MANIPAL UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

MATLAB for Engineers ME1201

Introduction to MATLAB

Module 1

Introduction to MATLAB environment and commands

Lecture 1 and 2

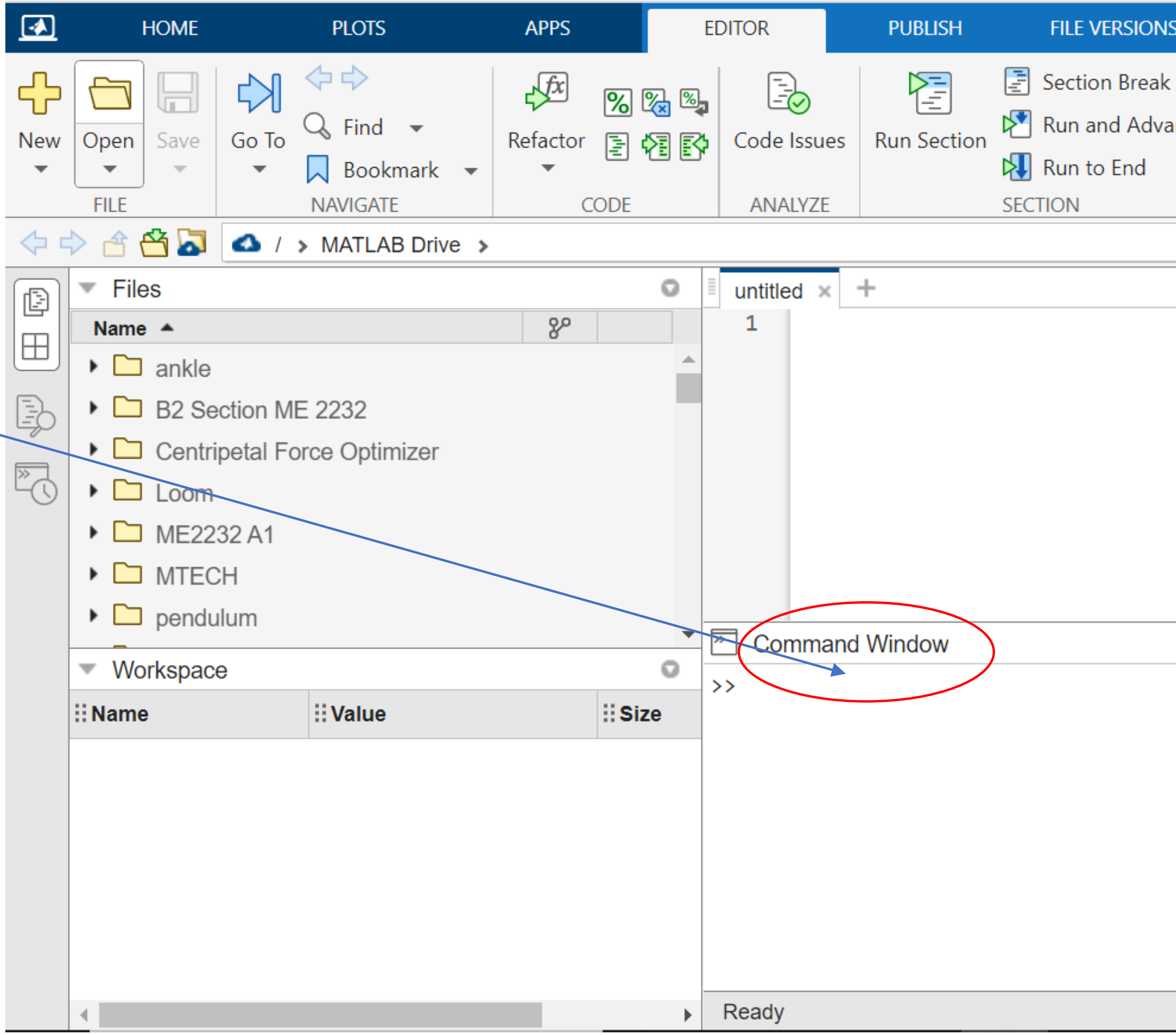
Introduction, aims and objectives of the course and discussion on course handout

