starts from 1 not from 0. If we want to access let’s say the 3rd element of our row vector then:

\[
\text{rvec}(3)
\]

\[
\begin{array}{c}
\text{ans} = \\
3 \\
\end{array}
\]

Here we have accessed the 3rd element of our row vector. In addition to accessing individual elements we can access blocks of elements by using the colon (:) notation. For example to get the last three elements:

\[
\text{rvec}(3:5)
\]

\[
\begin{array}{cccc}
3 & 4 & 5 \\
\end{array}
\]

Or we can use the “end” statement to get 3rd through to the last element as before.

\[
\text{rvec}(3:end)
\]

\[
\begin{array}{cccc}
3 & 4 & 5 \\
\end{array}
\]

To create a vector which is too large to be entered by hand we can use the colon operator or the “linspace” command (linear spaced vector).

\[
\text{vec} = 1:1:10
\]

\[
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array}
\]

A row vector with 10 elements with spacing of 1 is created. Here we do not have the control over how many elements are in the vector. By using the “linspace” command we can create a linear spaced vector between (and including) two numbers we want. For example:

\[
\text{vec} = \text{linspace}(1,10);
\]

gives 100 points (default value) linearly space between 1 and 10. By introducing the number of points in to “linspace” command we can get any number of points we want. For example:
>> vec = linspace(1,10,5)
vec =
   1.0000   3.2500   5.5000   7.7500  10.0000
gives five points between 1 and 10.

1.4.3. Creating a Matrix

A matrix has multiple rows and columns. To create a matrix in Matlab the following steps are required.

- Begin and end with square brackets [ ]
- Elements in rows are separated by space or commas
- Semi colons are used to separate rows

Let us now look at an example of a matrix

>> Mat = [1 2 3;4 5 6;7 8 9]
Mat =
1     2     3
4     5     6
7     8     9

To view a particular member in matrix we use Mat (m,n). As an example, let us view the element in the 1st row 2nd column.

>> Mat(1,2)
an =
   2

If we want to change an entry in the matrix we can use the above indexing to do so. Here, we want to change the entry in the 2nd row 2nd column from 5 to 10:

>> Mat(2,2) = 10
Mat =
1     2     3
4  10     6
7     8     9

If we want to select multiple elements in rows or columns we can use the colon operator to do so.

>> Mat(1:3,1)
an =
1
1
7
>> Mat(1,1:3)
an =
1     2     3
Similarly if we want to select part of the matrix we can use the colon operator.

```matlab
>> Mat(1:2,1:3)
ans =
    1     2     3
    4    10    6
```

An empty matrix can be created using the empty square brackets `[]`. This matrix by default is 1 by 5.

```matlab
>> empty = [];
>> size empty
ans =
    1     5
```

### 1.4.4. Deleting rows and columns in a matrix

A row or a column can be deleted by using the square brackets `[]`:

```matlab
>> Mat(:,3) = []
Mat =
    1     2
    4    10
    7     8
```

To add new values to the deleted row

```matlab
>> Mat = [Mat(:,1),Mat(:,2),[3;6;9]]
Mat =
    1     2     3
    4    10    6
    7     8     9
```

### 1.4.5. Concatenating two matrixes

In Matlab two arrays can be concatenated (combined) easily as follows:

Horizontally

```matlab
>> MatA = [1,2,3,4;5,6,7,8];
>> MatB = [9,10,11,12;13,14,15,16];
>> [MatA MatB]
ans =
    1     2     3     4     9    10    11    12
    5     6     7     8    13    14    15    16
```
Vertically

```
>> [MatA;MatB]
ans =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

1.4.6. Changing the dimension of a matrix

If we wish to change the dimensions of the array we can use the “reshape” function. As an example:

```
MatA =
     1     2     3     4
     5     6     7     8
>> reshape(MatA,1,8)
ans =
     1     5     2     6     3     7     4     8
```

1.4.7. Matrix dimension

To get the size of a matrix the function “size” is used as follows:

```
>> [m,n]= size(Mat)
 m =
     3
 n =
     3
```

1.4.8. Matrix arithmetic operations

Matlab allows arithmetic operation “+”, “-“, “*” and “^” to be performed on matrices. The following rules of matrix algebra are used in Matlab.

- A+B or B+A is true if the both matrixes A and B are of same size.
- A*B is true if number of columns of A equals number of rows of B
- A^2 is true if A is a square matrix and equals A*A
1.5. Array arithmetic operation

Array arithmetic operations are done on an element by element basis. This is different from the matrix operation described before. The period character “.” is used to indicate the array operation. For array operations the matrices have to be of the same size. The command for the array operations are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>.*</td>
<td>Element by element multiplication</td>
</tr>
<tr>
<td>./</td>
<td>Element by element division</td>
</tr>
<tr>
<td>.^</td>
<td>Element by element exponentiation</td>
</tr>
</tbody>
</table>

Table 3: Command for Array operations

Let us look at some examples:
```matlab
>> A = [4 4 4; 6 6 6; 8 8 8];
>> B = [2 2 2; 3 3 3; 4 4 4];
>> A*B
ans =
   36   36   36
   54   54   54
   72   72   72
>> A.*B
ans =
    8    8    8
   18   18   18
   32   32   32
>> A/B
Warning: Matrix is singular to working precision.
ans =
   NaN   NaN   NaN
   NaN   NaN   NaN
   NaN   NaN   NaN
>> A./B
ans =
    2    2    2
    2    2    2
    2    2    2
```

Here we can see the difference in the results of performing matrix and array operations.
1.6. Matrix Inversion

Let us consider a simple matrix A.

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]

The inverse of matrix A by hand is:

\[
A^{-1} = \frac{1}{9} \begin{bmatrix}
-16 & 8 & -1 \\
14 & -7 & 2 \\
-1 & 2 & -1
\end{bmatrix}
\]

If we use Matlab to calculate the inverse of matrix A:

