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

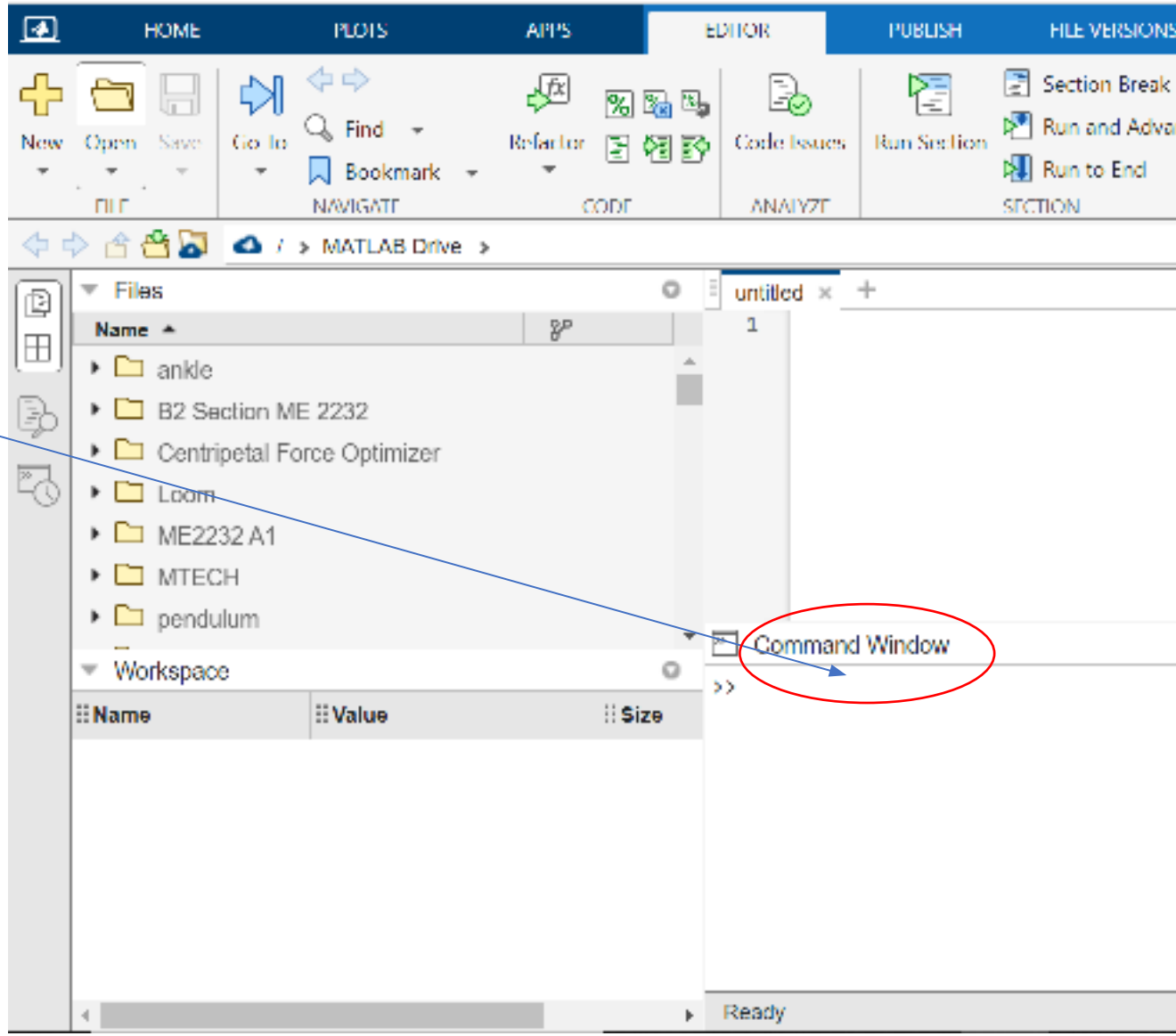
Lecture 1 and 2

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

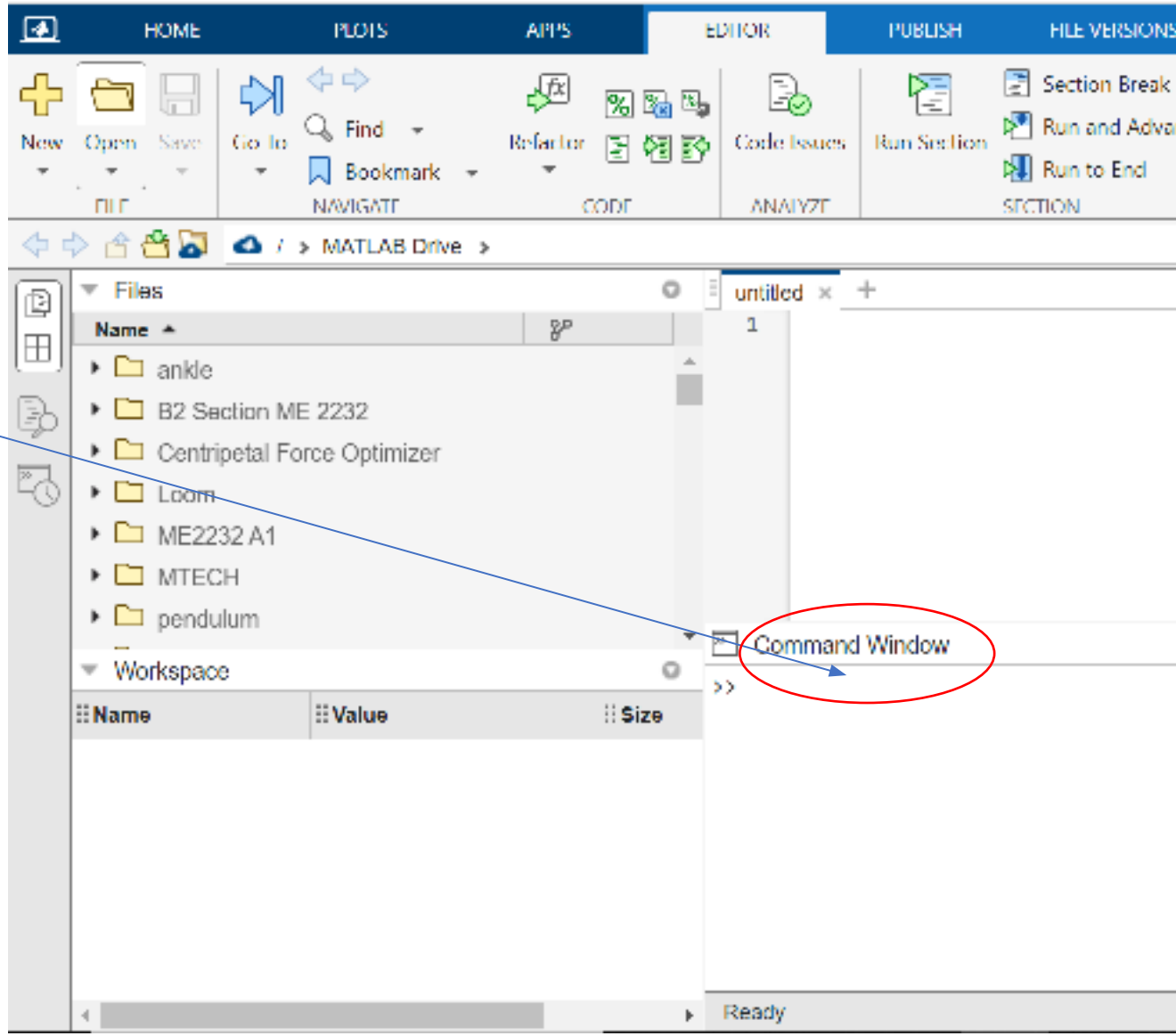
Introduction to MATLAB environment and commands

Basics

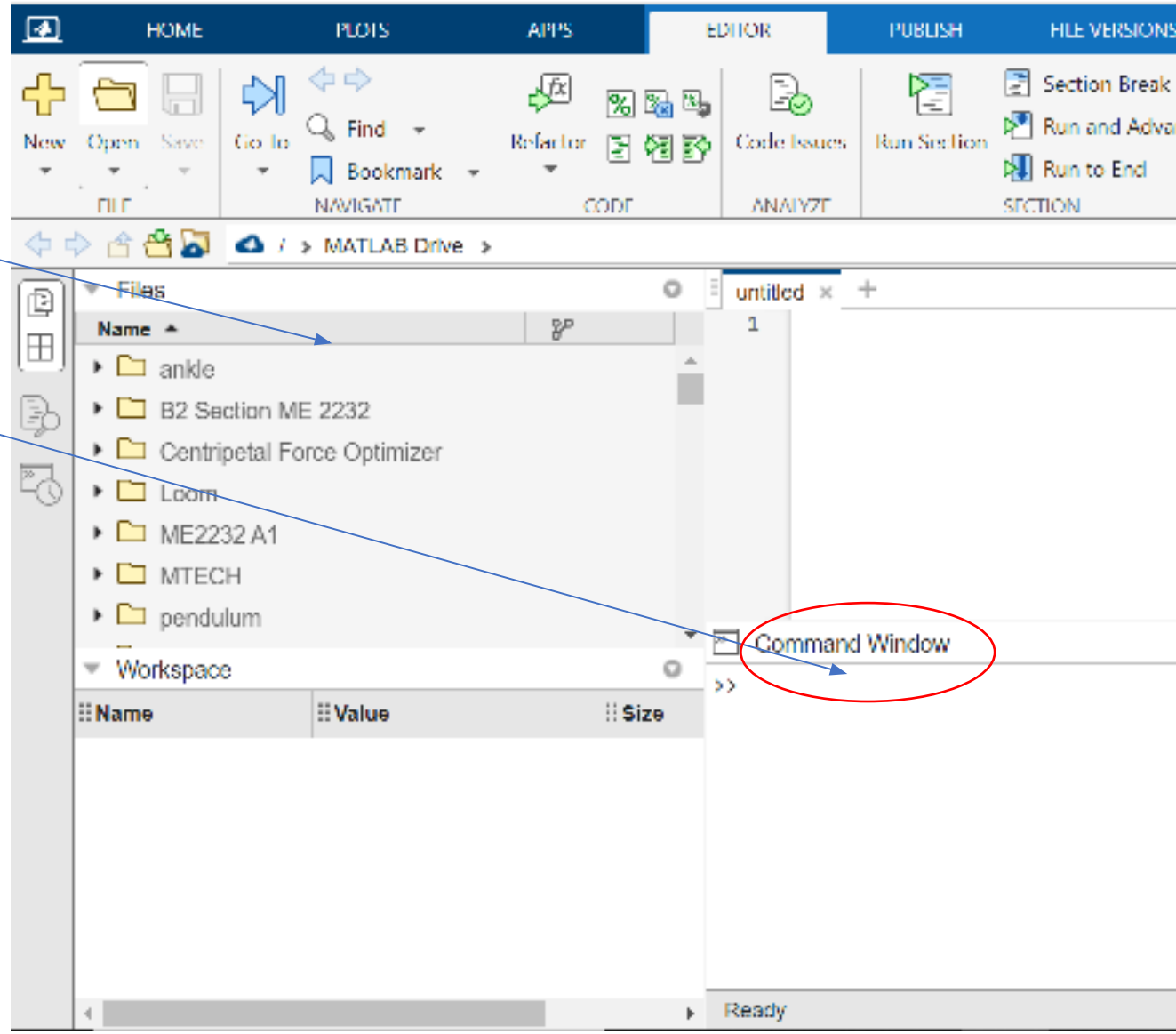
- On almost all systems, MATLAB works through three basic windows
  - Command Window
  - Figure Window
  - Editor Window