Introduction to MATLAB environment and commands

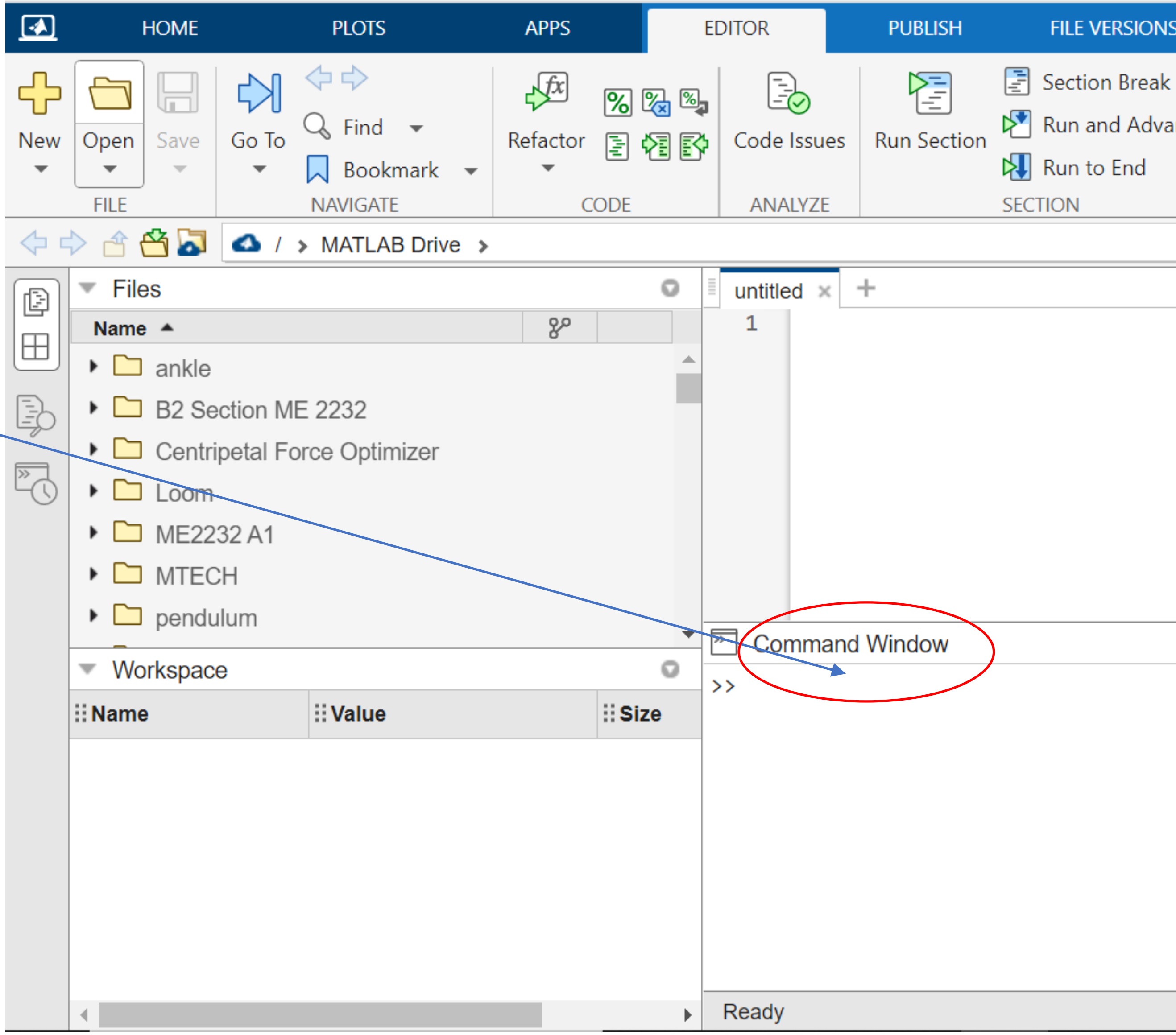
Basics

- On almost all systems, MATLAB works through three basic windows
  - Command Window
  - Figure Window
  - Editor Window

