Tutorial - I

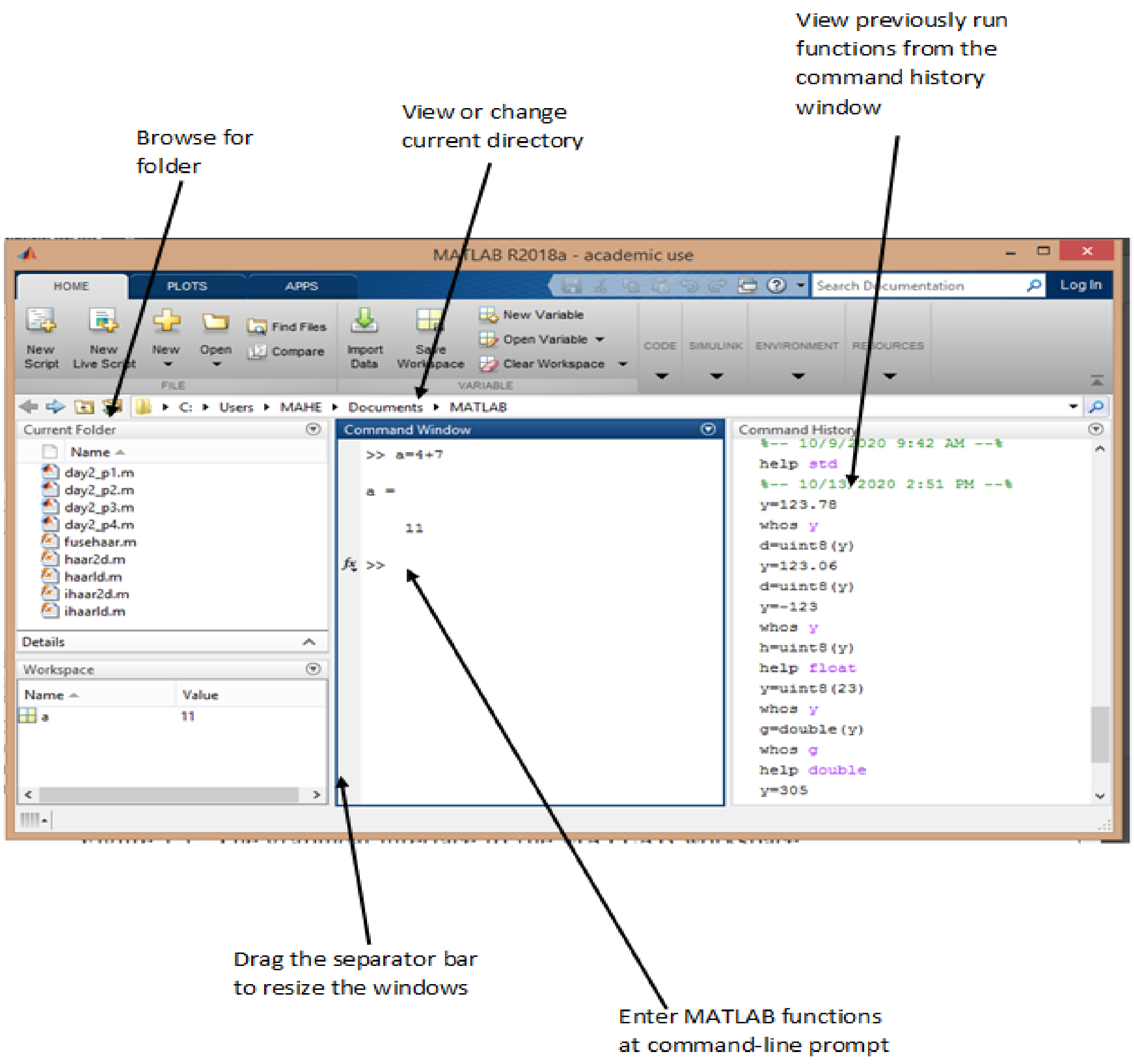
A brief introduction to MATLAB

General Instruction:
This is a hands-on tutorial. MATLAB commands for you to type are printed in **bold letters (blue in color)**. Bold letters are also used to make MATLAB expressions that are in lower case more visible when found in a sentence.

