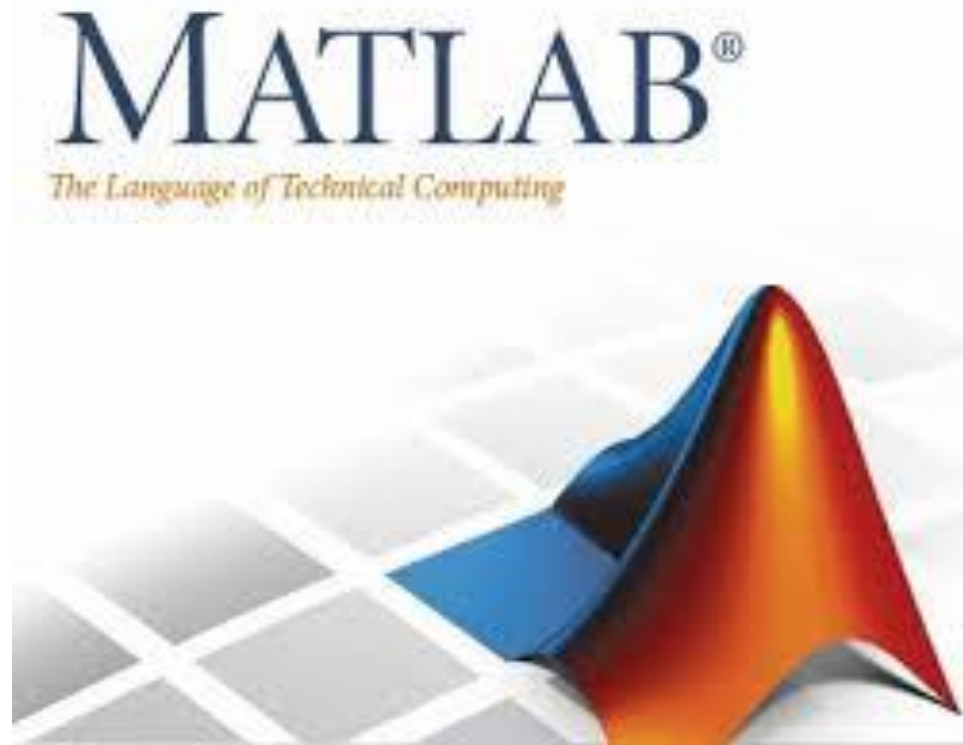




## Computer Aided Design (CAD)

### Lecture 3

- Arrays in Matlab
- Functions in Matlab



**Reference:**

**Matlab by Example: Programming Basics, Munther Gdeisat**



# **Chapter 4: Arrays in Matlab**

# Creating Arrays Manually

```
>> X = [1,2,4;7,3,5];
```

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

```
>> whos X
```

Name	Size	Bytes	Class	Attributes
X	2×3	48	double	

```
>> Y = [1,8;3,6;6,4];
```

$$\mathbf{Y} = \begin{bmatrix} 1 & 8 \\ 3 & 6 \\ 6 & 4 \end{bmatrix}$$

## Creating Arrays Manually: Column-By-Column

```
>> X = [[1;7],[2;3],[4;5]];
```

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

[1;7] creates the first column in the array X.

[2;3] creates the second column.

[4;5] creates the third column.

The three columns are combined together (concatenated) using commas.

# Creating Arrays using the repmat function

The repmat function is an abbreviation of “repeat matrix,”

## Syntax

`B = repmat(A,M,N)`

- This function creates a large matrix B consisting of an MxN copies of A.
- This function has the following three arguments:
  1. A is the source array.
  2. M is the number of times A is repeated in the vertical direction.
  3. N is the number of times A is repeated in the horizontal direction.