MANIPAL UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

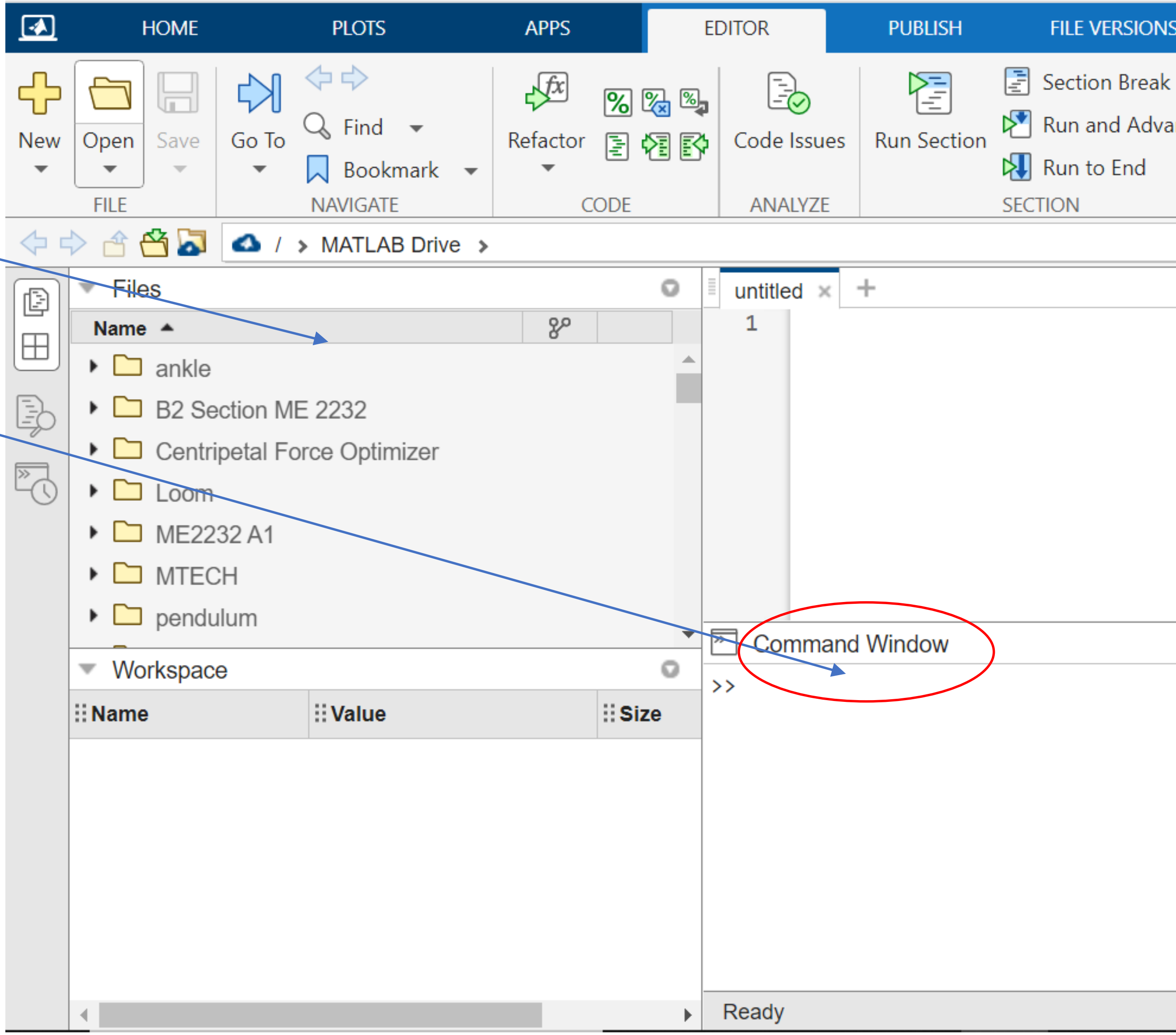


Command Window-
Type commands



Command Window-

Type commands

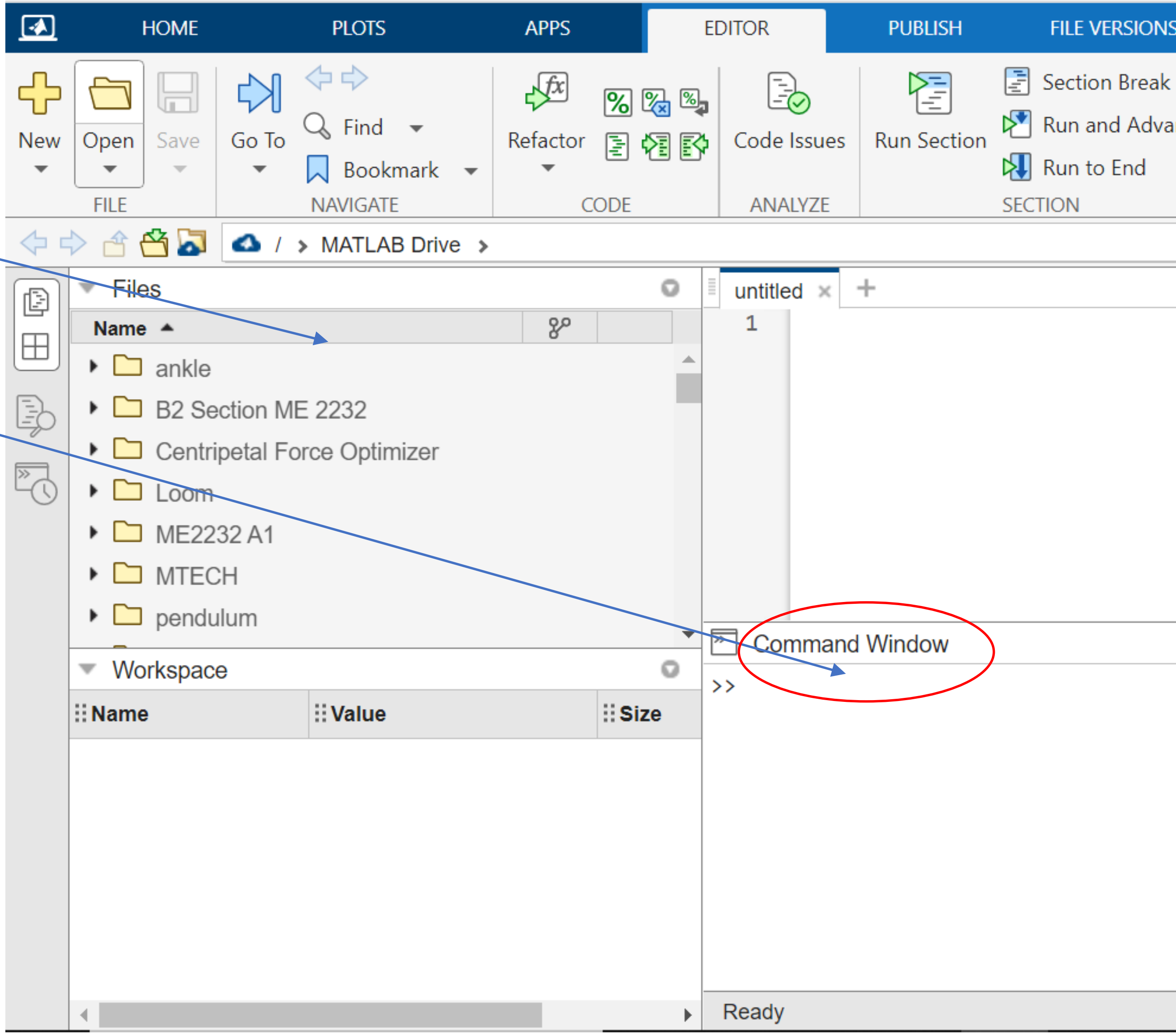


Current Directory-

.m files

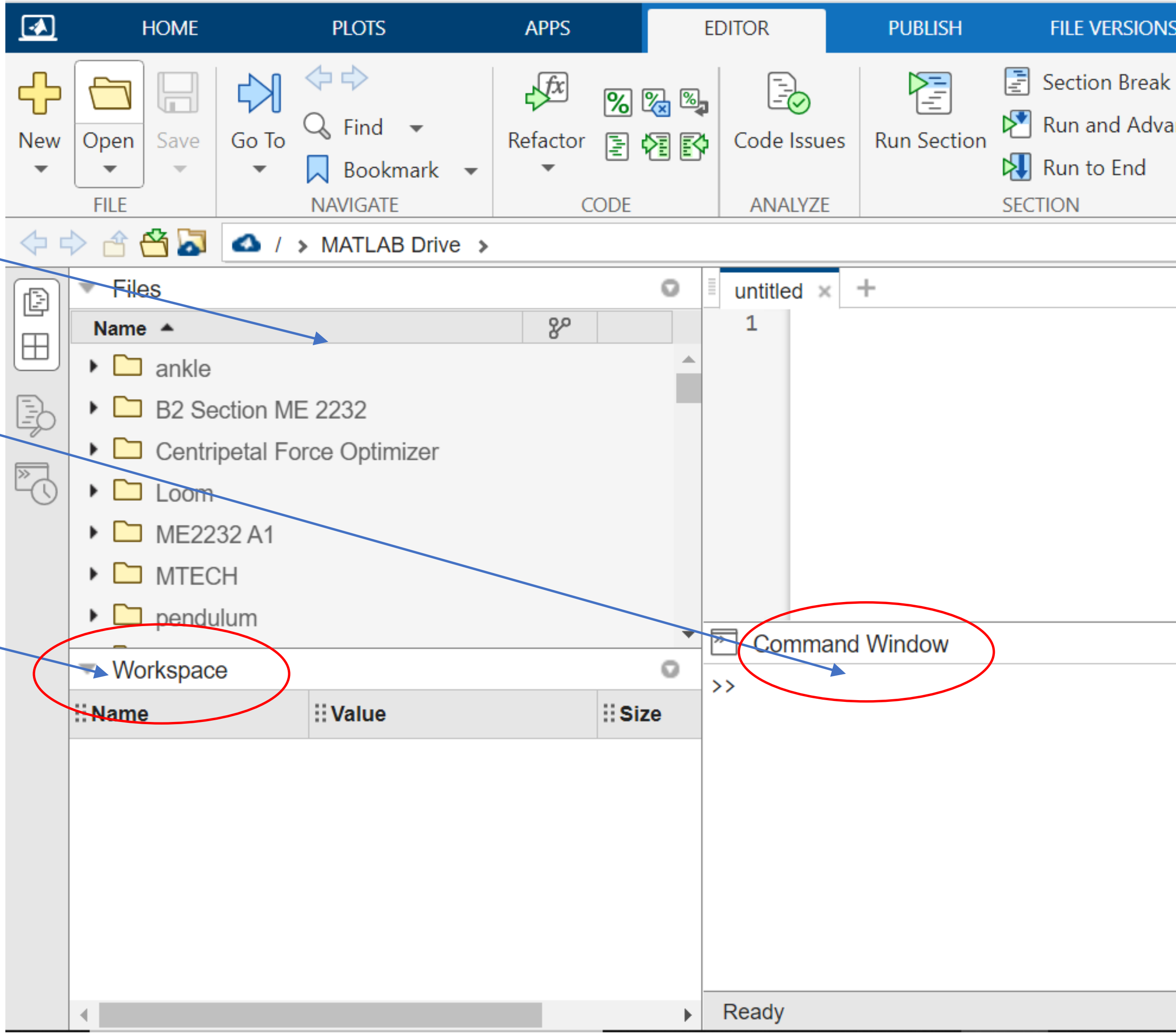
Command Window-

Type commands



Current Directory-
.m files

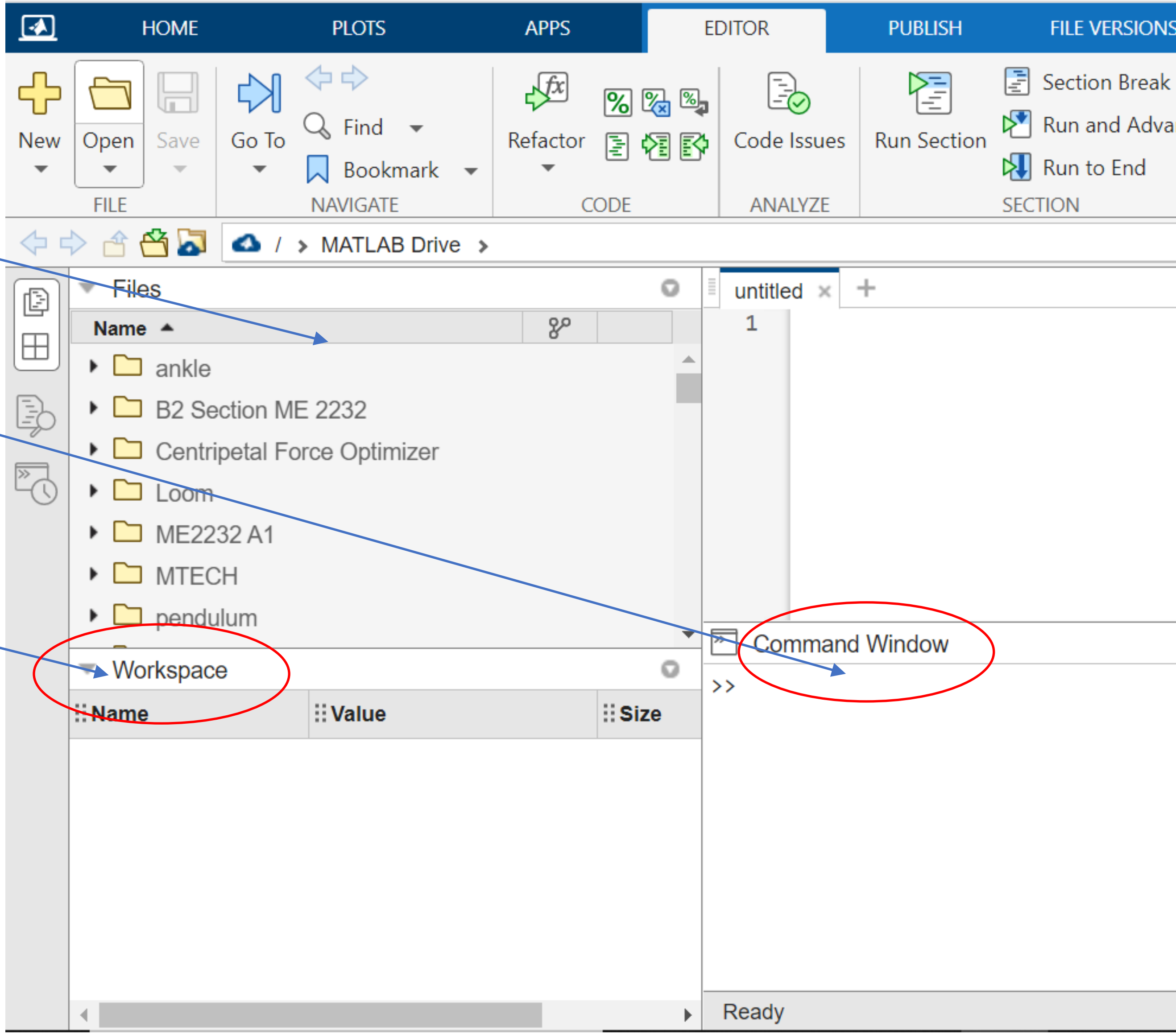
Command Window-
Type commands



Current Directory-
.m files

Command Window-
Type commands

Workspace: This subwindow lists all variables that you have generated so far and shows their type and size.



Recap

- The **command window** is where you'll give MATLAB its input and **view** its **output**.

- The **workspace** shows you all of your current working variables and other objects.

- The **history** shows you all commands you used in CW.

- The **Editor** for MATLAB scripts (M-files) . To save & run the m-file type 'F5'. To open the editor with a new or old m-file use the command **open** ***file_name***

Frequently Used Commands

what	List all m-files in current directory
dir/ls	List all files in current directory
type test	Display test.m in command window
delete test	Delete test.m
cd/chdir	Change directory
pwd	Show current directory
which test	Display directory path to ‘closest’ test.m
who	List known variables
whos	List known variables plus their size
clear	Clear variables from workspace
clc	Clear the command window

Frequently Used Commands

For help, command description etc use F1 or following commands:

- **help** *command_name*
- **helpwin** *command_name*
- **doc** *command_name*
- **helpdesk** *command_name*
- **demo** *command_name*
- **lookfor** *keyword* **(search unknown command)**

MATLAB and Matrices