This tutorial assumes you can open the MATLAB GUI. We are using **R2017/R2018** version of MATLAB.

Part-1: The basics:

1) Start MATLAB by double clicking on the MATLAB icon on Desktop. The MATLAB window should come up on your screen. It looks like as shown in the Figure 1.1.



```
>> a=4+7

a =

    11

>>
```

```
%-- 10/9/2020 9:42 AM --%
help std
%-- 10/13/2020 2:51 PM --%
y=123.78
whos y
d=uint8(y)
y=123.06
d=uint8(y)
y=-123
whos y
h=uint8(y)
help float
y=uint8(23)
whos y
g=double(y)
whos g
help double
y=305
```

Figure 1.1 The graphical interface to the MATLAB workspace

This is the window in which you interact with MATLAB. The main window on the middle is called the Command Window. You can see the command prompt in this window, which looks like >>. If this prompt is visible, MATLAB is ready for you to enter a command. In the figure, you can see that we typed in the command a=4+7. In the bottom left corner, you can view the Workspace window. The Workspace window will show you all variables that you are using in your current MATLAB session. In this example, the workspace contains the variable 'a'. When you first start up MATLAB, the workspace is empty.

In the right side of the window you can see the Command History window, which simply gives a chronological list of all MATLAB commands that you used, the Current Directory shows you the location of the directory you are currently working in and the Current Folder shows you the contents.

You can change the layout of the MATLAB window:

2) To change the layout of the MATLAB window, click on *Layout*. Different layout styles can be chosen.

Note that MATLAB is case-sensitive, so

3) Make sure that the 'Caps Lock' is switched off!

During the MATLAB sessions you will create files to store programs or workspaces.

4) Create an appropriate folder to store this lab session's files.
5) Click on Browse for folder. Select the folder you just created so that MATLAB will automatically save files in this folder.
6) Now, let's try a simple command. Next to the prompt in the Command Window type **a=4+7** and then press 'enter' to activate this command. On the screen you should see

```
a=

         11
```

Notice how MATLAB carries out this command immediately, and gives you the prompt >> for your next command. Here, a variable called 'a' is created by MATLAB and assigned the value of 11. MATLAB stores the variable 'a' in its *workspace* until you exit MATLAB or tell MATLAB to delete the variable.

7) Go to the Workspace window and check that the variable 'a' is in your workspace.

You will see in this window that 'a' is stored as a double and that it has size 1x1.

8) In the Workspace window, double click on the name 'a'. A small window will pop up displaying a’s value. This is a handy feature. We can change the value of the variable.
9) Try **a = 16**, followed by 'enter' to change the value of a to 16. Then type **b=sqrt(a)** to find the square root of a. Press 'enter' to enter the command (in future hit ‘enter’ will not be mentioned when there is a MATLAB command). Now you should see

```
b=

         4
```

10) Go back to the Workspace window and check that 'b' is indeed in your workspace. Also check by double- clicking on 'a' that a’s value has changed.

Part-2:The MATLAB on-line help facility

MATLAB has handy on-line help facilities. There are several ways to get help. You can go to Help on the menu (Under Home toolbar Help menu is present) and select any of the available help facilities listed there. There are simpler help commands as well that work in all versions of MATLAB.

1) Type the command help in the Command Window to find a long list of all different categories for which there are MATLAB commands. Each of the listed categories contains more detailed information about available MATLAB functions. For example, type **help ops** to find a list of MATLAB operators. You can see from this list, for example, that more information about addition can be found in **help arith**.
2) If you want to know more about a MATLAB command type **help** followed by MATLAB command. For example type **help plot**

Alternatively, you can launch the help window:

3) Type **helpwin** to launch the help window.

In this window, you can get help on operators, for example, by double clicking on 'ops'.

Another very useful MATLAB command is the **lookfor** command. You can use this to search for commands related to a keyword.

4) Find out which MATLAB commands are available for plotting using the command lookfor plot in the command window. It may take a bit of time to get all commands on the screen. You can stop the search at any time by typing Ctrl-c (the control and the c key simultaneously).