Suppose that you have the following array A:

A = 

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6



```
>> a = [1,2,3,4,5,6];  
>> A = repmat(a,5,1)
```

# Creating Arrays using the repmat function

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}$$



```
>> b = [1;2;3;4;5];  
>> B = repmat(b,1,5)
```

$$\mathbf{C} = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 \end{bmatrix}$$



```
>> C1 = [1,2;3,4];  
>> C = repmat(C1,2,2);
```

# Transpose an Array

```
>> X = [1,2,4;7,3,5]
```

X =

```
1 2 4
7 3 5
```

```
>> XT = X'
```

XT =

```
1 7
2 3
4 5
```

# Changing Array Dimensions Using the reshape Function

- This function changes the dimensions of the array X to the new size of MxN.

## Syntax

`B = reshape(X,M,N)`

- The elements are taken from the source array X in a **column-by-column fashion**.
1. X is the source array.
  2. M is the number of rows in the destination array B.
  3. N is the number of columns in the destination array B.

### Example 4

Using Matlab, change the dimensions of the  $2 \times 3$  array

$$x = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

to  $3 \times 2$ .



### Answer

```
>> X = [1,2,4;7,3,5]
>> B = reshape(X,3,2)
```

Matlab responds with

```
B =
     1     3
     7     4
     2     5
```



# Changing Array Dimensions Using the reshape Function

## Example 5

Create the array **B** using the reshape function.

$$\mathbf{B} = \begin{bmatrix} 1 & 6 & 11 & 16 & 21 & 26 \\ 2 & 7 & 12 & 17 & 22 & 27 \\ 3 & 8 & 13 & 18 & 23 & 28 \\ 4 & 9 & 14 & 19 & 24 & 29 \\ 5 & 10 & 15 & 20 & 25 & 30 \end{bmatrix}$$

## Answer

```
>> b = 1:1:30;  
>> B = reshape(b,5,6)
```

## Example 6

Create the following array **S** using the reshape function.

$$\mathbf{S} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}$$

## Answer

```
>> s = 1:1:25;  
>> S = reshape(s,5,5);  
>> S = S';
```

Remember: Case sensitivity applies, so `s` and `S` are different variables!

# Finding the Size of an Array

- Number of rows and columns of an array can be calculated using “size” function.

## Syntax

```
>> [m,n]=size(X)
```

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

- m is number of rows, n is number of columns

To find the number of rows in X,

```
>> m = size(X,1)
```

```
m =  
    2
```

To find the number of columns in X,

```
>> n = size(X,2)
```

```
n =  
    3
```

- To find the total number of elements in the array X:

```
>> r = numel(X)
```



```
r =  
    6
```

# Converting an Array to a Column Vector

- You can convert an array to a column vector using the colon (:) operator.

For example, to convert the array X to a column vector, type:

```
>> X = [1,2,4;7,3,5]
```

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix}$$

```
>> x = X(:)
```

```
x =
```

```
1  
7  
2  
3  
4  
5
```

Note that the elements have been extracted from the array X, in a **column-by-column fashion**.

# Arrays Concatenation

- Arrays can be concatenated (combined) together to produce larger arrays.

## Example 8

Concatenate the two arrays

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix} \quad \text{and} \quad \mathbf{Z} = \begin{bmatrix} 1 & 2 & 5 \\ 8 & 3 & 4 \\ 9 & 6 & 7 \end{bmatrix}$$

to produce the array

$$\mathbf{F} = \begin{bmatrix} 1 & 2 & 5 \\ 8 & 3 & 4 \\ 9 & 6 & 7 \\ 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{X} \end{bmatrix}$$

```
>> Z=[1,2,5;8,3,4;9,6,7];  
>> X=[1,2,4;7,3,5];  
>> F=[Z;X]
```

Note that here we have used the semicolon (;) to combine Z and X arrays in the **vertical** direction.

# Arrays Concatenation

## Example 9

Concatenate the arrays

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 7 & 3 & 5 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 3 & 5 \\ 9 & 7 \end{bmatrix}$$

to produce the array

$$S = \begin{bmatrix} 1 & 2 & 4 & 3 & 5 \\ 7 & 3 & 5 & 9 & 7 \end{bmatrix} = [X \ R]$$

## Answer

```
>> X = [1,2,4;7,3,5];  
>> R = [3,5;9,7];  
>> S = [X,R];
```

Note that here we have used the comma (,) to combine X and R arrays in the **horizontal** direction.