MATLAB treats all variables as matrices. For our purposes a matrix can be thought of as an array, in fact, that is how it is stored.

Vectors are special forms of matrices and contain only one row OR one column

Scalars are matrices with only one row AND one column

Arrays and Vectors

An array is a list of numbers or expressions arranged in horizontal rows and vertical columns.

When an array has only one row or column, it is called a vector

An array with m rows and n columns is called a matrix of size $m \times n$.

Arrays and Vectors

You already know how to launch MATLAB. Go ahead and try the commands shown

A row vector *x* with three elements can be created as :

```
>> x = [1 2 3]

x =
        1        2        3
```

A row vector *y* with three elements can be created as :

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

y =
        2
        1
        5
```

Arrays and Vectors

You can also add (or subtract) two vectors of the same size.

```
>> z = [2 1 0];
>> a = x + z
```

```
a =
     3     3     3
```

But you cannot add (or subtract) a row vector to a column vector

```
>> b = x + y

??? Error using ==> plus
Matrix dimensions must agree.
```

Arrays and Vectors

You can multiply (or divide) the elements of two same-sized vectors term by term with the array operator .* (or . /)

```
>> a = x.*z

a =
        2        2        0
```

But multiplying a vector with a scalar does not need any special operation (no dot before the *)

```
>> b = 2*a

b =
        4        4        0
```

Arrays and Vectors

Create a vector *x* with 5 elements linearly spaced between 0 and 10.

```
>> x = linspace(0,10,5)
x =
         0    2.5000    5.0000    7.5000   10.0000
```

Trigonometric functions sin, cos , etc., as well as elementary math functions sqrt, exp, log, etc., operate on vectors term by term.