Important
In the MATLAB **help** descriptions, the MATLAB commands and functions are often given in capital letters. But: always type commands in lower case. MATLAB is case sensitive and will generally not recognize commands typed in capital letters! Note that because of this case sensitivity the variables 'A' and 'a', for example, are different.

Part-3: Data representation in MATLAB

MATLAB stands for 'MATrix LABoratory'. This title is appropriate because the structure for the storage of all data in MATLAB is a matrix. The MATLAB matrix-variables may have any number of rows and columns. Scalars like the variables 'a' and 'b' that you worked with above are also stored as matrix variables with 1 row and 1 column.

So beware, a matrix-variable can be any variable in MATLAB, that is, it could be a scalar, a vector or a matrix of any size.

Entering variables:

An mxn ('*m* by *n*') MATLAB matrix-variable has *m* rows and *n* columns. For example, the variable $A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$ is a 2 x 3 matrix.

1) Create the matrix A by typing **A = [1 3 5;2 4 6]** followed by the enter. Make sure that you separate the elements 1,3 and 5, and 2, 4 and 6 with a space. Use square brackets instead of parentheses. Instead of a space, you can also use a comma to separate the elements. The semi-colon (;) in this context is used to separate the elements of one row from another, but you can also use a line break to separate rows as shown below.

2) Remove the matrix A by typing clear A. Then recreate A using

```
>> A = [1,3,5        (now hit enter for the line break)
        2,4,6]
```

Error messages:

If your command is invalid MATLAB gives you explanatory error messages (luckily!). Read them carefully. Normally MATLAB points to the exact position of where things went wrong.

3) Try to change A to $\begin{bmatrix} 17 & 15 \\ 16 & 18 \end{bmatrix}$ using the incorrect command **A=[17 15 16;18]**

The command has failed. MATLAB tells you that dimensions of arrays being concatenated are not consistent. You put the semi-colon in the wrong place.

Row and column vectors:

A row vector has one row and any number of columns. Similarly, a column vector has one column and any number of rows.

4) Type **v=[1, 2, 3]** . Now use the size function to check that v has 1 row and 3 columns by typing **n= size(v)**

The first number returned by the **size** function gives the number of rows in the variable and the second number gives the number of columns.

5) Enter the column vector $B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ by typing **B=[1;2;3]** and check that is has 3 rows and 1 column.

Changing matrices:

You can change a matrix simply by defining it to be something else. We already unsuccessfully tried this when we tried to change A. Let’s get it right this time.

6) Type **A=[11 22;33 44]** to make A become the 2x2 matrix given earlier.

The 2x3 matrix that A used to be has now been lost. The variable A still exists of course in MATLABs workspace but now has a different size and different values for its elements.

Part-4: Scrolling

Suppose you want to repeat or edit an earlier command. Instead of typing it again, there is a way to do it: MATLAB lets you search through your previous commands using the up-arrow and down-arrow keys. This is a very convenient facility, which can save a considerable amount of time.

1) Suppose you regret the last command, and that you want A still to be the 2x3 matrix you originally defined. Press the up-arrow key until the earlier command.

   A=[1 2 3;4 5 6] appears in the Command Window (the down-arrow key can be used if you go back too far). Now press enter. The matrix A is back to what it was originally.

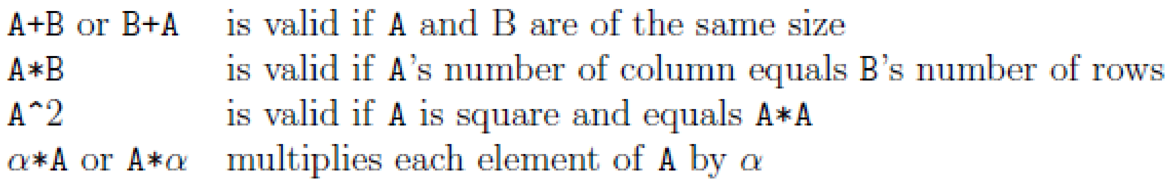
You can speed up the scroll if you remember the first letters of the command you are interested in. For example, if you quickly want to jump back to the command **v = [1, 2, 3]** just type **v =** and then press the up-arrow-key. MATLAB will display the most recent command starting with **v =**.