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
New
Open
Save
Go To
Find
Bookmark
Refactor
Code Issues
Run Section
Section Break
Run and Adva
Run to End
MATLAB Drive
Files
Name
ankle
B2 Section ME 2232
Centripetal Force Optimizer
Loom
ME2232 A1
MTECH
pendulum
untitled
1
Command Window
>>
Workspace
Name
Value
Size
Ready

Command Window-
Type commands

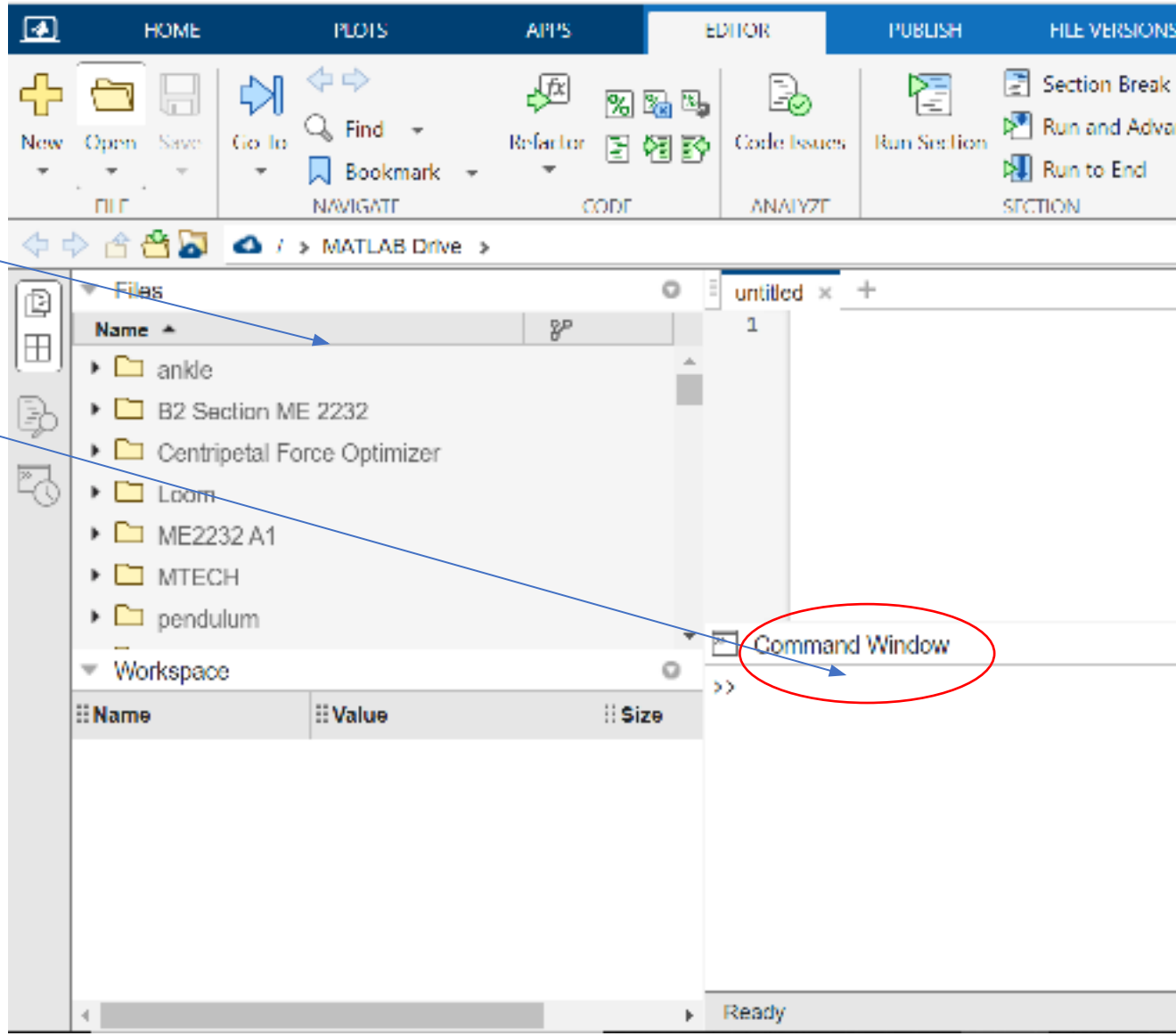


Command Window-
Type commands



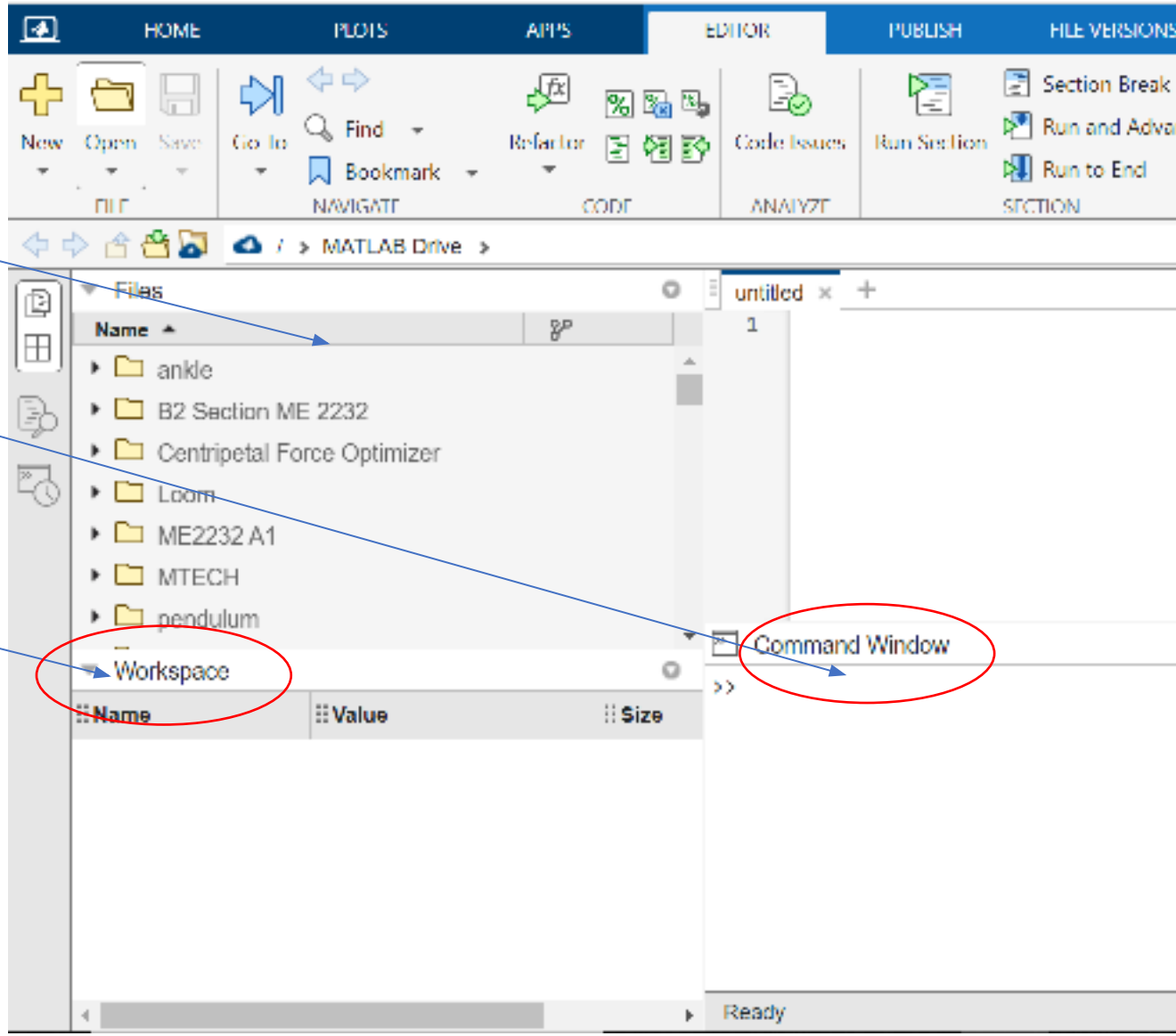
Current Directory-
.m files

Command Window-
Type commands



Current Directory-
.m files

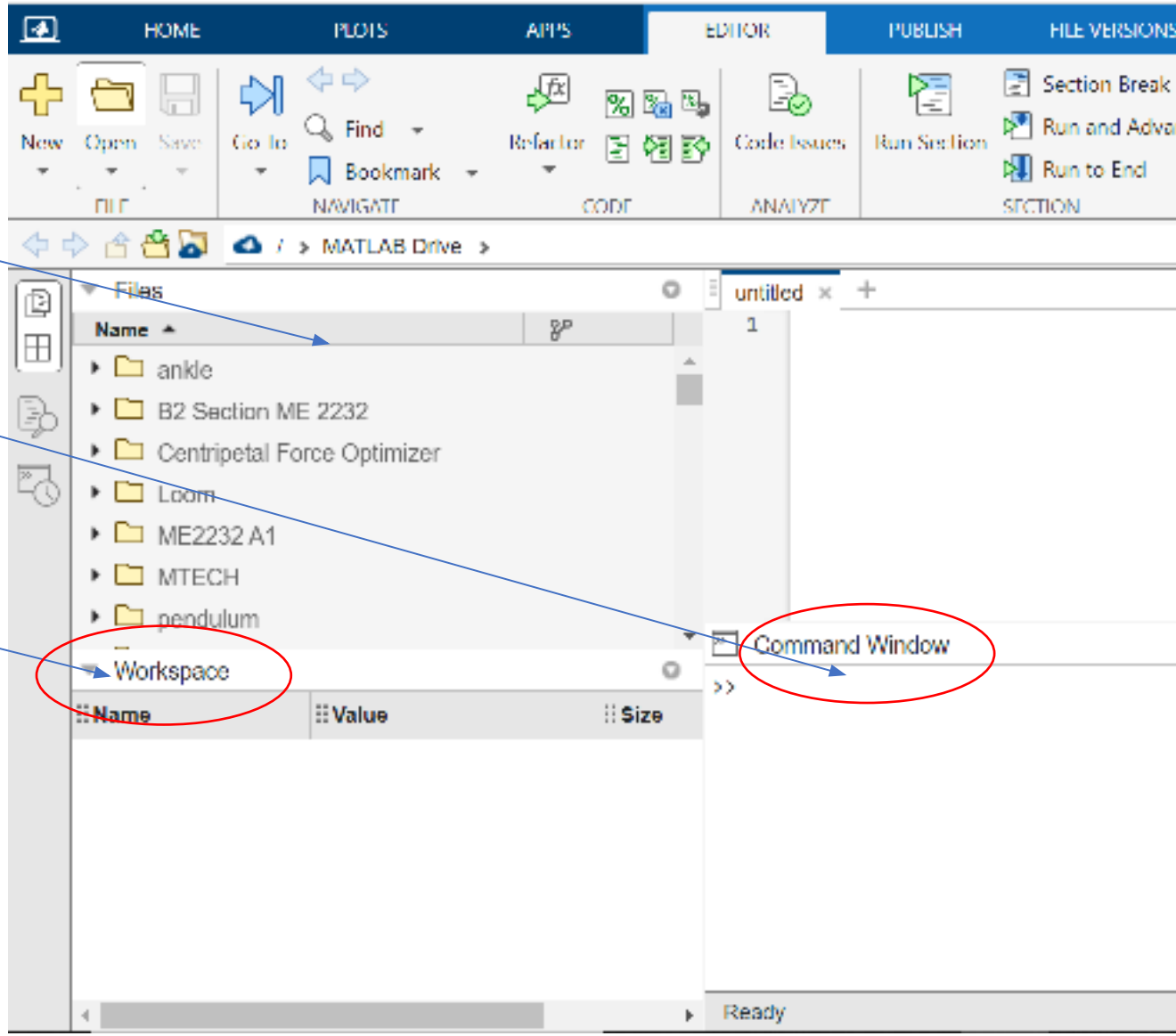
Command Window-
Type commands



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 x 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:

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

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

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')
```