```
>> y = sin(x);
```

```
>> z = sqrt(x).*y
z =
         0    0.9463   -2.1442    2.5688   -1.7203
```

Class Assignment I

Equation of a straight line: The equation of a straight line is $y = mx + c$, where m and c are constants. Compute the y-coordinates of a line with slope $m = 0.5$ and the intercept $c = -2$ at the following x-coordinates:

$$x = 0, \quad 1.5, \quad 3, \quad 4, \quad 5, \quad 7, \quad 9, \text{ and } 10.$$

Multiply, divide, and exponentiate vectors: Create a vector t with 10 elements: 1, 2, 3, ..., 10. Now compute the following quantities:

- $x = t \sin(t)$.
- $y = \frac{t-1}{t+1}$.
- $z = \frac{\sin(t^2)}{t^2}$.

Class Assignment

Points on a circle: All points with coordinates $x = r\cos\theta$ and $y = r\sin\theta$, where r is a constant, lie on a circle with radius r, i.e., they satisfy the equation $x^2 + y^2 = r^2$. Create a column vector for θ with the values 0, $\pi/4$, $\pi/2$, $3\pi/4$, π, and $5\pi/4$. Take $r = 2$ and compute the column vectors x and y. Now check that x and y indeed satisfy the equation of a circle, by computing the radius $r = \sqrt{(x^2 + y^2)}$. [To calculate r you will need the array operator `.^` for squaring x and y. Of course, you could compute x^2 by `x.*x` also.]

Lecture 3 and 4

Creating and Working with Arrays of Numbers, Creating and Printing Simple Plots, Creating, Saving, and Executing a Script File

Working with Arrays and Matrices, Working with Anonymous Functions

The MATLAB commands used are

`plot`	creates a 2-D line plot,
`axis`	changes the aspect ratio of the x-axis and the y-axis,
`xlabel`	annotates the x-axis,
`ylabel`	annotates the y-axis,
`title`	puts a title on the plot, and
`print`	prints a hard copy of the plot.

Creating and Printing Plots

Draw a circle of unit radius.

- To do this, first generate the data (x- and y-coordinates of, say, 100 points on the circle), then plot the data, and finally print the graph.
- For generating data, use the parametric equation of a unit circle:

$$x = \cos\theta, \quad y = \sin\theta, \quad 0 \leq \theta \leq 2\pi.$$

Creating and Printing Plots

Draw a circle of unit radius.

- To do this, first generate the data (x- and y-coordinates of, say, 100 points on the circle), then plot the data, and finally print the graph.
- For generating data, use the parametric equation of a unit circle:

$$x = \cos\theta, \quad y = \sin\theta, \quad 0 \le \theta \le 2\pi.$$

Create a linearly spaced 100 elements-long vector θ .

```
>> theta = linspace(0,2*pi,100);
```

Calculate x- and y-coordinates.

```
>> x = cos(theta);
>> y = sin(theta);
```

Creating and Printing Plots

Draw a circle of unit radius.

- To do this, first generate the data (x- and y-coordinates of, say, 100 points on the circle), then plot the data, and finally print the graph.
- For generating data, use the parametric equation of a unit circle:

$$x = \cos\theta, \quad y = \sin\theta, \quad 0 \le \theta \le 2\pi.$$

MATLAB draws an ellipse rather than a circle because of its default rectangular axes.

The command axis (' equal ') directs MATLAB to use the same scale on both axes, so that a circle appears as a circle. You can also use axis (' square ') to override the default rectangular axes.

Plot x vs. y

```
>> plot(x,y)
>> axis('equal');
```

Set the length scales of the two axes to be the same.

```
>> axis('equal');
```

Label the x-axis with x.

Label the y-axis with y.

```
>> xlabel('x')
>> ylabel('y')
```

Creating and Printing Plots

Draw a circle of unit radius.

- To do this, first generate the data (x- and y-coordinates of, say, 100 points on the circle), then plot the data, and finally print the graph.
- For generating data, use the parametric equation of a unit circle:

$$x = \cos\theta, \quad y = \sin\theta, \quad 0 \le \theta \le 2\pi.$$

Put a title on the plot.
Print on the default printer.

```
>> title('Circle of unit radius')

>> print
```

Creating and Printing Plots

Class Assignment II

A simple sine plot: Plot $y = \sin x$, $0 \le x \le 2\pi$, taking 100 linearly spaced points in the given interval. Label the axes and put "Plot created by *yourname*" in the title.

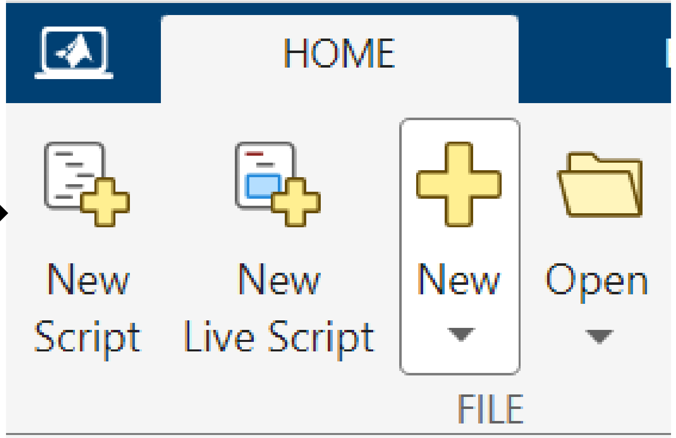
An exponentially decaying sine plot: Plot $y = e^{-0.4x} \sin x$, $0 \le x \le 4\pi$, taking 10, 50, and 100 points in the interval. [Be careful about computing y. You need array multiplication between `exp(-0.4*x)` and `sin(x)`.

Log-scale plots: The plot commands `semilogx`, `semilogy`, and `loglog` plot the x-values, the y-values, and both x- and y-values on a $\log_{10}$ scale, respectively. Create a vector `x=0:10:1000`. Plot x vs. x^3 using the three log-scale plot commands. [Hint: First, compute `y=x.^3` and then use `semilogx(x,y)`, etc.]

Creating, Saving, and Executing a Script File

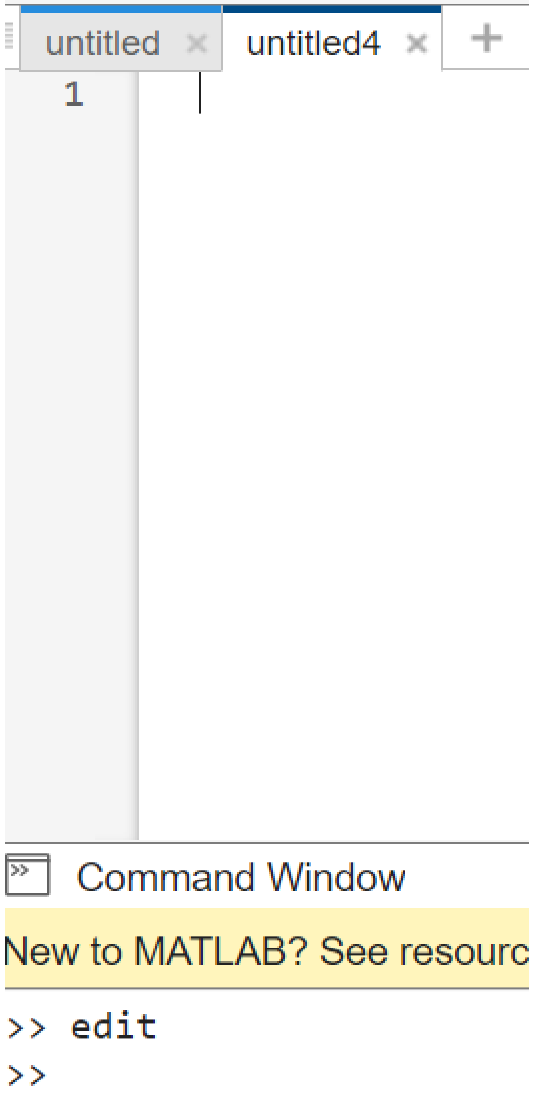
Create a new file:

- **On PCs and Macs:** Select File→New→Blank M-File from the File menu. A new edit window should appear.



In MATLAB online, a new file can be directly opened using "new script"

New script can also be opened using "edit" command



Creating, Saving, and Executing a Script File

Type the following lines into this file.