You can also use the Command History window to jump to a previous command:

2) Go to the Command History window. Jump back to the command **v=[1,2,3]** by double-clicking on this line in the Command History window. The command will be carried out immediately.

Part-5: Basic arithmetic operations in MATLAB

1) Solve and verify that the product of the matrices $A = \begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix} \mathbf{4} & \mathbf{2} \\ \mathbf{3} & \mathbf{5} \end{bmatrix}$ is equal to $AB = \begin{bmatrix} 10 & 12 \\ 3 & 5 \end{bmatrix}$. Also, find $BA = \begin{bmatrix} 4 & 10 \\ 3 & 11 \end{bmatrix}$.
2) Create the matrices A and B in MATLAB.
3) Type **A*B** to perform the first multiplication. Because you did not give the result of this expression a name MATLAB calls it 'ans' (which is short for answer). Note that multiplication of matrices in MATLAB as in A*B is the *matrix multiplication (product)* operation.
4) Type **B*A** to perform the second matrix multiplication. The matrix 'ans' has now changed to be the result of this expression. In MATLAB 'ans' is the result of the last unnamed MATLAB command.
5) Type **A+B** to add the two matrices. Type **A-B** to subtract the matrices. Type **3*A** to multiply A by 3.



`A+B` or `B+A`	is valid if `A` and `B` are of the same size
`A*B`	is valid if `A`'s number of column equals `B`'s number of rows
`A^2`	is valid if `A` is square and equals `A*A`
α`*A` or `A*`α	multiplies each element of `A` by α

Array arithmetic operations:

Array arithmetic operations or array operations for short, are done element-by-element.

`.*`	Element-by-element multiplication
`./`	Element-by-element division
`.^`	Element-by-element exponentiation

Example: **k= A.*B**

Part-6: Keeping in touch with your variables

You can see/check all the variables created in the Workspace window. Alternatively you can use commands in the Command window to do the same:

1) Type **who** to see all your matrices.
2) Type **whos** to see a more complete description, including sizes, of all stored data.

Deleting matrices:

3) Remove the matrix B and the vector v by typing **clear B v**. Alternatively, you can remove B and v by selecting them in the Workspace window and then clicking the 'Delete' button. Remember that you don't have to clear matrices to use the same names again. You can just set up a new matrix with the same name.
4) Change the matrix A by typing **A=2**.
5) Check that A is now a *scalar* - the previous A matrix is lost. Again, we changed the size and the value of the elements of the matrix-variable A. The size is now 1x1.
6) *Remove all* variables by just typing **clear** without any names following.
7) Check that there are is nothing left in your workspace. Careful with this command!

Part-7: Generating variables

MATLAB provides functions to create several basic matrices automatically without having to type or read in each of the elements. The most important functions are

zeros	**zeros(m,n)** creates an mxn matrix whose elements are equal to zero.
ones	**ones(m,n)** creates an mxn matrix whose elements are equal to one.
eye	**eye(m,n)** creates an mxn identity matrix
rand	**rand(m,n)** creates an mxn matrix whose elements are all random number between 0 and 1

1) Create a 5x5 matrix with random numbers. Now add to it a diagonal matrix with the diagonal elements equal to 2. You can create such a diagonal matrix using the command **2*eye(5,5)** .
2) Type **h=10:2:20**. You should see that you have created a row vector h with elements starting at 10 and increasing in steps of 2 up to 20. We could also have created this vector with **h=[10 12 14 16 18 20]**. Note that we do NOT need brackets in **h=10:2:20**

3) If the step size is negative the sequence decreases. So **i=9:-1:5** sets i to be [9 8 7 6 5].
4) Type **r=50:55**. You should have the vector [50 51 52 53 54 55].

Suppression of output:

Up till now you have always seen the results of each command on the screen. This is often not required or even desirable; the output might clutter up the screen, or be long and take a long time to print. MATLAB suppresses the output of a command if you finish the command with a semi-colon **;.**

5) Create an vector by typing **rr=1:50;**. Observe that output of the command is supressed.