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 $y = \sin x, \quad 0 \leq x \leq 2\pi$, 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 $y = e^{-0.4x} \sin x, \quad 0 \leq x \leq 4\pi$, 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)`

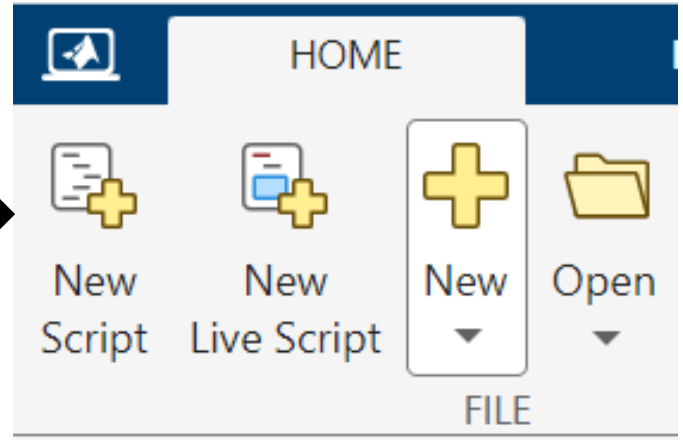
Log-scale plots: The plot commands `semilogx`, `semilogy`, and `loglog` plot the x-values, the y-values, and both x- and y-values on a $\log_{10}$ scale, respectively. Create a vector `x=0:10:1000`. Plot x vs. x^3 using the three log-scale plot commands. [Hint: First, compute `y=x.^3` and then use `semilogx(x,y)`,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

Creating, Saving, and Executing a Script File

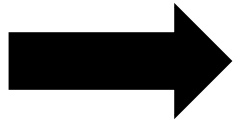
Create a new file:

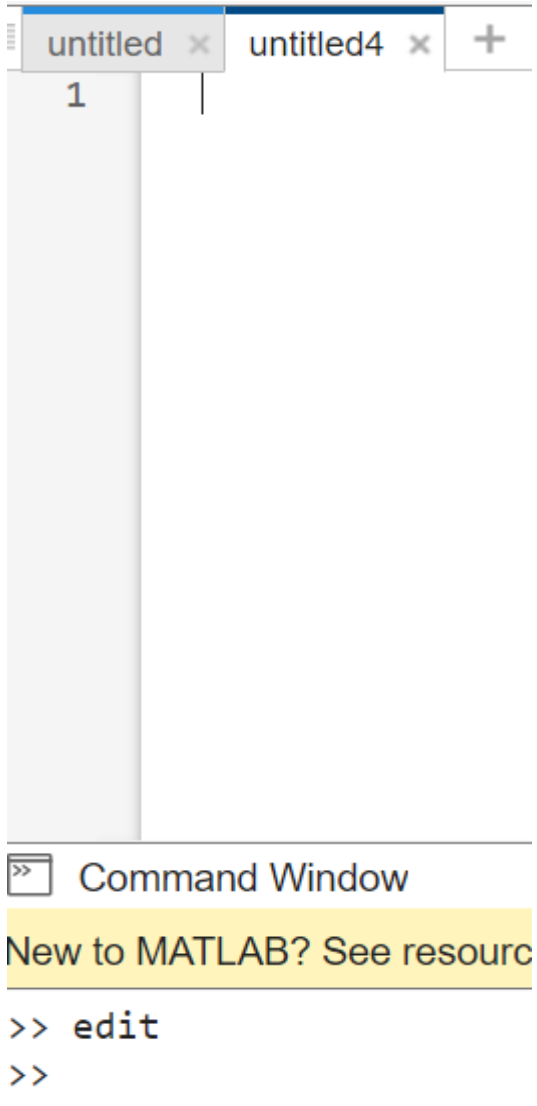
- **On PCs and Macs:** Select File→New→Blank M-File from the File menu. A new edit window should appear.

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.

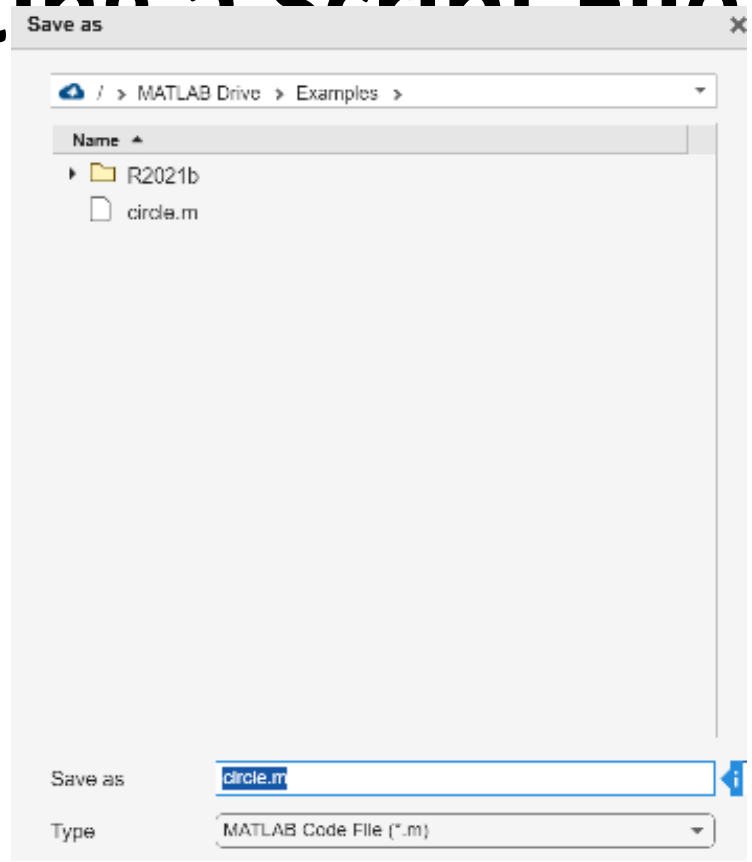
```
% CIRCLE - A script file to draw a unit circle

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.