```
% CIRCLE - A script file to draw a unit circle
```

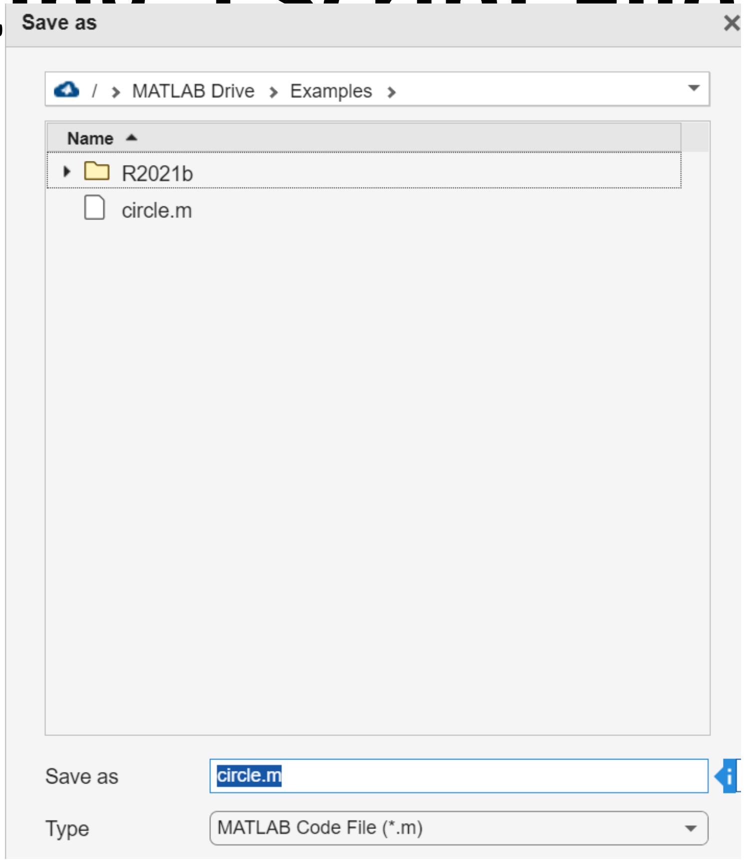
```
theta=linspace(0,2*pi,100);
x=cos(theta);
y=sin(theta);
plot(x,y);
axis('equal');
title('Circle of unit radius')
```

Write and save the file under the name circle.m:

Select Save As... from the File menu.

Type `circle.m` as the name of the document.

Click Save to save the file.




```
>> save

Saving to: /MATLAB Drive/Examples/matlab.mat
```

Creating, Saving, and Executing a Script File

Now get back to MATLAB and type the commands

```
>> help circle
```

Seek help on the script file to see if MATLAB can access it.

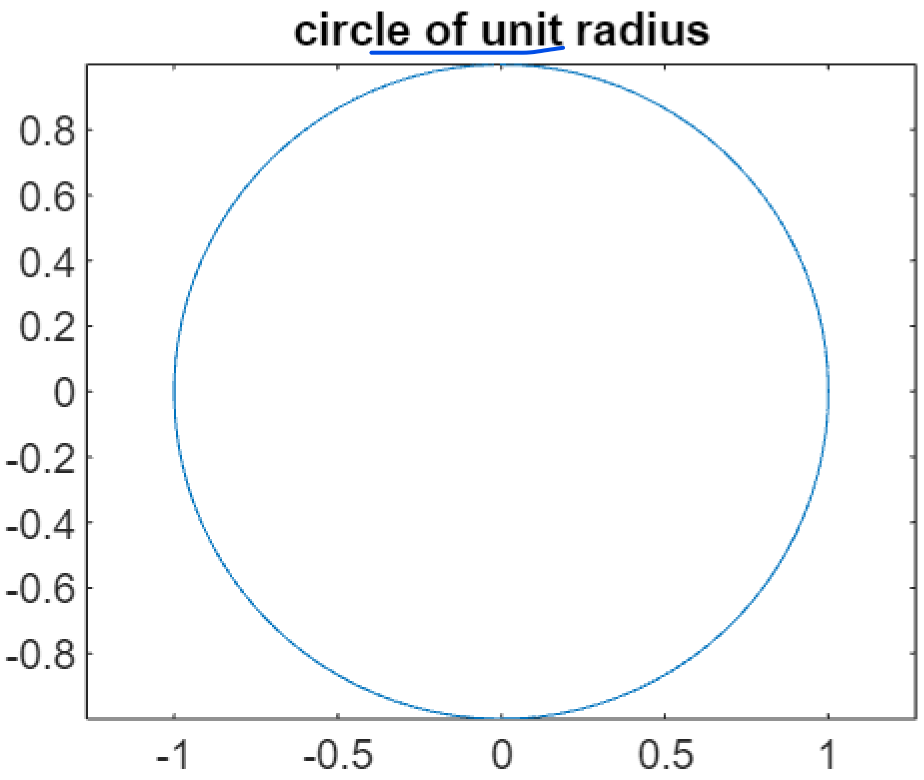
```
>> help circle
circle is a script.
```

MATLAB tells about the script

```
>> circle
```

Execute the file. You should see the circle plot in the figure window.

Creating, Saving, and Executing a Script File



If you have the script file open in the MATLAB editor window, you can execute the file by pressing the Run *file* icon (the little green arrowhead

Class Assignment III

1. **Show the center of the circle:** Modify the script file circle.m to show the center of the circle on the plot, too. Show the center point with a "+". (Hint:

2. **Change the radius of the circle:** Modify the script file circle.m to draw a circle of arbitrary radius r as follows:

   - Include the following command in the script file before the first executable line (`theta=...`) to ask the user to input (r) on the screen:

     ```
     r = input('Enter the radius of the circle:  ')
     ```

   - Modify the x- and y-coordinate calculations appropriately.
   - Save and execute the file. When asked, enter a value for the radius and press return.

Class Assignment III

3. **Variables in the workspace:** All variables created by a script file are left in the global workspace. You can get information about them and access them, too:

    - Type `who` to see the variables present in your workspace. You should see the variables `r`, `theta`, `x`, and `y` in the list.
    - Type `whos` to get more information about the variables and the workspace.
    - Type `[theta' x' y']` to see the values of θ, x, and y listed as three columns. All three variables are row vectors. Typing a single right quote ($'$) after their names transposes them and makes them column vectors.

Creating and executing function file

Open the script file circle.m:

- **On PCs:** Select File→Open... from the File menu. Navigate and select the file circle.m from the Open dialog box. Double-click to open the file. The contents of the file should appear in an edit window.