Building larger and sub matrices from original ones:

6) Delete all variables using **clear**.
7) Create the matrices $A = \begin{bmatrix} 5 & 3 \\ 8 & 9 \end{bmatrix}$ and $B = \begin{bmatrix} -3 & 0 & 4 \\ 2 & 5 & 0 \end{bmatrix}$
8) Now type **C=[A B]** or **C=[A,B]** and you should see that MATLAB put A and B side-by-side to get another matrix C.
9) Type **D=[A B;B A]** and you should see that MATLAB placed the matrix [B A] below the matrix [A B]
10) Now type **H=[A;B]**

This gives an error message, which tells you that this operation cannot be done. The number of columns in A and B are not equal.

11) To extract a submatrix P consisting of rows 2 and 3 and columns 1 and 2 of the matrix D, type **P = D([2 3],[1 2])**

Changing values of individual elements:

Individual elements in a variable can be identified using index numbers. For example the element 8 in the matrix A above can be referred to as A(2,1) because it is the first element of the second row of A. Note that the row and column indices are separated by a comma.

12) Change this element to have the value of 1 by typing **A(2,1)=1**
13) Type **A2 =A(1:2,1)** to extract the first column of A and store it in A2.

We could also achieved this by typing **A2 = A(:,1)** . The colon operator identifies all of the first column (taking the first element in ALL of the rows). Similarly **A(2,:)** identifies the whole of the second row of A. Note that MATLAB will give error if you use negative or zero indices for matrices.

However if you use an index which is above the maximum dimension the matrix is enlarged and all the intervening elements are set to zero.

14) Type **A(4,4)=2**.

See how the extra elements of the matrix have been created, but only A(4,4) has been set to 2, the other extra elements are set to zero by default.

15) Swap the first and second columns of B by typing **B = [B(:,2) , B(:,1), B(:,3)]** .

Transposing matrices and vectors:

The operator ' (single quote) finds the transpose of a variable. For example, if A is the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ then **B = A'** gives the matrix $B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$

16) Set up a column vector of the first 10 whole numbers by writing **(1:10)'**

Part-8: Saving and retrieving your work

All variables that you created in this MATLAB session are stored in MATLAB's workspace. Upon exiting MATLAB contents of workspace is deleted. So you must save your workspace if you want to use it in later MATLAB sessions.

1) Select **Save Workspace** under **Home** menu. A save window will come up. Check that the correct folder is given. Type in the name you wish to use for the workspace file, for example 'labsession1' and save the file. MATLAB automatically selects the 'mat' extension for the workspace files.

2) Type **clear** to delete your workspace. Type **who** to check that your workspace is empty (or check your Workspace window). Retrieve the saved workspace using **Import Data** under **Home** menu. Type **who** again to check that all your variables are back in the workspace (or check the Workspace window).

   This method of saving saves all the variables in your workspace. You can select any number of variables if you save from within the Command Window instead.

3) Create two random variables by typing **x=rand**; and **y=rand**; To save these two variables in a .mat file called **'savexy.mat'** type **save savexy x y** .

   The contents of your current directory is shown in the Current Folder. But you can also find out its contents using a command:

4) The command **dir** gives you the list of all files that are in your current directory. The command **what** gives you the files in the current folder relevant to MATLAB only. Try them both.

Practice:

1) Create a 4x3 matrix. Interchange the 2nd and 3rd rows using one line of code.
2) Create a 8x8 matrix with random numbers between 0 and 8. Now, make all elements in the first row and first column equal to 1.
3) We would like to create the row vector [10 8 6 4 2 0 ….. 0 0 10] with a total number of elements equal to 100 (that means there are 94 zeros in the vector). Think of two ways to create this variable without typing in all the numbers.

Part-9: Script files

Let us see how to store commands in a file. Such MATLAB are known as script files. They are stored with the .m extension. In MATLAB files with the .m extension are also called M-files.

1) From the Home menu choose *New Script*. File then New then M-file.
2) Type the following lines in the new file window. Note that the file is still called '*Untitled*'.

```
% Add two matrices and display the result
clc;
A = [1, 1; 2, 2];
B = [2, 2; 3, 3];
D = A + B;
disp('The result is ');
disp(D);
```

The first line is a comment written to explain what your program does. MATLAB ignores everything on a line after a % character.