```
A =
1  2  3
4  5  6
7  8  0
>> inv(A)
ans =
-1.7778  0.8889 -0.1111
1.5556 -0.7778  0.2222
-0.1111  0.2222 -0.1111
```

The answer is exactly the same. The determinant of a matrix in Matlab can be found by:

```
>> det(A)
ans =
27
```
2. Logical Statements, Loops And Matlab Functions

Like all other programming languages Matlab has decision making structures. In Matlab there are 4 decision making structures, which include “for loops”, “while loops”, “switch statements” and “if - else - end statements”. These decision making structures are usually used in “scripts” and in “m files”.

2.1. If – else – end statements

Like all other programming languages, Matlab offers “if” logical statements. There are 3 different ways in which the “if” structure can be used, which are:

- if (logical statement)
  ....
  end
- if (logical statement)
  ....
  else
  ....
  end
- if (logical statement)
  ....
  elseif (logical statement)
  ....
  else
  ....
  end

Let us look at some examples:

Example 1:
Let us consider the example in section 1 for solving the quadratic equations. When we wrote the “m file” we did not take in to account the number of roots and the nature of the roots. The “discriminant” \(b^2 - 4ac\) of a quadratic equation tells us the nature of the roots, which are:

- if the discriminant is zero there is one distinct root
- if the discriminant is negative then there are no real roots, rather two complex roots
- if the discriminant is positive then there are two distinct real roots

Now let us incorporate these conditions into our previous example:
function [r1,r2] = quadratic_test(a,b,c)
    % function to calculate the roots of x
    % help is here
    discriminant = b^2-4*a*c;

    if discriminant == 0
        disp('The equation has one real root')
    elseif discriminant < 0
        disp('The equation has no real roots')
    else
        disp('The equation has real roots')
    end

    r1 = (-b+(sqrt(discriminant)))/(2*a);
    r2 = (-b-(sqrt(discriminant)))/(2*a);
end

Example of case where there is one root

>> [r1,r2] = quadratic_test(4,4,1)
The equation has one real root
    r1 =
    -0.5000
    r2 =
    -0.5000

Example of case where there is no real roots

>> [r1,r2] = quadratic_test(1,0,0.5)
The equation has no real roots
    r1 =
    0 + 0.7071i
    r2 =
    0 - 0.7071i

Example of case where there is real root

>> [r1,r2] = quadratic_test(5,6,1)
The equation has real roots
    r1 =
    -0.2000
    r2 =
    -1
2.2. Logical Operators

A logical operator compares two variables to determine if the comparison is “true or “false”.

The logical operators available in Matlab are listed in the table below:

<table>
<thead>
<tr>
<th>operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>~=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>~</td>
<td>Logical NOT</td>
</tr>
</tbody>
</table>

Table 4: Logical Operators Available in Matlab

2.3. “for loop” statements

A for loop statement executes a command or a set of command for a fix number of times. In Matlab the syntax of for loop statements are as follows:

FOR variable = expression
    statement, ..., statement
END

Let us look at an example where a for loop statement can be used.

The Fibonacci sequence is defined as $f_1 = 1$, $f_2 = 1$ and $f_n = f_{n-1} + f_{n-2}$. We want to find the 50th element of Fibonacci sequence. Since we need to repeat the calculation 50 times using for loop is ideal. The Matlab code is as follows:

```matlab
function [ele] = fibonacci_test(n)
% this function calculates the nth Fibonacci element
% ele is nth Fibonacci element

f(1:n) = zeros; % create an array of zeros, n long
f(1) = 1; % first Fibonacci element
f(2) = 1; % second Fibonacci element
```
for \( j = 3:n \)
   \[ f(j) = f(j-1)+f(j-2); \] \% next element
end

\[ \text{ele} = f(n); \] \% nth Fibonacci element
end

\[ [\text{ele}] = \text{fibonacci} \_\text{test}(50) \]
\[ \text{ele} = 1.2586e+10 \]

### 2.3.1. Nested “for loops”

In the above example we have only used a single for loop but in reality we can use multiple for loops inside one another, this is called nested for loops. We will look at an example of nested for loop next.

Let us assume we have a 4 by 4 matrix “mat” and we would like to create a matrix “new\_mat” from “mat”.

\[
\begin{align*}
\text{>> mat} &= \begin{bmatrix} 1 & 1 & 1 & 1; 2 & 2 & 2 & 2; 3 & 3 & 3 & 3; 4 & 4 & 4 & 4 \end{bmatrix}; \\
\text{>> [rows col]} &= \text{size(mat)}; \% \text{size of mat} \\
\text{>> new\_mat} &= []; \% \text{create an empty matrix called new\_mat} \\
\text{>> for } i &= 1:col \\
\text{for } j &= 1:rows \\
\text{\hspace{0.5cm}} &= \text{mat}(j,i); \% \text{assign the values from mat to new\_mat} \\
\text{end} \\
\text{end} \\
\text{>> new\_mat} \\
\text{new\_mat} = \\
\begin{bmatrix}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3 \\
4 & 4 & 4 & 4
\end{bmatrix}
\end{align*}
\]

### 2.4. The “while loop”

A while loop continues until the set condition is satisfied. The syntax for a while loop has the form:

```
WHILE expression
   statements
END
```
The statements will be executed as long as the statement is true. It is very important to make sure that the statements in side a **while loop** is well defined as if this is not the case then the looping may continue indefinitely. If this occurs, the execution can be stoped by pressing **Ctrl and C** together. Let us look at an example of **while loop**.

Let us look at the example of Fibonacci sequence, we will change the function to use a **while loop** instead of a **for loop**.

```matlab
function [ele] = fibonacci_test(n)
    % this function calculates the nth Fibonacci element
    % ele is nth Fibonacci element

    f(1:n) = zeros; % create an array of zeros, n long
    f(1) = 1; % first Fibonacci element
    f(2) = 1; % second Fibonacci element
    j = 3; % the starting point
    while (j<=n)
        f(j) = f(j-1)+f(j-2); % next element
        j = j + 1; % increment j
    end
    ele = f(n); %nth Fibonacci element
end
```

The **while loop** requires that the variable j be incremented manually whereas in the **for loop** this was done automatically.

### 2.5. The switch statement

The switch statements allow execution of one set of statements from several choices. Each of these choices is a case. When a case is true then the statements in that case is executed. Let us look at an example:

```matlab
function test_switch(a)
    % example of switch statement
    temp = a;
    switch temp
        case 1
            disp('the input is 1')
        case 2
            disp('the input is 2')
        otherwise
            disp('the input is another number')
    end
end
```
>> test_switch(1)
    the input is 1
>> test_switch(2)
    the input is 2
>> test_switch(3)
    the input is another number

2.6. Break Statements

Unlike C++ the switch statements in Matlab does not require a break statement at the end. But Break is used in Matlab to terminate a loop from executing and exit. Let’s look at a simple example of break in Matlab. In this example we have an infinite while loop that keeps on looping until the correct number is entered by the user. Once the correct number is entered the break statement terminates the while loop.

function test_break
    % example of infinite loop and break
    while 1 % goes into a infinite loop
        in = input('Enter a value '); % get an input
        if in == 0.5 % if the input is the correct value
            break % then break
        end
        disp('Try again');
    end
    disp('we are out of the loop'); % once out of the loop say that

>> test_break
    Enter a value 0.1
    Try again
    Enter a value 0.2
    Try again
    Enter a value 0.3
    Try again
    Enter a value 0.4
    Try again
    Enter a value 0.5
    we are out of the loop
There are some other commands that are useful when using loops these are tabled below.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue</td>
<td>To skip the remaining statements in the loop</td>
</tr>
<tr>
<td>Return</td>
<td>Return to invoking function</td>
</tr>
</tbody>
</table>

Table 5: Command useful in loops

### 2.7. Matlab Functions

In the introduction we have looked at scripts and functions briefly. In this section we take a closer look at the functions. A script file is an external file that contains set of Matlab instructions which are saved with “.m” extensions and are called M files. M files can be a simple set of commands in a file or more complex function that can accept arguments and can produce outputs.

#### 2.7.1. M Files Functions

Functions are routines that accept input arguments and return output arguments. Each M file function has its own area of work space, separated from Matlab’s own workspace. We have already seen how a function is created in Matlab. Here we will focus on debugging of M files. Let’s recall our example of M file in section 1. Before we do that lets take a look at the general form of a function.

The basic parts of a function are shown in this simple function.

```
function f = exp(n,m)  
% EXP(N,M) returns NM. 
% Compute NM. 
f = n^m;  
end
```

The first line (1) of the M-file always have to have the key word “function” which gives the function name and order of arguments. In this case we have two input arguments and one output argument. The numbers of input and output arguments are not limited. If more than one output argument is needed a square bracket “[ ]” is used. The lines (2) and (3) are the description and help lines. We can put as many help and description lines as we want. From line four (4) the body of the function starts. Here we put all the Matlab commands we want to execute. Finally line five (5) the “end” command tells Matlab the end of the function. This is useful when there are more than one function in a single m file.

All the function names have to begin with a letter and must be less than 63 characters. And the text file will be saved with the function name. In this case this function will be called “exp.m”. There are several differences between functions and scripts, the major differences are tabulated below.
### Script vs. Function

<table>
<thead>
<tr>
<th>Script</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not accept input or return output arguments</td>
<td>Accept input and return output arguments</td>
</tr>
<tr>
<td>Store variables in a workspace that is shared with other scripts</td>
<td>Store variables in a workspace internal to the function.</td>
</tr>
<tr>
<td>Are useful for automating a series of commands</td>
<td>Are useful for extending the Matlab language for your application</td>
</tr>
</tbody>
</table>

Table 6: Differences between functions and scripts

Now let us look at our previous example of m file.

We wanted to evaluate a quadratic equation. The general form of a quadratic equation is:

\[ ax^2 + bx + c = 0 \]

with the roots of equation given by

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

We want to create a function that will accomplish this:

- Use the Matlab editor to create a file: File → New → M-file.
- Declare the function header and then enter the command statements
- It is very important not to give function names already used in Matlab or function names that may have been used before. Always try to give a unique function name.