```
function [x,y] = circlefn(r);
theta=linspace(0,2*pi,100);     % create vector theta
x = r*cos(theta);               % generate x-coordinates
y = r*sin(theta);               % generate y-coordinates
plot(x,y);                      % plot the circle
axis('equal');                  % set equal scale on axes
title(['Circle of radius r =',num2str(r)])
                                % put a title with the value of r
```

Alternatively, you could select File→New→Function M-File and type all the lines in the new file.

Working with Arrays and Matrices

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

A =

     1     2     3
     4     5     6
     7     8     8
```

Matrices are entered row-wise. Rows are separated by semicolons and columns are separated by spaces or commas.

```
>> A(2,3)

ans =

     6
```

Element A_{ij} of matrix A is accessed as `A(i,j)`.

Working with Arrays and Matrices

Correcting any entry is easy through indexing.

```
>> A(3,3) = 9
```

```
A =

     1     2     3
     4     5     6
     7     8     9
```

Any submatrix of A is obtained by using range specifiers for row and column indices.

```
>> B = A(2:3,1:3)
```

```
B =

     4     5     6
     7     8     9
```

Working with Arrays and Matrices

The colon by itself as a row or column index specifies all rows or columns of the matrix.

```
>> B = A(2:3,:)

B =

     4     5     6
     7     8     9
```

A row or a column of a matrix is deleted by setting it to a null vector [].

```
>> B(:,2)=[]

B =

     4     6
     7     9
```

Matrices are transposed using the single right-quote character ('). Here x is the transpose of the first row of A.

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

>> x = A(1,:)'

x =

     1
     2
     3
```

Working with Arrays and Matrices

Matrix or vector products are well-defined between compatible pairs. A row vector (x') times a column vector (x) of the same length gives the inner product, which is a scalar, but a column vector times a row vector of the same length gives the outer product, which is a matrix.

```
>> x'*x

ans =

    14
```

```
>> x*x'

ans =

     1     2     3
     2     4     6
     3     6     9
```

```
>> A*x

ans =

    14
    32
    50
```

You can even exponentiate a matrix if it is a square matrix. A^2 is simply A*A.

```
>> A^2

ans =

    30    36    42
    66    81    96
   102   126   150
```

Working with Arrays and Matrices

When a dot precedes the arithmetic operators `*`, `^`, and `/`, MATLAB performs array operation (element-by-element operation). So, `A.^2` produces a matrix with elements $(a_{ij})^2$.

```
>> A.^2

ans =

     1     4     9
    16    25    36
    49    64    81
```

Class Assignment IV

1. **Entering matrices:** Enter the following three matrices.

$$A = \begin{bmatrix} 2 & 6 \\ 3 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad C = \begin{bmatrix} -5 & 5 \\ 5 & 3 \end{bmatrix}$$

2. **Check some linear algebra rules:**

- **Is matrix addition commutative?** Compute `A+B` and then `B+A`. Are the results the same? T

- **Is matrix addition associative?** Compute `(A+B)+C` and then `A+(B+C)` in the order prescribed. Are the results the same? T

- **Is multiplication with a scalar distributive?** Compute α`(A+B)` and αA+αB, taking $\alpha = 5$, and show that the results are the same.

- **Matrices are different from scalars!** For scalars, $ab = ac$ implies that $b = c$ if $a \neq 0$. Is that true for matrices? Check by computing `A*B` and `A*C` for the matrices given in Exercise 1. Also, show that `A*B` $\neq$ `B*A`.

Manipulate a matrix: Do the following operations on matrix G created in Exercise 4.

- Delete the last row and last column of the matrix.
- Extract the first 4×4 submatrix from G.
- Replace $G(5,5)$ with 4.
- What do you get if you type `G(13)` and hit return? Can you explain how MATLAB got that answer?
- What happens if you type `G(12,1)=1` and hit return?

$$G = \begin{bmatrix} 2 & 6 & 0 & 0 & 0 & 0 \\ 3 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -5 & 5 \\ 0 & 0 & 0 & 0 & 5 & 3 \end{bmatrix}.$$

Working with Anonymous Functions

An anonymous function is a function of one or more variables that you create on the command line for subsequent evaluation

- Anonymous functions are defined on the command line. They live in the MATLAB workspace and are alive as long as the MATLAB session lasts.

- You can define an anonymous function with any number of input variables.

- You must use a vectorized expression (using array operators) for the function if you intend to use an array as an input variable.

- You can use anonymous functions as input to other functions where appropriate.

Working with Anonymous Functions

The key to anonymous functions is the syntax of its creation:

$$fn\_name = @(list\ of\ input\ variables)\ function\_expression$$

Create a function

$f(x) = x^3 - 3x^2 + x\log(x-1) + 100$

```
>> f = @(x) x^3-3*x^2 +x*log(x-1)+100

f =

   @(x)x^3-3*x^2+x*log(x-1)+100
```

Working with Anonymous Functions

Evaluate the function at $x = 0$, i.e., find $f(0)$.

```
>> f(0)

ans =

   100
```

```
>> f(1)

ans =

  -Inf
```

Evaluate the function at $x = 1$. Note that f is singular at $x = 1$.

You can use f in an array also.

```
>> values = [f(0) f(1) f(2) f(10)]

values =

  100.0000      -Inf   96.0000  821.9722
```

Working with Anonymous Functions

```
>> x=[0 1 2 10];
>> f(x)
```

Using an array as the input to *f* causes an error. This is because the expression for *f* is not vectorized.

```
???? Error using ==> mpower
Matrix must be square.

Error in ==> @(x)x^3-3*x^2+x*log(x-1)+100
```

Redefine *f* by vectorizing the expression (use array operators). Now use it with an array argument.