Now save this file. Saving option is there under '*Editor*' Menu.

3) *Save* the file by clicking on '***save***' or '***save as***'. Save the file in the folder you already created. See to that '*save as type*' has *.m* extension.

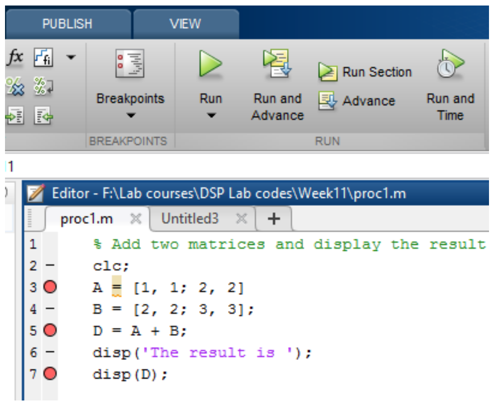
Rules for naming variable/script files:
Start with a letter, followed by letters or numbers or underscore, maximum 64 characters (excluding the .m extension), and must not be the same as any MATLAB reserved word

4) Run the file by clicking on '***Run***' under Editor Menu.

Running a program step-by-step:

Often it is convenient to run a program step by step to carefully check that all the lines of code that you wrote are doing what you expect them to do. This is a good way to debug your code to fix the errors in your code.

5) Click on the first command line of the .m file already. Go to Breakpoints option under Editor Menu. Click on ‘*Set/Clear’*. Click the mouse to on the code to mention the breakpoints in .m file as shown in the figure. Remove the semi-colon from the end of command lines where breakpoints are present. Save the file.



```
% Add two matrices and display the result
clc;
A = [1, 1; 2, 2]
B = [2, 2; 3, 3];
D = A + B;
disp('The result is ');
disp(D);
```

6) Next click on ‘*Run’*. See the partial result on Command window. If multiple breakpoints are present, you need to click ‘*Run’* multiple times to see all the partial outputs. To come out of debugging mode click on ‘*Quit Debugging’*.
7) To remove the breakpoints click on ‘*clear all’* under ‘*Breakpoints’* Menu.

Under Breakpoints on the menu there are more options. Explore the MATLAB debugger and use it in this lab course. It is a very useful tool.

Also, know to comment out parts of your cod or uncomment using Comment menu under Editor Toolbar.

Part-10: Logical expressions

Relational operators:

eq	- Equal	==
ne	- Not equal	~=
lt	- Less than	<
gt	- Greater than	>
le	- Less than or equal	<=
ge	- Greater than or equal	>=

In MATLAB a logical expression has two possible values that are ‘true’ or ‘false’ but numeric, i.e. 1 if the expression is true and 0 if it is false.

1) Go back to the Command Window. Set up the scalars **q=10**; **w=10**; and **e=20**.
2) Type **w< e** to see that the result is given the value of 1 because w is indeed less than e. Now type **q==e**. The == operator checks if two variables have the same value. Therefore the answer is 0 in this case.

3) Relational operations are performed after arithmetic operations. For example **q == w-e** results in 0. Parentheses can be used to override the natural order of precedence. So **(q==w) - e** results in -20.
4) Create the vectors x and y by typing **x=[-1, 3, 9]** and **y=[-5, 5, 9]** . Now type **z= (x < y)**. The i-th element of z is 1 if x(i) < y(i), otherwise it is 0. So, the answer is the vector with elements 0, 1, and 0.

Logical operators: Please go through logical operators.

and	- Element-wise logical AND	&
or	- Element-wise logical OR	|
not	- Logical NOT	~

5) Give the answer for **(e>0) & (q<0)**
6) Type **(e>0) | (q<0)**. What is the answer?

Part-11: Solving linear equations

In linear algebra we learn that the solution to Ax = b can be written as x = A^{-1}b, where A^{-1} is the inverse of A.
For example, consider the following system of linear equations:

$$x + 2y + 3z = 1$$
$$4x + 5y + 6z = 1$$
$$7x + 8z = 1$$

1) To get the solution type g=inv(a)*b where $a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 8 \end{bmatrix}$ and $b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ or **g=a\b**

Matrix Functions:

Some of the Matrix based MATLAB commands are given below.