# Accessing Elements in Arrays

## Accessing an Individual Element in an Array Using its Index

- Two methods to access the elements within an array:
  - **Row-and-column** indexing.
  - **Linear** indexing.

### 1- Row-and-Column Indexing Method

An element in the array  $X$  is referred to as  $X_{m,n}$ , where  $m$  refers to the **row number** and  $n$  refers to the **column number**.

$$X = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

$$\gg f = X(2, 3) \quad \longrightarrow \quad f = 7$$

$$\begin{bmatrix} X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} \\ X_{2,1} & X_{2,2} & X_{2,3} & X_{2,4} \\ X_{3,1} & X_{3,2} & X_{3,3} & X_{3,4} \end{bmatrix}$$

To access the last element in the first row of  $X$ ,

$$\gg s = X(1, \text{end}) \quad \longrightarrow \quad s = 12$$

To access the last element in the third column of  $X$ ,

$$\gg t = X(\text{end}, 3) \quad \longrightarrow \quad t = 9$$

# Accessing an Individual Element in an Array Using its Index

## 2- Linear-Indexing Method

$$\mathbf{X} = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

$$\begin{bmatrix} X_1 & X_4 & X_7 & X_{10} \\ X_2 & X_5 & X_8 & X_{11} \\ X_3 & X_6 & X_9 & X_{12} \end{bmatrix}$$

```
>>a=X(1)  
a=  
3
```

```
>>b=X(7)  
b=  
8
```

```
>>c=X(12)  
c=  
10
```

```
>>c=X(end)  
c=  
10
```

## Accessing Rows in an Array

You can use the colon operator (:) to access a **Row** in an array.

To access the first row:

```
>> a = X(1, :)
```



```
a =  
3 4 8 12
```

$$\mathbf{X} = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

To access the last row,

```
>> b = X(end, :)
```



```
b =  
1 6 9 10
```

To access the last two rows

```
>> B = X(end - 1:end, :)
```



```
B =  
2 5 7 11  
1 6 9 10
```

To access the first and the third rows,

```
>> C = X([1,3], :)
```



```
C =  
3 4 8 12  
1 6 9 10
```



## Accessing Columns in an Array

You can use the colon operator (:) to access a **Column** in an array.

To access the first column:

```
>> a = X(:,1)
```



```
a =  
3  
2  
1
```

$$\mathbf{X} = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

To access the last column,

```
>> b = X(:,end)
```



```
b =  
12  
11  
10
```

To access the first and second columns,

```
>> C = X(:,[1,2])
```



```
C =  
3 4  
2 5  
1 6
```

## Accessing a Group of Elements in an Array Using Their Indices

To access the first and second rows of the third column.

```
>> r = X([1,2],3)
```



```
r =  
    8  
    7
```

$$X = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

To access the second, third, and fourth columns of the second row

```
>> e = X(2,[2,3,4])
```



```
e =  
    5    7   11
```

$$X = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

To access the elements that are encircled by the rectangular area below:

```
>> G = X([2,3],[2,3,4])
```



```
G =  
    5    7   11  
    6    9   10
```

$$X = \begin{bmatrix} 3 & 4 & 8 & 12 \\ 2 & 5 & 7 & 11 \\ 1 & 6 & 9 & 10 \end{bmatrix}$$

## Accessing Elements in an Array Using Their Values

$$E = \begin{bmatrix} 7 & 3 & 9 \\ 1 & 0 & 2 \\ 5 & 8 & 4 \end{bmatrix}$$

- To find the indices of the elements whose values is greater than 7,

```
>> [a,b] = find(E > 7);  
>> [a,b]
```

ans =  
3 2  
1 3

This output means that the elements  $E(3,2)$  and  $E(1,3)$  are greater than 7.