```
>> f = @(x) x.^3-3*x.^2 +x.*log(x-1)+100;
f(x)
ans =

   100.0000      -Inf   96.0000  821.9722
```

You can also use *f* as input to other functions where appropriate.

```
>> x = linspace(-10,10);
>> plot(x,f(x))
```

Symbolic Computation

The most important step in carrying out symbolic computation is to declare the independent variables to be symbolic before you do anything with them.
Suppose you want to use x and y as symbolic variables

Declare x and y to be symbolic variables. Define a function f as

$$f = (x + y)^3 .$$

```
>> syms x y
```

```
>> f = (x+y)^3
```

```
f =

(x + y)^3
```

Use `expand` to multiply out and expand algebraic or trigonometric expressions.

```
>> expand(f)
ans =
x^3 + 3*x^2*y + 3*x*y^2 + y^3
```

Use `factor` to find factors of long algebraic expressions.

```
>> factor(ans)
ans =
(x + y)^3
```

Use pretty to get the expression in more readable form.

```
>> pretty(subs(exp2))
  2 a (5 a + 3)    6 a - 5 b
```

Symbolic Computation

Define a trigonometric expression.

```
>> z = sin(x+y);
>> expand(z)
ans =
cos(x)*sin(y) + cos(y)*sin(x)
```

Substitute $y = \pi - x$ in expression z.

```
>> subs(z, y, pi-x)
ans =
0
```

Differentiate z with respect to x, i.e., find $\frac{\partial z}{\partial x}$.

```
>> diff(z,x)
ans =
cos(x + y)
```

Find the second derivative of z with respect to x, i.e., find $\frac{\partial^2 z}{\partial x^2}$.

```
>> z_xx = diff(z,x,2)
z_xx =
-sin(x + y)
```

n

Symbolic Computation

Integrate z with respect to x from 0 to $\pi/2$, i.e., evaluate $\int_0^{\frac{\pi}{2}} z\,dx$.

```
>> int(z,x,0,pi/2)
ans =
cos(y) + sin(y)
```

Now declare (redefine) x and y to be real and evaluate the inner product. Since x and y are real,

$$v^T v = x^2 + y^2 .$$

```
>> syms x y real
>> inner_product
inner_product =
x^2 + y^2
```

Solve two simultaneous algebraic equations for x and y:

$$\begin{aligned} ax + by - 3 &= 0 \\ -x + 2ay - 5 &= 0 . \end{aligned}$$

```
>> syms a b
>> exp1 = 'a*x + b*y -3';
>> exp2 = '-x + 2*a*y -5';
```

```
>> [x,y] = solve(exp1, exp2)
x =
(6*a - 5*b)/(2*a^2 + b)
y =
(5*a + 3)/(2*a^2 + b)
```

Symbolic Computation

Solve two simultaneous algebraic equations for x and y:

$$\begin{aligned} ax + by - 3 &= 0 \\ -x + 2ay - 5 &= 0 . \end{aligned}$$

```
>> syms a b
>> exp1 = 'a*x + b*y -3';
>> exp2 = '-x + 2*a*y -5';
```

```
>> [x,y] = solve(exp1, exp2)
x =
(6*a - 5*b)/(2*a^2 + b)
y =
(5*a + 3)/(2*a^2 + b)
```

Substitute the values of x and y just found in `exp1` to check the result.

```
>> subs(exp1)
ans =
(b*(5*a + 3))/(2*a^2 + b)+(a*(6*a - 5*b))/(2*a^2 + b)-3
```

Simplify the answer to see if it reduces to zero (as it must in order to satisfy the equation).

```
>> simplify(ans)
ans =
0
```

Class Assignment V

Solving simultaneous linear equations: Solve the following nonlinear algebraic equations simultaneously.

$$\begin{aligned} 3x^3 + x^2 - 1 &= 0. \\ x^4 - 10x^2 + 2 &= 0. \end{aligned}$$

Importing and Exporting Data

Mat-file: This is MATLAB's native binary format file for saving data. Two commands , s ave and load make it particularly easy to save data into and load data from these files.

M-file: If you have a text file containing data, or you want to write a text file containing data that you would eventually like to read in MATLAB , making it an M-file may be an excellent option.

Clear the MATLAB workspace (all variables are deleted).

```
>> clear all
>> theta=linspace(0,2*pi,201);
>> r = sqrt(abs(2*sin(4*theta)));
>> x = r.*cos(theta);
>> y = r.*sin(theta);
```

```
>> f = char('sqrt(abs(2*sin(4*theta)))');
```

Save variables x, y and f in a binary (Mat) datafile `xydata.mat`. The file is created by the `save` command.

```
>> save xydata x y f
>> whos
```

Name	Size	Bytes	Class
f	1x27	54	char
r	1x201	1608	double
theta	1x201	1608	double
x	1x201	1608	double
y	1x201	1608	double

Importing and Exporting Data

Clear all variables, query to see no variables are present, load the datafile, and query the workspace again to see the loaded variables.

```
>> clear all
>> whos
>> load xydata
>> whos
```

Name	Size	Bytes	Class	Attributes
f	1x27	54	char	
x	1x201	1608	double	
y	1x201	1608	double	

The newly loaded variables are *f*, *x* and *y*. Use these variables to verify the data they contain.

```
>> plot(x,y),axis('square');
>> f

f =
sqrt(abs(2*sin(4*theta)))
```

You can also load only selected variables from the Mat-file.

```
>> clear all
>> load xydata x y
>> whos
```

Name	Size	Bytes	Class	Attributes
x	1x201	1608	double	
y	1x201	1608	double	

Importing and Exporting Data

```
% TempData: Script file containing data on monthly maximum temperature
Sl_No = [1:12]';
Month = char('January','February','March','April','May','June',...
        'July','August','September','October','November','December');
Ave_Tmax = [22 25 30 34 36 30 29 27 24 23 21 20]';
```

Clear the MATLAB workspace.
Execute the *script file* `TempData`
and check the new variables.

```
>> clear all
>> TempData
>> whos
```

```
Name          Size        Bytes  Class     Attributes
Ave_Tmax      12x1           96  double
Month         12x9          216  char
Sl_No         12x1           96  double
```

Check one of the variables, `Month`, to see the data it contains. All data typed in file `TempData` is loaded.

Using script files to load data typed in them is a very safe and sure shot method of getting data in the MATLAB workspace.

```
>> Month

Month =

January
February
March
April
...
```

Read data from an MS Excel file that contains a header line, text, and numeric data in three columns.

```
>>[A,txt,raw] = xlsread('TempData.xls');
```

```
>> raw
```

```
raw =
     'SN'     'Month'       'Ave. Tmax.'
     [ 1]     'January'     [        22]
     [ 2]     'February'    [        25]
     [ 3]     'March'       [        30]
     ...
     [11]     'November'    [        21]
     [12]     'December'    [        20]
```