```matlab
function [r1,r2] = quadratic_test(a,b,c)
% function to calculate the roots of x
% help is here
r1 = (-b+(sqrt(b^2-4*a*c)))/(2*a);
r2 = (-b-(sqrt(b^2-4*a*c)))/(2*a);
end
```

- Save the function in the current directory and call the function in the command line or inside another function as follows:

```matlab
>> [r1,r2] = quadratic_test(5,6,1)
r1 =
   -0.2000
r2 =
   -1
```
If a function which is not a built-in Matlab function is called while it is not in the current directory, the following error might appear

```plaintext
??? Undefined function or variable
```

### 2.7.2. Debugging M Files

Debugging is the term used for finding errors and fixing them in computer programs. In general there are two types of errors that are corrected by debugging. Which are:

1. syntax errors (omitting a parenthesis or misspelling a function name) and
2. Run time errors (errors that are due to mistakes in coding which are hard to find and correct).

There are some simple steps that have to be followed in debugging, which are:

1. Preparation – Make sure that all the files that are called by the function are open
2. Set the break point – there are three types of break points:
   a. **standard breakpoint**, which stops at a specified line.
   b. **conditional breakpoint**, which stops at a specified line and under specified conditions.
   c. **An error breakpoint** that stops when it produces the specified type of warning, error, NaN, or infinite value
3. Run the M file from the Editor/Debugger or from the Command Window. The prompt in the Command Window changes to
   ```plaintext
   K >>
   ```
   This shows that Matlab is in debug mode. The program waits at the **breakpoint**. This means that line will be executed when you continue. The pause is indicated by the green arrow. In breakpoint, we can examine variable, step through programs, and run other calling functions.
4. Examine Values – we can see the values of the variables by pointing the mouse over them. Check if the values are correct and then step to the next line.
5. Find the error and correct the mistakes and then save the file after stopping the debugging.
3. Character strings and Cell arrays

In this section we will look at character strings and cell arrays.

3.1. Character strings

In Matlab, like any other programming language, characters are represented by numerical values. The mapping of characters to the corresponding numerical values is standardised in the American Standard Code for Information Interchange (ASCII). In Matlab text is stored in character strings. These strings are numerical arrays of the ASCII values which are displayed as the equivalent characters.

In Matlab a character string can be created by enclosing the text within single quotes "‘...’".

Let's look at an example of a character string.

```matlab
>> string = 'hello world'
string =
     hello world
```

each character has 2 bytes, if we look at the memory allocation for our character string we will be able to see this.

```matlab
>> whos
Name        Size            Bytes  Class    Attributes
string      1x11               22  char
```

Here we can see that these are 11 characters and with 2 bytes each hence, we have 22 bytes. In Matlab character strings can be manipulated just like an array. Strings can be concatenated, sliced and arithmetic, and logical operations can be performed on them.

3.1.1. Concatenating, slicing and flipping Strings

In Matlab strings are internally represented as vectors, hence all the operations that are available for vectors can be performed on strings. We looked at concatenating vectors in Section 2. Here let us look at concatenating two character strings.

```matlab
>> First_name = 'Mohamed';
>> Surname = 'Li';
>> Full_Name = [First_name,' ',Surname]
Full_Name =
    Mohamed Li
```
If we want to extract the first name or the last name only from "Full_Name" then we can do this.

```matlab
>> Full_Name(1:7)
ans =
    Mohamed
>> Full_Name(9:end)
ans =
    Li
```

Here, we have skipped the space between the first and the last names. If we want to extract some parts of the Full_Name string we can do this similar to arrays. Extract every other letter in the Full_Name string.

```matlab
>> Full_Name(1:2:end)
ans =
    MhmdL
```

Similarly we can flip the characters in the string using the “fliplr” function.

```matlab
>> fliplr(Full_Name)
ans =
    iL demahoM
```

To create a multiple row string arrays of different length in Matlab the function 'char ( )' is used. Example:

```matlab
>> List_of_Names = char('Mohamed','Li','Jhon')
List_of_Names =
    Mohamed
    Li
    Jhon
```

If we want to concatenate to string arrays of the same length we can use ‘strcat( )’ function.

```matlab
>> List_of_Girls = char('Eve','Juliet','Jenifer')
List_of_Girls =
    Eve
    Juliet
    Jenifer
>> Combined = strcat(List_of_Names,List_of_Girls)
    Combined =
    MohamedEve
    LiJuliet
    JhonJenifer
3.1.2. Arithmetic and Logical Operations

Mathematical operations can be performed on the character strings. Similarly logical operations can be performed on character strings. The operations are performed on the numerical mapping of the character strings. Let's look at some examples:

```matlab
>> 'a'+1
   ans =
         98
>> char(98)
   ans =

   b
```

If we add the character ‘a’ and 1 it gives us the numerical mapped value which corresponds to ‘b’ in ASCII. Similarly we can do logical operation on numerical mapped character strings.

```matlab
>> n = 'Eve'
   n =

   Eve

>> n > 'e'
   ans =

   0     1     0
```

Here we are finding the characters that have an ASCII values greater than ‘e’.