- To find the **indices** of the elements in the array E whose value is less than 3,

```
>> [c,d] = find(E < 3);  
>> [c,d]
```

ans =  
2 1  
2 2  
2 3

- To find the **values** of the elements in E that have values that are less than 3,

```
>> f = find(E < 3);  
>> r = E(f)
```

r =  
1  
0  
2

## Accessing Elements in an Array Using Relational and Logical Operators

- To find the elements in  $X$  whose values are greater than 3.

$$\mathbf{X} = \begin{bmatrix} 1 & 2 & -4 \\ 7 & 0 & 5 \end{bmatrix}$$

$$\gg X > 3 \quad \longrightarrow \quad \text{ans} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} -3 & 5 & 1 \\ 9 & 7 & 0 \end{bmatrix}$$

- To access the elements of  $Y$  which have the corresponding positions within the array to locations in the array  $X$  where  $X$  is greater than 3.

$$\gg r = Y(X > 3) \quad \longrightarrow \quad r = \begin{bmatrix} 9 \\ 0 \end{bmatrix}$$

# Plotting Arrays

## 3D Plot an Array with the mesh Function

➤ Let us plot the function  $Z = X^2 - Y^2$  where  $X$  is in the range of  $[-2, 2]$  and  $Y$  is in the range of  $[-3, 3]$ .

```
x = -2:1:2;  
y = -3:1:3;  
[X,Y] = meshgrid(x,y);
```

X =

-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2
-2	-1	0	1	2

Y =

-3	-3	-3	-3	-3
-2	-2	-2	-2	-2
-1	-1	-1	-1	-1
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

To evaluate Z using Matlab, type

```
Z = X.^2 - Y.^2
```

Z =

-5	-8	-9	-8	-5
0	-3	-4	-3	0
3	0	-1	0	3
4	1	0	1	4
3	0	-1	0	3
0	-3	-4	-3	0
-5	-8	-9	-8	-5

# Plotting Arrays

## 3D Plot an Array with the mesh Function

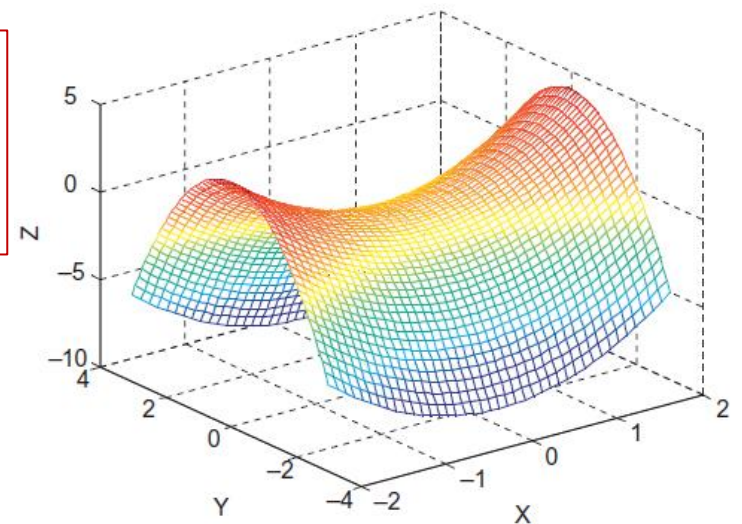
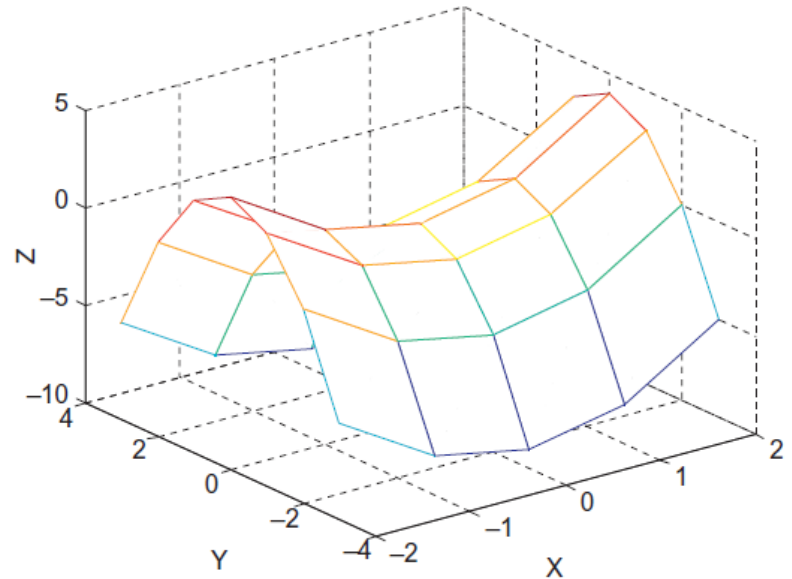
To plot the array  $Z$  versus  $X$  and  $Y$ , type

```
mesh(X,Y,Z)
```

```
x = -2:1:2;  
y = -3:1:3;  
[X,Y] = meshgrid(x,y);  
Z = X.^2 - Y.^2;  
mesh(X,Y,Z)  
xlabel('X')  
ylabel('Y')  
zlabel('Z')
```