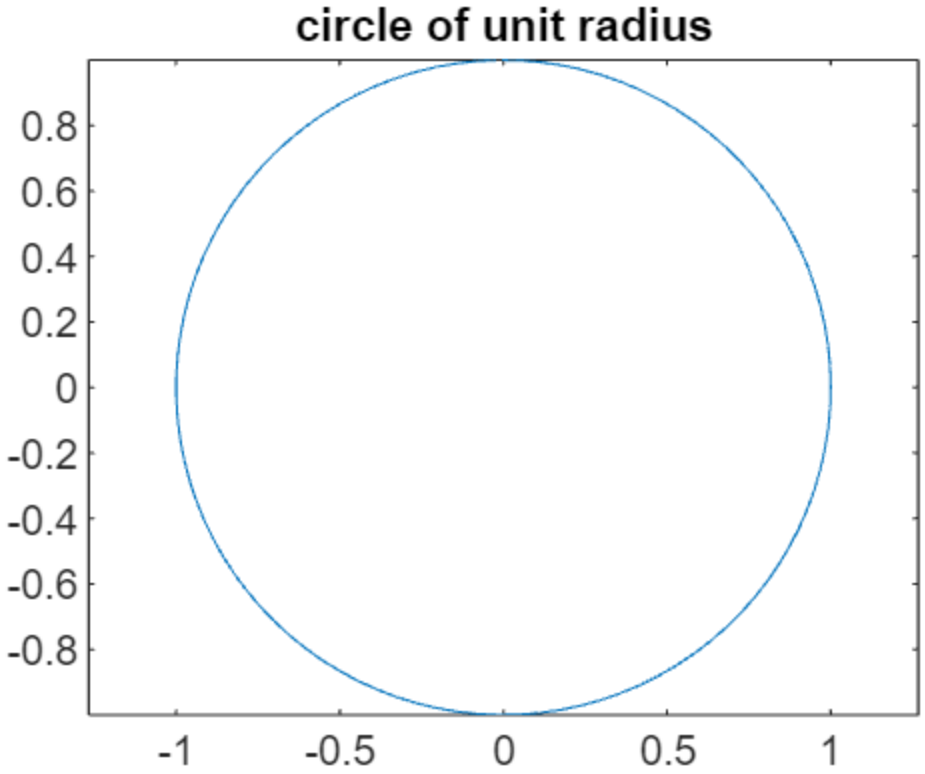
```
>> help circle
circle is a script.
```

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]
```

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

```
A =

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

```
>> 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.

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

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

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

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

- **Is multiplication with a scalar distributive?** Compute α`(A+B)` and α`A`+α`B`, 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
------------- - --------- - 5
       2                2
    2 a  + b         2 a  + 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)
```

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
```

```
v = [x; y];
inner_product = v'*v
```

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]
```

Continuation

If it is not possible to type the entire input on the same line, then use three consecutive periods (. . .) to signal continuation and continue the input on the next line.

The three periods are called an ellipsis.

```
A = [1/3    5.55*sin(x)   9.35   0.097;...
     3/(x+2*log(x))    3    0    6.555; ...
     (5*x-23)/55    x-3    x*sin(x)    sqrt(3)];
```

Matrix manipulation

Create a 4 x 3 random matrix *A*.

```
>> A = rand(4,3)

A =

    0.8147    0.6324    0.9575
    0.9058    0.0975    0.9649
    0.1270    0.2785    0.1576
    0.9134    0.5469    0.9706
```

Get those elements of A that are located in rows 3 to 4 and columns 2 to 3.

```
>> A(3:4, 2:3)

ans =
    0.2785    0.1576
    0.5469    0.9706
```

Add a fourth column to A and set it equal to the first column of A.

```
>> A(:,4) = A(:,1)
```

```
A =
    0.8147    0.6324    0.9575    0.8147
    0.9058    0.0975    0.9649    0.9058
    0.1270    0.2785    0.1576    0.1270
    0.9134    0.5469    0.9706    0.9134
```

Replace the last 3 x 3 submatrix of A (rows 2 to 4, columns 2 to 4) by a 3 x 3 identity matrix.

```
>> A(2:4,2:4) = eye(3)
```

```
    0.8147    0.6324    0.9575    0.8147
    0.9058    1.0000         0         0
    0.1270         0    1.0000         0
    0.9134         0         0    1.0000
```

Delete the first and third rows of A.

```
A([1 3],:) = []
```

Round off all entries of A.

```
>> A = round(A)

A =

     1     1     0     0
     1     0     0     1
```

```
A =

    0.9058    1.0000         0         0
    0.9134         0         0    1.0000
```

String out all elements of A in a row (note the transpose at the end).

```
>> A(:)'

ans =

     1     1     1     0     0     0     0     1
```

From a given matrix, picking out either a range of rows or columns or a set of noncontiguous rows or columns is straightforward

$$Q = \begin{bmatrix} 2 & 3 & 6 & 0 & 5 \\ 0 & 0 & 20 & -4 & 3 \\ 1 & 2 & 3 & 9 & 8 \\ 2 & -5 & 5 & -5 & 6 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix} \quad \text{and} \quad v = \begin{bmatrix} 1 & 4 & 5 \end{bmatrix},$$

$$\texttt{Q(v,:)} = \begin{bmatrix} 2 & 3 & 6 & 0 & 5 \\ 2 & -5 & 5 & -5 & 6 \\ 5 & 10 & 15 & 20 & 25 \end{bmatrix}$$

$$\texttt{Q(:,v)} = \begin{bmatrix} 2 & 0 & 5 \\ 0 & -4 & 3 \\ 1 & 9 & 8 \\ 2 & -5 & 6 \\ 5 & 20 & 25 \end{bmatrix}.$$

Reshaping matrices

As a vector: All the elements of matrix A can be strung into a single-column vector b by the command b = A (:) (matrix A is stacked in vector b columnwise)

If matrix A is an m x n matrix, it can be reshaped into a p x q matrix, as long as mx n = p x q, with the command re shape (A , p, q) .

`reshape(A,9,4)` transforms A into a 9×4 matrix, and
`reshape(A,3,12)` transforms A into a 3×12 matrix.

Appending a row or column

A row can be easily appended to an existing matrix, provided the row has the same length as the length of the rows of the existing matrix.

A= [A ; u] appends a row vector u to the rows of A, while A= [A v] appends a column vector v to the columns of A.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad u = \begin{bmatrix} 5 & 6 & 7 \end{bmatrix}, \text{ and } v = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix},$$

`A1 = [A; u]` produces $A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 5 & 6 & 7 \end{bmatrix}$, a 4×3 matrix,

`A2 = [A v]` produces $A_2 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \end{bmatrix}$, a 3×4 matrix,

`A3 = [A u']` produces $A_3 = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 7 \end{bmatrix}$, a 3×4 matrix,

`A4 = [A u]` produces an error,

`B = []; B = [B; 1 2 3]` produces $B = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$, and

`B = []; for k = 1:3, B = [B; k k+1 k+2]; end` produces $B = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$

Utility matrices

`eye(m,n)`	returns an m by n matrix with ones on the main diagonal,
`zeros(m,n)`	returns an m by n matrix of zeros,
`ones(m,n)`	returns an m by n matrix of ones,
`rand(m,n)`	returns an m by n matrix of random numbers,
`randn(m,n)`	returns an m by n matrix of normally distributed numbers,
`diag(v)`	generates a diagonal matrix with vector v on the diagonal,
`diag(A)`	extracts the diagonal of matrix A as a vector, and
`diag(A,1)`	extracts the first upper off-diagonal vector of matrix A.

Relational operations

`<`	less than
`<=`	less than or equal
`>`	greater than
`>=`	greater than or equal
`==`	equal
`~=`	not equal

Examples: If $x = [1\ \ 5\ \ 3\ \ 7]$ and $y = [0\ \ 2\ \ 8\ \ 7]$, then

`k = x < y`	results in $k = [0\ 0\ 1\ 0]$	because $x_i < y_i$ for $i = 3$,
`k = x <= y`	results in $k = [0\ 0\ 1\ 1]$	because $x_i \leq y_i$ for $i = 3$ and 4,
`k = x > y`	results in $k = [1\ 1\ 0\ 0]$	because $x_i > y_i$ for $i = 1$ and 2,
`k = x >= y`	results in $k = [1\ 1\ 0\ 1]$	because $x_i \geq y_i$ for $i = 1, 2$, and 4,
`k = x == y`	results in $k = [0\ 0\ 0\ 1]$	because $x_i = y_i$ for $i = 4$, and
`k = x ~= y`	results in $k = [1\ 1\ 1\ 0]$	because $x_i \neq y_i$ for $i = 1, 2$, and 3.

These operations result in a vector or matrix of the same size as the operands, with 1 where the relation is true and 0 where it is false.

A=[1 0;1 0]; B=[1 1;0 0]; xor(A,B) ans = 0 1 1 0

Logical operations

`&`	logical AND
`|`	logical OR
`~`	logical complement (NOT)
`xor`	exclusive OR

These operators work in a similar way as the relational operators and produce vectors or matrices of the same size as the operands, with 1 where the condition is true and 0 where false.

Examples: For two vectors $x = [0\ 5\ 3\ 7]$ and $y = [0\ 2\ 8\ 7]$,

`m = (x>y)&(x>4)` results in $m = [0\ 1\ 0\ 0]$, because the condition is true only for x_2,
`n = x|y` results in $n = [0\ 1\ 1\ 1]$, because either x_i or y_i is nonzero for $i = [2\ 3\ 4]$,
`m = ~(x|y)` results in $m = [1\ 0\ 0\ 0]$, which is the logical complement of `x|y`, and
`p = xor(x,y)` results in $p = [0\ 0\ 0\ 0]$, because there is no such index i for which x_i or y_i, but not both, is nonzero.

XOR(S,T) is the logical symmetric difference of elements S and T. The result is logical 1 (TRUE) where either S or T, but not both, is nonzero. The result is logical 0 (FALSE) where S and T are both zero or nonzero. S and T must have the same dimensions (or one can be a scalar). A=[1 0;1 0]; B=[1 1;0 0];
xor(A,B) ans = 0 1; 1 0

Character strings

All character strings are entered within two single right-quote characters

```
message = 'Leave me alone'
```

```
names = ['John'; 'Ravi'; 'Mary'; 'Xiao']
howdy = char('Hi','Hello','Namaste').
```

Manipulating character strings

Character strings can be manipulated just like matrices.

```
c = [howdy(2,:)  names(3,:)]
```

produces `Hello Mary` as the output for `c`

Relational Operators

Relational Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal
~=	Not equal

Ex- a=[1 2 3]; b=[1 2 2]; a==b returns ans=1 1 0 since the last case did not match.