3.1.3. Format conversion

Some times it is useful to change data types from one type to another, e.g. from numerical data to text strings and vice versa. This is useful for flow of executable commands within Matlab’s programming environment. The particular numbers are sometimes converted to strings for annotation of charts and graphs. The commands that are used in data type conversion are listed in the table below:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ischar(a)</td>
<td>Returns true if “a” is a character string</td>
</tr>
<tr>
<td>isspace(a)</td>
<td>Returns true if “a” is a space character</td>
</tr>
<tr>
<td>isletter(a)</td>
<td>Returns true for indices that contain letters</td>
</tr>
<tr>
<td>int2str</td>
<td>Converts “a” integer to a string</td>
</tr>
<tr>
<td>num2str</td>
<td>Converts “a” number to string</td>
</tr>
<tr>
<td>sprintf</td>
<td>Write formatted data in to a string</td>
</tr>
<tr>
<td>fprintf</td>
<td>Write data to text file</td>
</tr>
<tr>
<td>mat2str</td>
<td>Converts a 2-D matrix to a string</td>
</tr>
<tr>
<td>str2num</td>
<td>Converts a string to a number</td>
</tr>
<tr>
<td>findstr</td>
<td>Find one string within another</td>
</tr>
<tr>
<td>strcmp</td>
<td>Compare strings</td>
</tr>
</tbody>
</table>
Let us look at some example:

```matlab
>> value = str2num('3.14159')
value =
    3.1416
>> whos
Name       Size            Bytes  Class     Attributes
Value      1x1                 8  double
```

In this example the string “3.14159” has been interpreted as a numerical value. Similarly we can change numerical values to strings

```matlab
>> value = 3.14159
value =
    3.1416
>> st = num2str(value)
st =
    3.1416
>> whos
Name       Size            Bytes  Class     Attributes
st         1x6                12  char
value      1x1                 8  double
```

In this example we can see “value” which is a double 8 bytes long has been converted to a string with 6 characters and each 12 bytes. If we want to examine if the value is character string or not we can use “ischar” function.

```matlab
>> ischar(value)
ans =
    0
>> ischar(st)
ans =
    1
```

The “ischar” returns a “0” or “false” because the variable “value” is a number and “1” or “true” because “st” is a string. Now let us look at finding part of the string and writing formatted data to strings.

```matlab
>> file_name = input('Enter the file name:')
Enter the file name:'expenses_2012'
file_name =
    expenses_2012
```

Notice the colons around the input this indicates that the input is a string of characters. If the colon was removed Matlab would complain, because it’s expecting an integer or a number.
>> file_name = input('Enter the file name:')
Enter the file name:hello world
??? Error: Unexpected Matlab expression.
Enter the file name:1
file_name =
    1
Now to find if the string “2012” is in the string we use
>> findstr(file_name,'2012')
    ans =
        10
We can see that the Matlab has given us a number this is the index of the start of the string we were looking for. Now we want to find out if two strings are the same.
>> strcmp(file_name,'hello_world')
    ans =
        0
The two strings are not the same hence a “0” or “false” is returned.
>> strcmp(file_name,'expenses_2012')
    ans =
        1
When the strings are exactly the same a “1” or a true is returned. Now we want to output a string to the screen, to do this we use the “sprintf” function. Notice that “%s” is the format part which tells Matlab how to format the output. In this case “%s” tells Matlab that the output is a string of characters. Other format options can be found in Matlab help.
>> sprintf('%s',file_name)
    ans =
        expenses_2012
A very closely related function to “sprintf” is the “fprintf”, which is the function to write data to file which will be explored in detail later.

### 3.2. Cell Array and Structures

#### 3.2.1. Cell Array

A cell array is a data type that can hold any legal Matlab object and they can be created “on the fly”. As an example a cell array can hold numbers, characters strings and doubles all in one. When we looked at vectors and matrices, we used square brackets “[]” to create an empty vector or matrix. Similarly a cell array is created using curly brackets “{ }”. Let’s look at example of a cell array. Lets say we have to create a data type for students, which contains their names, student ID’s, majors and fee status.
>> Student{1,1} = 'Jhon B';
>> Student{1,2} = '2012111';
>> Student{1,3} = 'Chemistry';
>> Student{1,4} = 'Full Fee';
>> Student{2,1} = 'Li X';
>> Student{2,2} = '2012112';
>> Student{2,3} = 'Engineering';
>> Student{2,4} = 'Full Fee';
>> Student{3,1} = 'Sunny B';
>> Student{3,2} = '2012113';
>> Student{3,3} = 'Biology';
>> Student{3,4} = 'Exempt';
>> Student

Student =
    'Jhon B'    '2012111'    'Chemistry'    'Full Fee'
    'Li X'      '2012112'    'Engineering'  'Full Fee'
    'Sunny B'   '2012113'    'Biology'      'Exempt'

Here we have a cell array with only character strings. The cell array consists of rows and columns. If we want to index a particular element in the cell we use the row and column numbers. Let's say we want to get the major for second student and then we want the fee status for all students.

>> Student{2,3}
ans =
    Engineering

>> Student(1:3,4)
ans =
    'Full Fee'
    'Full Fee'
    'Exempt'

Like a matrix in a cell array we can accesses individual elements as well as blocks.

### 3.2.2. Structures

In a cell array we did not have the ability to include a field name. Matlab like other programming languages allows the use of field names in structures. Let us form structure for one student first.