- To improve the resolution of the 3D plot, increase the number of points within the  $X$  and  $Y$  arrays

```
x = -2:0.1:2;  
y = -3:0.1:3;  
[X,Y] = meshgrid(x,y);  
Z = X.^2 - Y.^2;  
mesh(X,Y,Z)
```



# Plotting Arrays

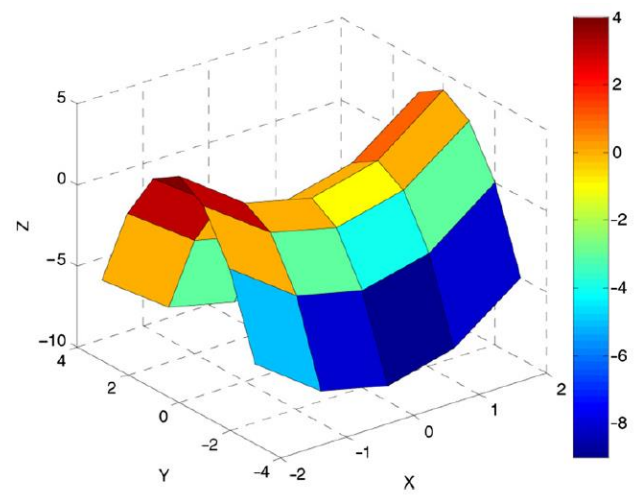
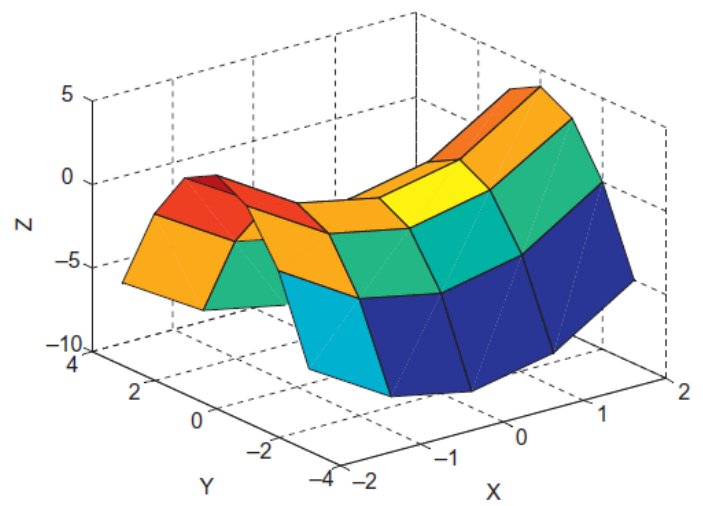
## 3D Plot an Array with the surf Function


➤ The surf function plots an array as a surface

```
x = -2:1:2;  
y = -3:1:3;  
[X,Y] = meshgrid(x,y);  
Z = X.^2-Y.^2;  
surf(X,Y,Z)
```

➤ The relationship between the color in the surface plot and the value of Z can be shown using colorbar command

```
x = -2:1:2;  
y = -3:1:3;  
[X,Y] = meshgrid(x,y);  
Z = X.^2-Y.^2;  
surf(X,Y,Z)  
xlabel('X')  
ylabel('Y')  
zlabel('Z')  
colorbar
```