Logical Operators and Functions

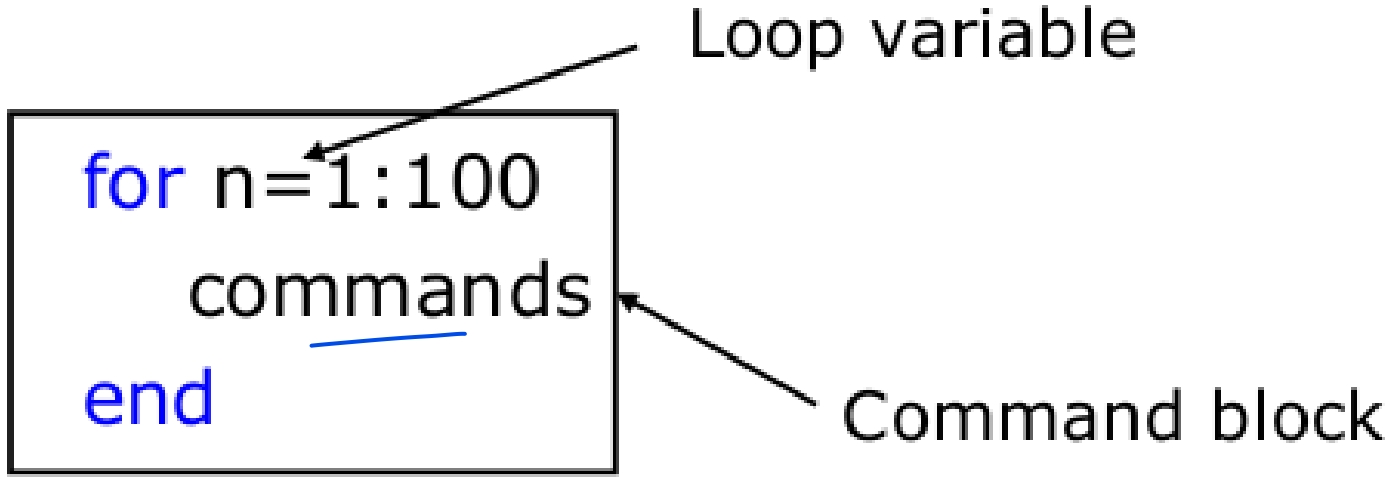
Logical Operator	Meaning
&	And
|	Or
~	Not

Flow Control

- Matlab has four kinds of statements which can be used to control the flow through your code:
  - ***if***, ***else*** and ***elseif*** execute statements based on a logical test
  - ***switch***, ***case*** and ***otherwise*** execute groups of statements based on a logical test
  - ***while*** and ***end*** execute statements an indefinite number of times, based on a logical test
  - ***for*** and ***end*** execute statements a fixed number of times

For loops

- It allows a group of statements to be repeated a fixed number of times.



- expression may be of the form i:k or i:j:k

  where *i* is beginning value, *k* is ending value and *j* is increment.

```
for ii=1:5
    x=ii*ii
end
```

```
for x = 1:10
        y(x) = x.^2+3*x+5;
end
```

```
A = rand(1,5);
for a = A
    disp(a)
end


for k = [ 1 7 3 pi i]
    disp(k)
end
```

```
n = 5; A = eye(n);
for j=2:n
    for i=1:j-1
        A(i,j)=i/j;
        A(j,i)=i/j;
    end
end
```

Example: Suppose we want a function that accepts an integer *N*, and then returns an *N*-by-*N* matrix with the following pattern: Using a '*for*' loop.

Pattern:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 2 & 0 & \dots & 0 \\ 1 & 2 & 3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \dots & \dots & \dots & N \end{bmatrix}$$

Solution:

```
1   function [A]=func_2_4(N)
2     A=zeros(N); %initialization (N-by-N zeros)
3   for k=1:N,
4       A(k,1:k)=1:k;
5   end
```

>> A=func_2_4(5)

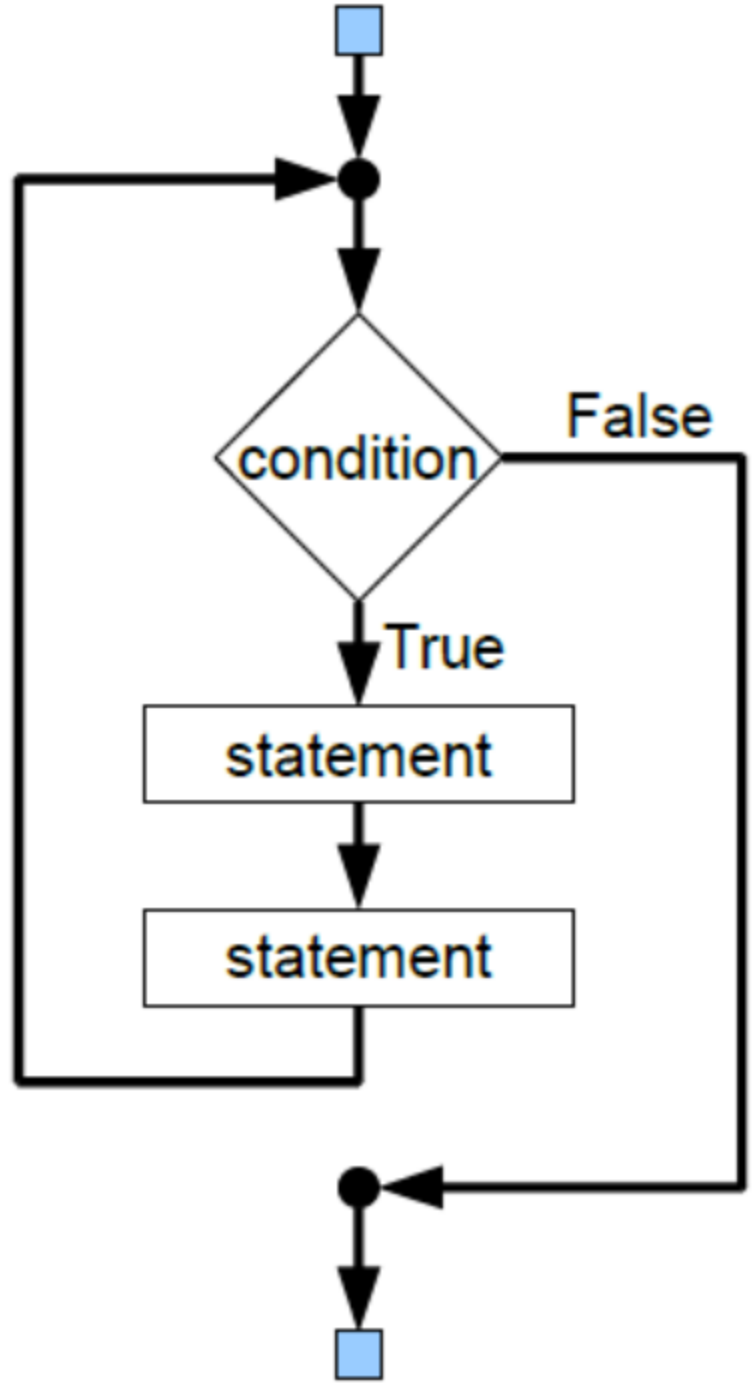
Output of *'func_2_4.m'* when input=5.

```
>> A=func_2_4(5)
A =
     1     0     0     0     0
     1     2     0     0     0
     1     2     3     0     0
     1     2     3     4     0
     1     2     3     4     5
```

While Loops

- A while loop allows one to repeat a group of statements as long as the specified condition is satisfied.
  - Don't need to know number of iterations

- Beware of infinite loops!


condition
False
True
statement
statement

```
x = 1
while x <= 10
        x = 3*x
end
```

```
A = 7; %Want to find the square root of A.
x=1; %First guess
err=1;   %Set error to something to get started
while(err>0.0001)
x = (x.^2+A)./(2*x); %Iteration equation (Newton Raphson)
err = abs(A-x.^2); % calculate the error
end
disp(x);
```

Example: Now suppose we want to rework the for loop example, this time using a 'while' loop.

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 2 & 0 & \dots & 0 \\ 1 & 2 & 3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \dots & \dots & \dots & N \end{bmatrix}$$

```
1   function [A]=func_2_5(N)
2   A=zeros(N); %initialization (N-by-N zeros)
3   k=1; %initialization
4   while k<=N,
5       A(k,1:k)=1:k;
6       k=k+1;
7   end
```

➔ Branching and Conditional statements

```
if expression,
        statements
end
```

'**expression**' a logical expression using
a logical operator such as (==), (>=),
(<=), (~=), (&), (|);
output of this expression is **True / False;**

Example:

if 2==3

disp("2 and 3 are not equal";

end

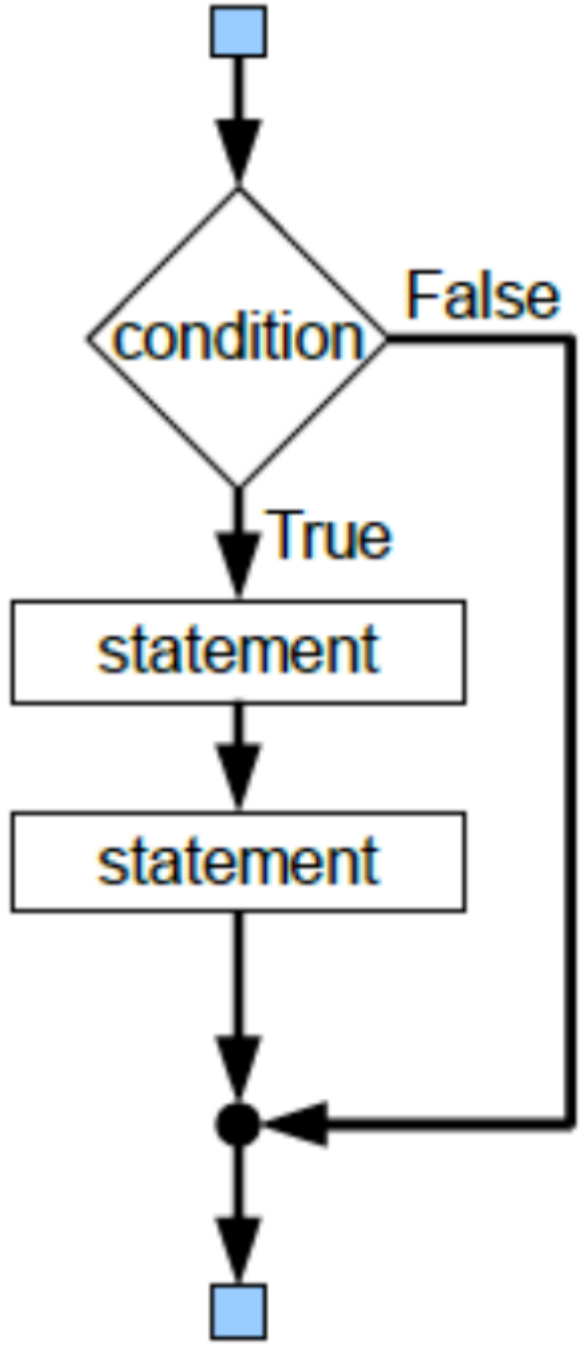
Example:

A=2

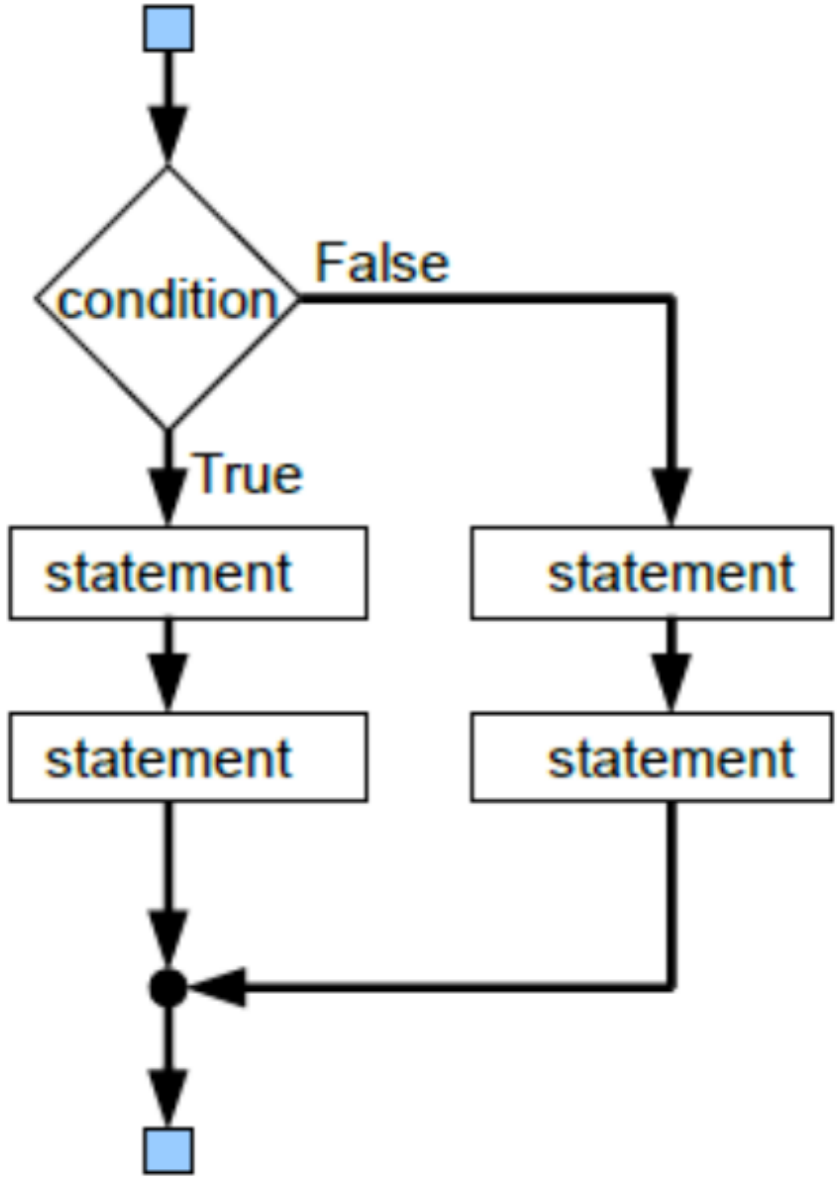
if (A<0) % If A is negative

A = -A; % Make A Positive

end



```
if expression,
        statements
else
        statements
end
```



Example:

```
A=2;  B=3;
if (A<B)
        disp('A is less than B')
else
        disp('A is not less than B')
end
```

Example: Let's say for example that we want to write a function that returns the equivalent grade for an input score according to the following table:

Input Score	Equivalent Grade
96-100	4.0
87-95	3.0
78-86	2.0
70-77	1.0
0-69	0.5

Solution:

```
1   function [grade]=func_2_3(score)
2   if score<70,
3       grade=0.5;
4   elseif score<78,
5       grade=1.0;
6   elseif score<87,
7       grade=2.0;
8   elseif score<96,
9       grade=3.0;
10  else
11      grade=4.0;
12  end
```

```
>> g=func_2_3(85)
g =
     2
```

Polynomials

Examples of polynomials are:

$f(x) = 5x^5 + 6x^2 + 7x + 3$	polynomial of degree 5.
$f(x) = 2x^2 - 4x + 10$	polynomial of degree 2.
$f(x) = 11x - 5$	polynomial of degree 1.

In MATLAB, polynomials are represented by a row vector in which the elements are the coefficients a_n, a_{n-1}, …., a_1, a_0. The first element is the coefficient of the *x* with the highest power. The vector has to include all the coefficients, including the ones that are equal to 0. For example:

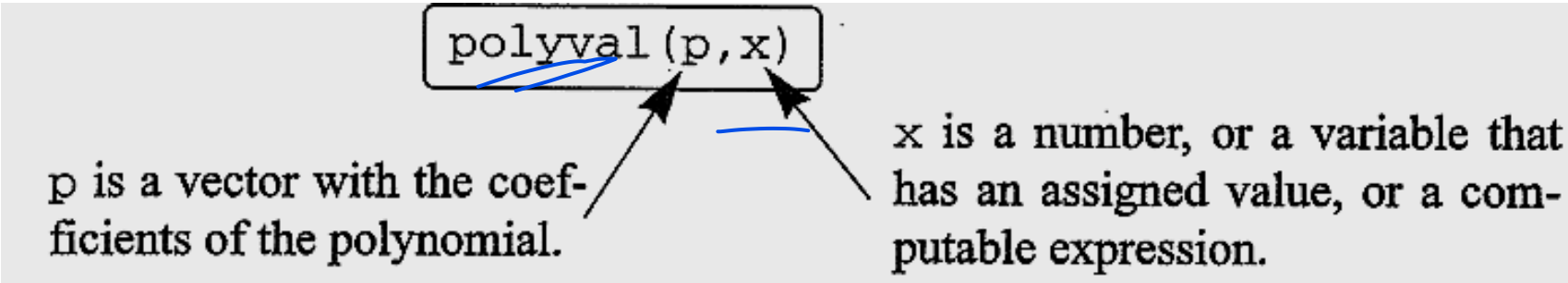
Polynomials

Polynomial	**MATLAB representation**
$8x + 5$	p = [8 5]
$2x^2 - 4x + 10$	d = [2 –4 10]
$6x^2 - 150$, MATLAB form: $6x^2 + 0x - 150$	h = [6 0 –150]
$5x^5 + 6x^2 - 7x$, MATLAB form: $5x^5 + 0x^4 + 0x^3 + 6x^2 - 7x + 0$	c = [5 0 0 6 –7 0]

Polynomials

Value of a Polynomial

The value of a polynomial at a point *x* can be calculated with the function ***polyval*** that has the form:



Polynomials

Calculating polynomials with MATLAB

For the polynomial: $f(x) = x^5 - 12.1x^4 + 40.59x^3 - 17.015x^2 - 71.95x + 35.88$

a) Calculate $f(9)$.

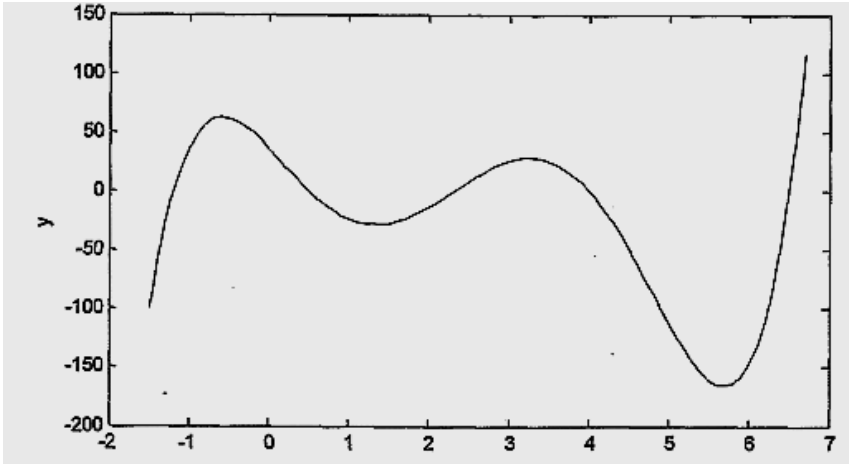
b) Plot the polynomial for $-1.5 \le x \le 6.7$.