>> Student.Name = 'Jhon B';
>> Student.SID = '2012111';
>> Student.Major = 'Chemistry';
>> Student.Fee = 'Full fee';
Student =
Name: 'Jhon B'
SID: '2012111'
Major: 'Chemistry'
Fee: 'Full fee'

Let's see how we can enter a second student.

```matlab
>> Student(2).Name = 'Li X';
>> Student(2).SID = '2012112';
>> Student(2).Major = 'Engineering';
>> Student(2).Fee = 'Full fee';
>> Student(2)
ans =
    Name: 'Li X'
    SID: '2012112'
    Major: 'Engineering'
    Fee: 'Full fee'
```

Entering the data as we have done so far is very cumbersome; Matlab allows us to do things in an easy way. The Matlab function "struct" lets us do this much more efficiently. First we have to create the cell arrays containing the data.

```matlab
>> Name = {'Jhon B','Li X','Sunny B'};
>> SID = {'2012111','2012112','2012113'};
>> Major = {'Chemistry','Engineering','Biology'};
>> Fee = {'Full Fee','Full Fee','Exempt'};

Once we have the cell arrays we can then combine them into a structure using the struct command.

```matlab
>> Student = struct('Name',Name,'SID',SID,'Major',Major,'Fee',Fee)
Student =
1x3 struct array with fields:
    Name    SID    Major    Fee
>>
```

We have formed a structure called students. If we want to access any of the members of Students we can do as we did before. Let's say we want to access the second student data.

```matlab
>> Student(2)
ans =
    Name: 'Li X'
    SID: '2012112'
    Major: 'Engineering'
    Fee: 'Full Fee'
```
We can also access individual elements of the structure. Let’s get the name of the third student.

```matlab
>> Student(3).Name
ans =
Sunny B
```

Let’s say that we need to change the name of the third student.

```matlab
>> Student(3).Name = 'Sunny B A';
>> Student(3).Name
ans =
Sunny B A
```

Or if we want to delete the name.

```matlab
>> Student(3).Name = [];
>> Student(3).Name
ans =
[]
```

But this will not delete the rest of the data in the structure we can still get the other information.

```matlab
>> Student(3).SID
ans =
2012113
```

There are lots of commands that can be used with structures which can be found in Matlab Matlab Help. It would be a good idea to go through them to see which commands are handy.
4. File Input / Output and Plotting

So far we have discussed how to use Matlab and how to create programs. The data that is analyzed by programs and outputs that are generated from those programs need to be read and saved. It is not always practical to enter data manually, especially if there is a large amount of data and it is very important that the outputs that are generated from such large sets of data be saved appropriately. There are several ways to do Input/output operations in Matlab. Let’s look at each one at a time.

4.1. Work Space I/O

Matlab has its own format to store files which is “.mat” extension. Matlab allows you to store your complete work space to a file with “save” command and reload it using the “load” command. You can choose to save with or without variables specified, if you choose not to specify variables then all variables in the Matlab work space will be save.

Let’s look at some simple examples:

>> save mydata Student

This command saves the struct we build before in a .mat file called mydata.m

>> load mydata

This command will load the file mydata. Here we have specified a file name, but if we do not specify a file name then Matlab will save the file under the default file name “Matlab.mat”.

4.2. Text Files

Sometimes it is necessary to save the data that is generated by Matlab into a text file. This is usually the case when some other program is used to analyse the data. Matlab allows you to save to text files with “.txt” extension. A detail look at opening text files and manipulating it is given in the recommended text book “Engineering Computation with MATLAB”. Here we will look at an example and see if we can write and read from a text file.

The command that we will use is the “fopen” command which lets you open a file for reading or writing. If you don’t have a file this command will create a file. Let’s say we don’t have any file and we want to write a sentence to a file.

>> fid = fopen('mytest.txt','w'); % create or opens a file called “mytest.txt” for writing

>> fprintf(fid,'Hello World');% write the text to file

>> fclose(fid);% close the file

Here we have created a text file called mytest.txt and written sentence to it and then closed the file. It is very important that you always remember to close any file after you have done with.
Now let's read the file:

```matlab
>> fid = fopen('mytest.txt','r'); % open the file to read
>> text = fscanf(fid,'%c',inf) % read from file
    text =
        Hello World
    >> fclose(fid); % close the file
```

Here we have opened the file from before, read the text and closed the file. The “fscanf” command lets you read from the file and ‘%c’ is the format option which tells “fscanf” that we are reading a sequence of characters, and “inf” tells “fscanf” to read till the end of the file.

The most common commands that are used in Matlab for file operations are:

- fopen – open file for read/write
- fclose – close file after done
- fread,fscanf – read from file
- fwrite,fprintf – write to a file.

The options that are available with these commands and more examples can be found in Matlab help files.

Now let's look at another example. This time we want to create two columns one with value of x and the other with the sin (x) and we want to save the file and then read the saved file.

```matlab
>> x = 0:0.2:pi;
>> y = [x;sin(x)];
>> fid = fopen('sin.txt','w');
>> fprintf(fid,'%6.2f %12.8f
',y);
>> fclose(fid);
```

The output file looks like this.