`det`	Determinant
`diag`	Diagonal matrices and diagonals of a matrix
`eig`	Eigenvalues and eigenvectors
`inv`	Matrix inverse
`norm`	Matrix and vector norms
`rank`	Number of linearly independent rows or columns

Part-12: The if statements

MATLAB supports the variants of ‘if’ construct.

1) **if ...... end**

Example:

```
mark=30;
if mark<40
disp(‘F grade');
disp(mark);
end;
```

2) **if … else … end**

Example:

```
mark=80;
if mark>40
disp(‘P grade ');
else
disp(‘F grade’);
end;
```

3) **if … elseif … else … end**

Example:

```
mark=80;
if mark>70
disp(‘Excellent ');
elseif mark>40
disp(‘P grade ');
else
disp(‘F grade’);
end;
```

Part-13: The for loop

Example-1

```
angle = pi*rand(3,1);
for k=1:3
cosangle = cos(angle(k));
disp(cosangle);
end;
```

Example-2

```
x = rand(4,1);
maxval = 0;
for k=1:4
if (x(k) > maxval)
maxval = x(k);
```

```
end;
end;
disp(maxval);
```

Part-14: Nested for-loops

Example

```
A = rand(2,2);
for k=1:2
for m=1:2
exponential = exp(A(k,m));
display(exponential);
end;
end;
```

Part-15: While loops

Example

```
x = 15;
while x > 0
x = x-4;
disp(x);
end;
disp('Done');
```

Part-16: Writing your own MATLAB functions

The first MATLAB function you will write is a simple function to convert temperatures given in degrees Fahrenheit into degrees Celsius, according to the formula:

$$T_C = (T_F - 32)\frac{5}{9}$$

1) Open a new script file. Type the following code:

```
function tc = fahtocel(tf)
% converts function input 'tf' of temperatures in Fahrenheit to
% function output 'tc'of temperatures in Celsius
temp = tf-32;
tc = temp*5/9;
return;
```

The word function on the first line tells MATLAB that you are about to define a function. The next words tc = fahtocel(tf) tell MATLAB that:
- the function is called 'fahtocel'
- the function returns a function output variable called 'tc'
- the function is given a function input variable called 'tf'.

The return statement indicates the end of the function and asks MATLAB to return to the place from which the function was called. You can call functions from the command window of course, or from script files, or even from other functions.

Function M-files must always start with the function statement and the name of the function M-file must always be the same as the name of the function. So here, we must save this M-file as 'fahtocel.m'.

2) Save the M-file as 'fahtocel.m'. MATLAB will suggest the name fahtocel.m to you automatically and all you have to do is press the save button.

<u>Calling the function from a script file:</u>

3) Open a new script file. Type the following code:

```
tempf=1000;
tempc=fahtocel(tempf)
```

4) Save and run the file.

<u>Functions with more than one output variable:</u>

5) Type the below function file

```
function [vol, surf] = cuboid(len,br,dep)
% Function to return the volume and surface area of a cuboid given
% the length len, breadth br and depth dep
vol = len*br*dep;
surf = 2*(len*br + len*dep + br*dep);
return;
```

6) Write the script file:

```
% script file for cuboid
length =[ 5, 10];
breadth = [4, 10];
depth = [3, 8];
% call the function
[volume, surface] = cuboid(length,breadth,depth)
```

7) Save and run the file.

Part-17: Plotting using MATLAB

Basic Two-Dimensional Plot:

```
t=0:0.01:2;
x=2*sin(2*pi*50*t);
plot(t,x)
xlabel('Time')
ylabel('signal amplitude')
title('first plot')
```

Multiple plots in single window:

```
x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
plot(x,y1,'* -',x,y2,'r s -')
xlabel('x')
ylabel('sin(x), cos(x)')
title('Trig Functions')
legend('sin','cos')
```

Subplots:

```
x=0:pi/16:2*pi;
y1=sin(x);
y2=cos(x);
subplot(2,1,1)
plot(x,y1,'* -')
xlabel('x')
ylabel('sin(x)')
subplot(2,1,2)
plot(x,y2,'r s -')
xlabel('x')
ylabel('cos(x)')
```