(a)

```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.881;
>> polyval(p,9)
ans =
7.2611e+003
```

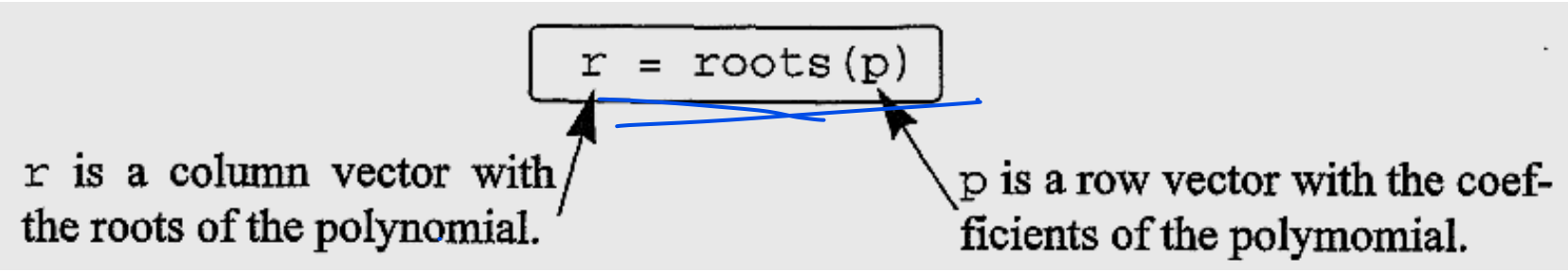
(b)

```
>> x = -1.5:0.1:6.7;
>> y = polyval(p,x);
>> plot(x,y)
```



Roots of a Polynomial

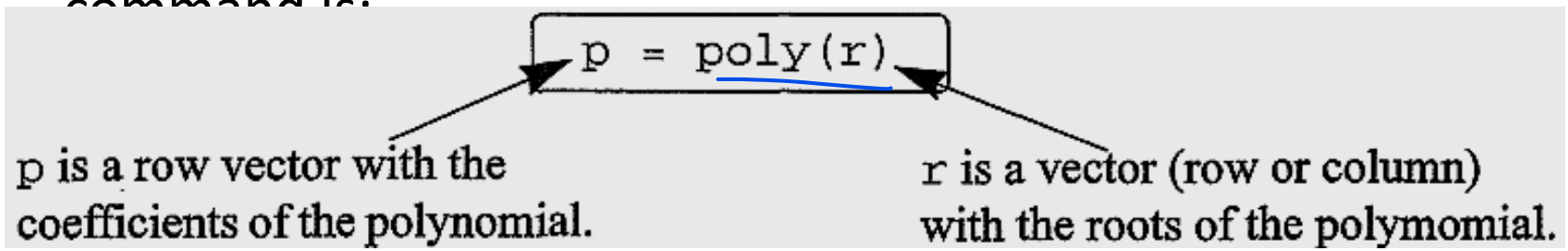
Roots of a polynomial are the values of the argument for which the value of the polynomial is equal to zero.




```
>> p = [1 -12.1 40.59 -17.015 -71.95 35.881];
>> r= roots(p)
r=
6.5000
4.0000
2.3000
-1.2000
0.5000
```

Polynomial of Roots

When the roots of a polynomial are known, the ***ploy*** command can be used for determining the coefficients of the polynomial. The form of the ploy command is:




```
>> r= [6.5 4 2.3 -1.2 0.5]
>> p=poly(r)
p=
  1.0000
  -12.1000
  40.5900
  -17.0150
  -71.9500
  35.8800
```

Derivatives of Polynomials

The built-in function ***polyder*** can be used to calculate the derivative of a single polynomial as

`k = polyder(p)`	**Derivative of a single polynomial. `p` is a vector with the coefficients of the polynomial. `k` is a vector with the coefficients of the polynomial that is the derivative.**

```
>> f1 = [3  -2  4]; f2 = [1  0  5];
>> k = polyder(f1)
k =
     6    -2
```

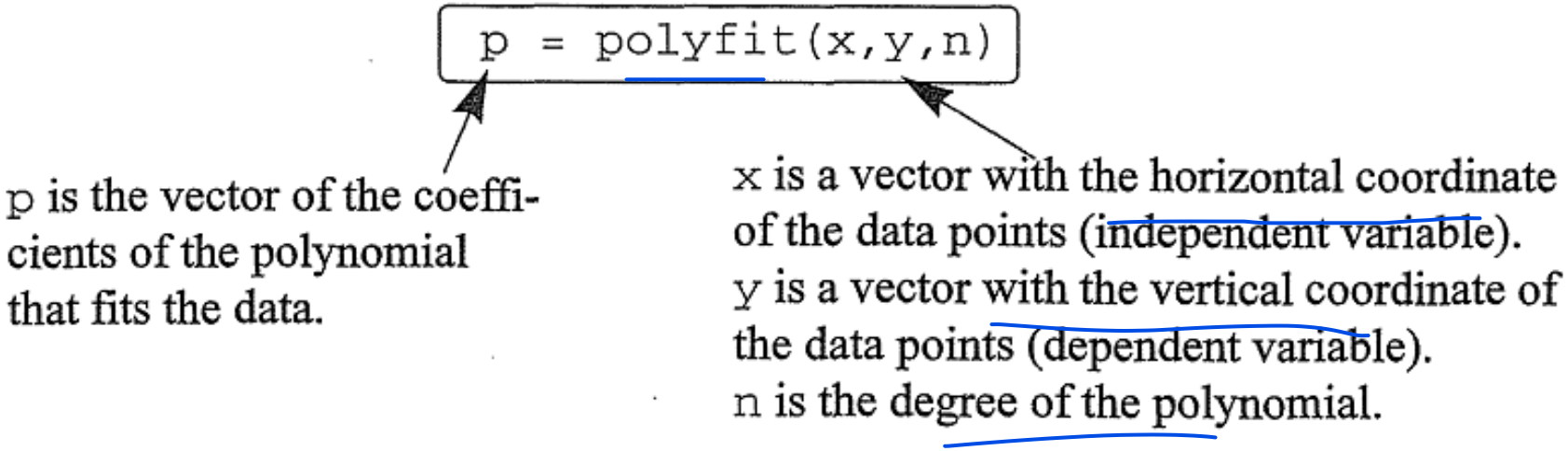
Creating the vectors coefficients of f_1 and f_2.

The derivative of f_1 is: $6x - 2$.

Curve fitting

- Curve fitting, also called regression analysis, is a process of fitting a function to a set of data points.
- The function can then be used as a mathematical model of the data.
- Since there are many types of functions (linear, polynomial, power, exponential, etc.) curve fitting can be a complicated process.

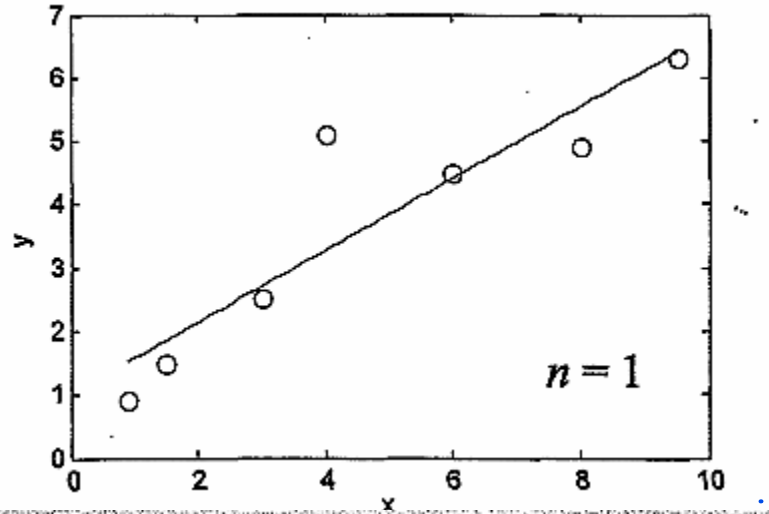
Curve Fitting with Polynomials, the *po1yfit* Function

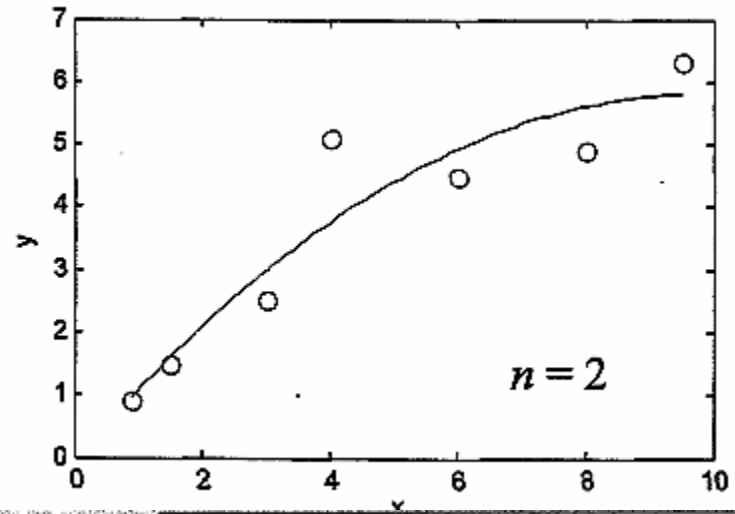


- Make polynomials of different degrees fit for the set of seven points given by:

(0.9, 0.9), (1.5, 1.5), (3, 2.5), (4, 5.I), (6,4.5), (8,4.9), and (9.5, 6.3)

Curve Fitting with Polynomials, the *po1yfit* Function






```
x = [0.9  1.5  3  4  6  8  9.5];
y = [0.9  1.5  2.5  5.1  4.5  4.9  6.3];
p = polyfit(x,y,3)
xp = 0.9:0.1:9.5;
yp = polyval(p,xp);
plot(x,y,'o',xp,yp)
xlabel('x'); ylabel('y')
```

Create vectors `x` and `y` with the coordinates of the data points.

Create a vector `p` using the `polyfit` function.

Create a vector `xp` to be used for plotting the polynomial.

Create a vector `yp` with values of the polynomial at each `xp`.

A plot of the 7 points and the polynomial.

Curve Fitting with Functions Other than Polynomials

Function		**`polyfit` function form**
power	$y = bx^m$	`p=polyfit(log(x),log(y),1)`
exponential	$y = be^{mx}$ or	`p=polyfit(x,log(y),1)` or
	$y = b10^{mx}$	`p=polyfit(x,log10(y),1)`
logarithmic	$y = m\ln(x) + b$ or	`p=polyfit(log(x),y,1)` or
	$y = m\log(x) + b$	`p=polyfit(log10(x),y,1)`
reciprocal	$y = \frac{1}{mx + b}$	`p=polyfit(x,1./y,1)`

Fitting an equation to data points

The following data points are given. Determine a function $w = f(t)$ (t is the independent variable, w is the dependent variable) with a form discussed in this section that best fits the data.

t	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
w	6.00	4.83	3.70	3.15	2.41	1.83	1.49	1.21	0.96	0.73	0.64

```
t = 0:0.5:5;
w = [6 4.83 3.7 3.15 2.41 1.83 1.49 1.21 0.96 0.73 0.64];
p = polyfit(t,log(w),1);
m = p(1)
b = exp(p(2))
tm = 0:0.1:5;
wm = b*exp(m*tm);
plot(t,w,'o',tm,wm)
```

Create vectors `t` and `w` with the coordinates of the data points.