```
0.00   0.00000000
0.20   0.19866933
0.40   0.38941834
0.60   0.56464247
0.80   0.71735609
1.00   0.84147098
1.20   0.93203909
1.40   0.98544973
1.60   0.99957360
1.80   0.97384763
2.00   0.90929743
2.20   0.80849640
2.40   0.67546318
2.60   0.51550137
2.80   0.33498815
3.00   0.14112001
```

Here we have opened the file from before, read the text and closed the file. The “fscanf” command lets you read from the file and ‘%c’ is the format option which tells “fscanf” that we are reading a sequence of characters, and “inf” tells “fscanf” to read till the end of the file.

The most common commands that are used in Matlab for file operations are:

- fopen – open file for read/write
- fclose – close file after done
- fread,fscanf – read from file
- fwrite,fprintf – write to a file.

The options that are available with theses command and more examples can be found in Matlab help files.
Now we want to read the file in the correct format.

```matlab
>> fid = fopen('sin.txt','r');
>> Mat = fscanf(fid, '%g %g', [2 inf]);
```
In this code we are using the fscanf command with ‘%g’ which tells Matlab that the data we are reading is floating point numbers. And there are 2 columns to read. The syntax of “fcanf” can be found in Matlab help. There are other functions that can be used to read data from text files which include, “fgetl”, “fgets”, “sscanf” and “fread”. For brevity we will let you explore these functions using the Matlab help command.

```matlab
>> fclose(fid);
>> Mat = Mat'
```
```
Mat =
   0    0
 0.2000  0.1987
 0.4000  0.3894
 0.6000  0.5646
 0.8000  0.7174
 1.0000  0.8415
 1.2000  0.9320
 1.4000  0.9854
 1.6000  0.9996
 1.8000  0.9738
 2.0000  0.9093
 2.2000  0.8085
 2.4000  0.6755
 2.6000  0.5155
 2.8000  0.3350
 3.0000  0.1411
```

Another simpler way to output formatted text is to use Microsoft Excel spared sheets.

### 4.3. Read and Writing data to spread sheets in Matlab

Matlab allows data to be written and retrieved from Microsoft Excel Files. Let’s see if we can write some data into an excel file. The two commands that are most handy are the “xlswrite” and “xlsread” command. The syntax of these two commands are given below.

```
[SUCCESS,MESSAGE]=XLWRITE(FILE,ARRAY,SHEET,RANGE)
[NUMERIC,TXT,RAW]=XLSREAD(FILE)
```
In the “xlswrite” function we have the array that we want to save, the sheet to which we want to save and the range to which we want to save. In the “xlsread” function we have three forms of output, which are the numerical values only, text only and the raw data. Let us continue with a new example now.

```matlab
>> example{1,1} = 'add';
>> example{2,1} = 'she';
>> example{3,1} = 'bir';
>> example{4,1} = 'hop';
>> example{1,2} = '3.5';
>> example{2,2} = '4.0';
>> example{3,2} = '2.0';
>> example{4,2} = '3.5';

>>xlswrite('test.xlsx',example,1)
```

Here we have written the entire cell array in to excel spread sheet at once.

Now let’s read the spread sheet in to Matlab.

```matlab
[num,text,raw] = xlsread('test.xlsx')
```

```
num =
    3.5000
    4.0000
    2.0000
    3.5000
text =
    'add'
    'she'
    'bir'
    'hop'
raw =
    'add'    [3.5000]
    'she'    [ 4]
    'bir'    [ 2]
    'hop'    [3.5000]
```
4.4. Reading and writing audio and image files

Matlab allows you read and write audio and image files. The commands that are related to reading and writing image and audio files are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>Load a matrix as MAT file</td>
</tr>
<tr>
<td>save</td>
<td>Save a Matrix as MAT file</td>
</tr>
<tr>
<td>imread</td>
<td>Load a image file</td>
</tr>
<tr>
<td>imwrite</td>
<td>Save a image file</td>
</tr>
<tr>
<td>image / imagesc</td>
<td>Display an image</td>
</tr>
<tr>
<td>ind2rgb</td>
<td>Convert indexed image to rgb</td>
</tr>
</tbody>
</table>

Table 8: Commands for Reading and writing Image files

Example to load a image:

```
>> RGB = imread('ngc6543a.jpg'); % load image
>>image(RGB); % display image
>>imwrite(RGB,'ngc.jpg','jpeg'); % writes the file
```

Let's look at audio files now.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wavread</td>
<td>Reads audio file</td>
</tr>
<tr>
<td>wavwrite</td>
<td>Writes a audio file to file</td>
</tr>
<tr>
<td>sound</td>
<td>Plays an audio file</td>
</tr>
</tbody>
</table>

Table 9: Command for Reading and writing audio files

Examples:

```
>>[y,fs,bts] = wavread('filename.wav'); % read a audio file
Here the y contains the values of wave file, “fs” is the sampling frequency and “bts” is the number of bits per sample.

>>wavwrite(y,fs,bts,' filename.wav');
>>% writes the wave file with the sampling rate “fs” and number of bits per sample “bts” with file name “filename.wav”.
```
4.5. Plotting

Matlab contains a comprehensive set of graphics tools. It is very easy to plot a given set of data using very simple to use graphics commands.

4.5.1. Creating a simple 2D plot

The command to plot a 2D plot of two arrays is “plot”. The syntax for plot command is:

```
PLOT(X,Y,'format')
```

Here X and Y are vectors of equal length and “format” specifies the color, and style of the plot. Let’s look at example.

We want to plot a sin curve.

```matlab
>> x = -pi:0.1:pi;
>> y = sin(x);
>> plot(x,y)
```

The output look like this

![Figure 1: Plot of Sin(x)](image)

If we wanted to change the color and the line style we can do that. Let’s say we want to plot this graph in red with stars.

```matlab
>> plot(x,y,'r*')
```
The plot now looks like this:

![Plot of Sin(Θ)](image)

Figure 2: Example of Plot Color and Markers

The plot does not have any labels, let’s add title to the graph and the x and y axis labels.

```matlab
>> title('Plot of Sin(\Theta)')
>> xlabel('-\pi \leq \Theta \leq \pi');
>> ylabel('Sin(\Theta)');
```

![Plot of Sin(Θ)](image)

Figure 3: Example of Title and Axis Labels

Let’s say we want to keep our existing plot and on top of it plot 0.5sin (Θ). In this case we have to tell Matlab to hold the current plot. If you don’t, Matlab will replace this plot. We use the “hold” command to keep the current plot. If we want we could plot the new graph in new window without replacing the existing plot. To do this we use the “figure” command. Every time you type “figure” a new figure window will be opened. Let see how we can do this.
>> hold
Current plot held
>> y = 0.5*sin(x);
>> plot(x,y,'g')
>> legend('sin(\Theta)','0.5*sin(\Theta)')

Figure 4: Example of Hold Function

It is important to un-hold the current figure if you don’t want to plot on it. To do this we type “hold” command again.

>> hold
Current plot released

Let’s say we want to plot the two graphs from the previous example but side by side so that we can compare. And we would like to add cos(\theta) and 0.5cos(\theta) as well. Matlab allows us to split the figure window into many smaller sub plots. Let’s see how we can do this.

>> figure
>> subplot (2,2,1)

With these two commands we have created a fresh figure window and then we have split the window into two columns and two rows. And we are going to plot our first graph in the top left corner.
Let’s plot the first graph and put in all the labels.

```matlab
>> plot(x,y)
>> title('Plot of Sin(\Theta)');
>> xlabel('-\pi \leq \Theta \leq \pi');
>> ylabel('Sin(\Theta)');
```

![Plot of Sin(\Theta)](image.png)
Now we want to put $0.5\sin(\theta)$ below this graph.

```matlab
>> subplot (2,2,3)
>> ys = 0.5*sin(x);
>> plot(x,ys)
>> title('Plot of 0.5\sin(\Theta)');
>> xlabel('-\pi \leq \Theta \leq \pi');
>> ylabel('0.5\sin(\Theta)');
```

![Plot of Sin(\Theta)](image)

Now that we have mastered this we can now put the remaining graphs in .

```matlab
>> subplot (2,2,2)
>> yc = cos(x);
>> plot(x,yc)
>> title('Plot of \cos(\Theta)');
>> xlabel('-\pi \leq \Theta \leq \pi');
>> ylabel('\cos(\Theta)');
```

![Plot of 0.5Sin(\Theta)](image)

```matlab
>> subplot (2,2,4)
>> ycc = 0.5*cos(x);
>> plot(x,ycc)
>> title('Plot of 0.5\cos(\Theta)');
>> xlabel('-\pi \leq \Theta \leq \pi');
>> ylabel('0.5\cos(\Theta)');
```

![Plot of 0.5Cos(\Theta)](image)
4.5.2. 3D plots

The simplest way to draw a 3D plot is to extent the 2D plot to by adding a set of z axis value. In Matlab we use the command “plot3” for 3D line plots. Let’s look at a simple example of 3D line plots in Matlab.

```matlab
>> t = -pi:0.01:pi;  
>> x = cos(t);  
>> y = sin(t);  
>> z = sin(5*t);  
>> plot3(x,y,z);  
>> grid;  
>> title('x = cos(t), y = sin(t), z=sin(5t)');  
>> xlabel('x');  
>> ylabel('y');  
>> zlabel('z');
```

Figure 8: Example of Four Plots

We have put four graphs in one figure window. It is important to remember that you will need to put the title and the axis on each graph individually. Now that we have mastered 2D plots let’s look at 3D plots.
Now let’s look at more complicated 3D plots which are known as surface plots. The previous example of a 3D line plot was a linear one where we only have one single line. Now we want to add surfaces, color and texture to our plots. The three functions that we can use to do this are the “meshgrid”, “mesh” and “surface” functions. These functions are useful when we have a single variable which is a function of two independent variables. Let’s see how we use these functions to create a 3D plot.

```matlab
>> x = -2:1:2; % generate axis vectors
>> y = -2:1:2;
>> [X,Y] = meshgrid(x,y) % make a mesh grid.
X =
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
% X is a matrix with every element value set to its x coordinates
Y =
    -2    -2    -2    -2
    -1    -1    -1    -1
     0     0     0     0
     1     1     1     1
     2     2     2     2
% Y is a matrix with every element value set to its x coordinates
>> Z=X.*exp(-(X.^2+Y.^2)); % A 2D Gaussian .* X.
>> mesh(X,Y,Z); % display the plot as a mesh
```

Figure 9: 3D Line Plot Example

```matlab
x = cos(t), y = sin(t), z = sin(5t)
```
Figure 10: Mesh Plot Example

>> surf(X,Y,Z); % display the plot as surface

Figure 11: Surface Plot Example

>> contour(X,Y,Z); % Display the plot as contour
Figure 12: Contour Plot Example