Use the `polyfit` function with `t` and `log(w)`.

Determine the coefficient b.

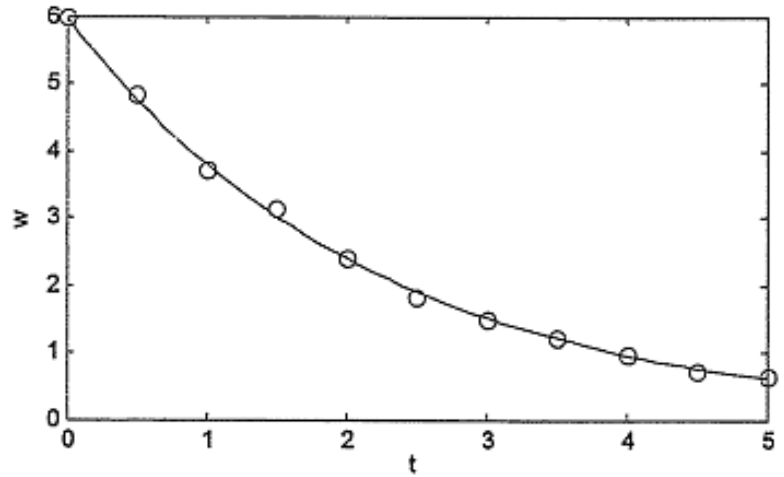
Create a vector `tm` to be used for plotting the polynomial.

Calculate the function value at each element of `tm`.

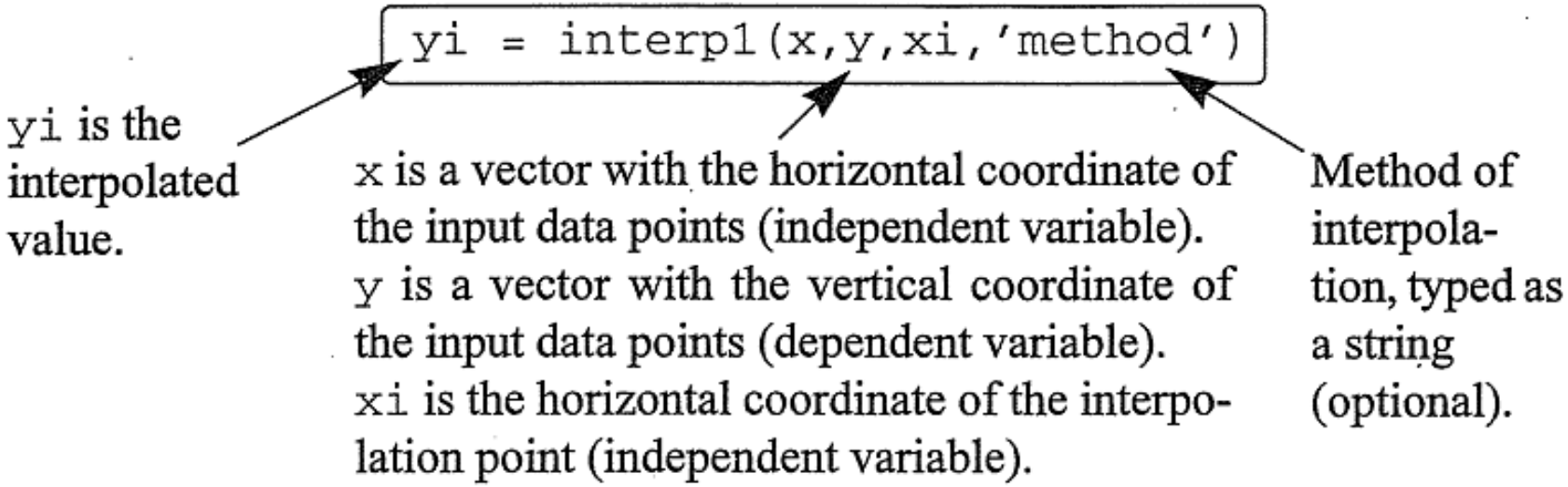
Plot the data points and the function.

```
m =
   -0.4580
b =
    5.9889
```

The plot, generated by the program, that shows the data points and the function is (axes labels were added with the Plot Editor):



Interpolation in MATLAB



'nearest' - nearest neighbor interpolation

'linear' - linear interpolation

'spline' - piecewise cubic spline interpolation (SPLINE)

'pchip' - piecewise cubic Hemite interpolation

'cubic' - same as 'pchip'

'v5cubic' - the cubic interpolation, which does not extrapolate and uses 'spline' if X is not equally spaced.

- The following data points, which are points of the function $f(x) = 1.5^x \cos(2x)$, are given.
- Use linear, spline, and pchip interpolation methods to calculate the value of y between the points.
- Make a figure for each of the interpolation methods.
- In the figure show the points, a plot of the function, and a curve that corresponds to the interpolation method.

x	0	1	2	3	4	5
y	1.0	–0.6242	–1.4707	3.2406	–0.7366	–6.3717

```
x = 0:1.0:5;
y = [1.0  -0.6242  -1.4707  3.2406  -0.7366  -6.3717];
xi = 0:0.1:5;
yilin = interp1(x,y,xi,'linear');
yispl = interp1(x,y,xi,'spline');
yipch = interp1(x,y,xi,'pchip');
yfun = 1.5.^xi.*cos(2*xi);
subplot(1,3,1)
plot(x,y,'o',xi,yfun,xi,yilin,'--');
subplot(1,3,2)
plot(x,y,'o',xi,yfun,xi,yispl,'--');
subplot(1,3,3)
plot(x,y,'o',xi,yfun,xi,yipch,'--');
```

Create vectors `x` and `y` with coordinates of the data points.

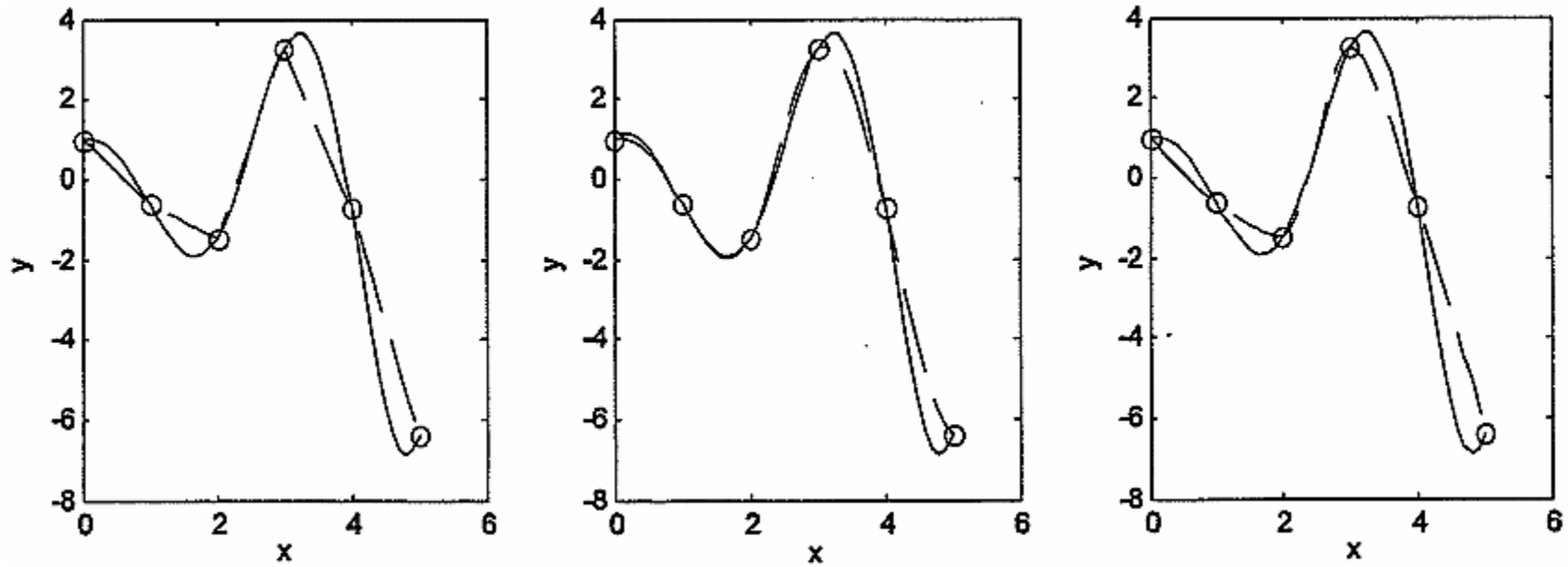
Create vector `xi` with points for interpolation.

Calculate *y* points from linear interpolation.

Calculate *y* points from spline interpolation.

Calculate *y* points from pchip interpolation.

Calculate *y* points from the function.


4
2
0
-2
-4
-6
-8
y
0
2
4
6
x

Solution to Differential Equations

dsolve('equation', 'condition'): Symbolically solves the ordinary differential equation specified by **'equation'**. The optional argument **'condition'** specifies a boundary or initial condition.

The symbolic equation uses the letter **D** to denote differentiation with respect to the independent variable. **D** followed by a digit denotes repeated differentiation. Thus, **Dy** represents dy/dx, and **D2y** represents d^2y/dx^2. For example, given the ordinary second order differential equation;

$$\frac{d^2x}{dt^2} + 5\frac{dx}{dt} + 3x = 7$$

with the initial conditions $x(0) = 0$ and $\dot{x}(0) = 1$.

The MATLAB statement that determines the symbolic solution for the above differential equation is the following:

```
x = dsolve('D2x = -5*Dx - 3*x + 7', 'x(0) = 0', 'Dx(0) =1')
```