# **Chapter 5: Functions in Matlab**

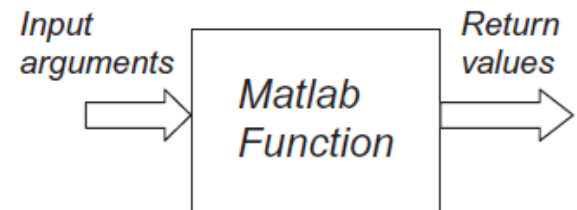


# Functions in Matlab

- A Matlab function is a collection of commands that does a specific task and must have a unique name.
- Functions can **improve** Code **Readability** and **Reusability**.

## Example:

```
function a = add2(b,c)
a = b + c;
end
```



The name of this function is add2.

This function has two input arguments b and c.

The collection of commands in this function is  $a = b + c$ ;

This function returns one value a, which is the sum of b and c.

## Example:

```
function [r,theta] = Cartesian2polar(x,y)
r = sqrt(x^2 + y^2);
theta = atan2(y,x);
end
```

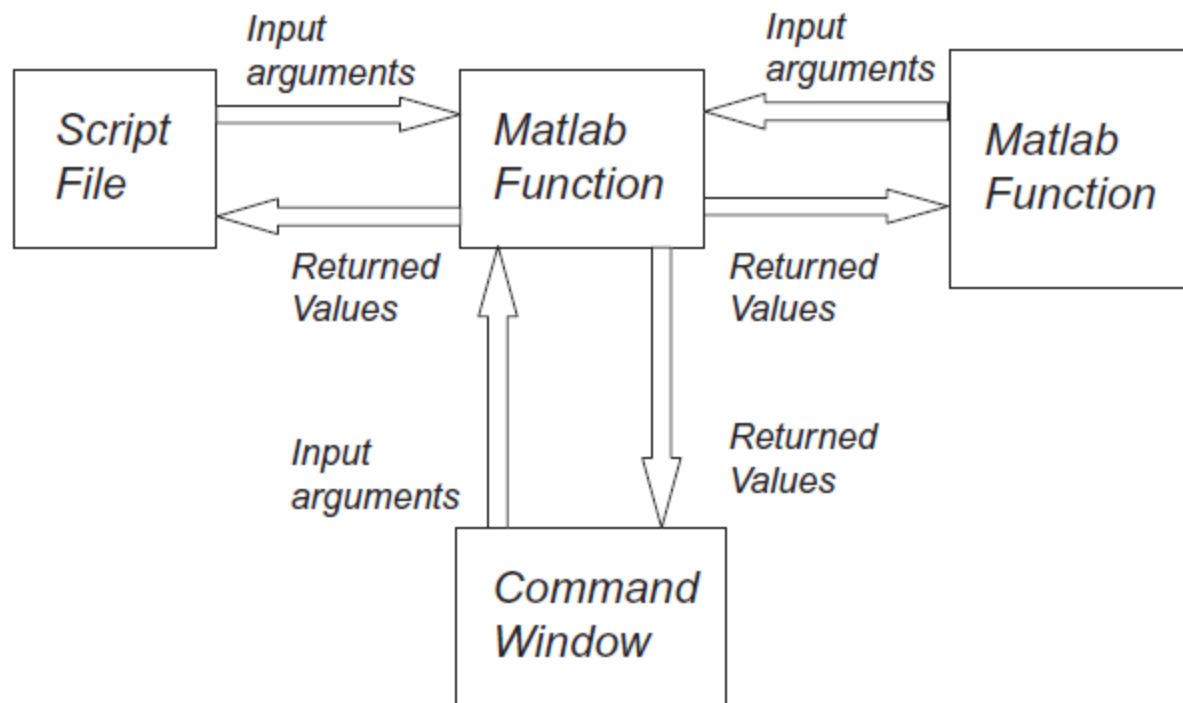
Function name: "Cartesian2polar"

Inputs: x,y

Outputs: r, theta

# Calling a Matlab Function

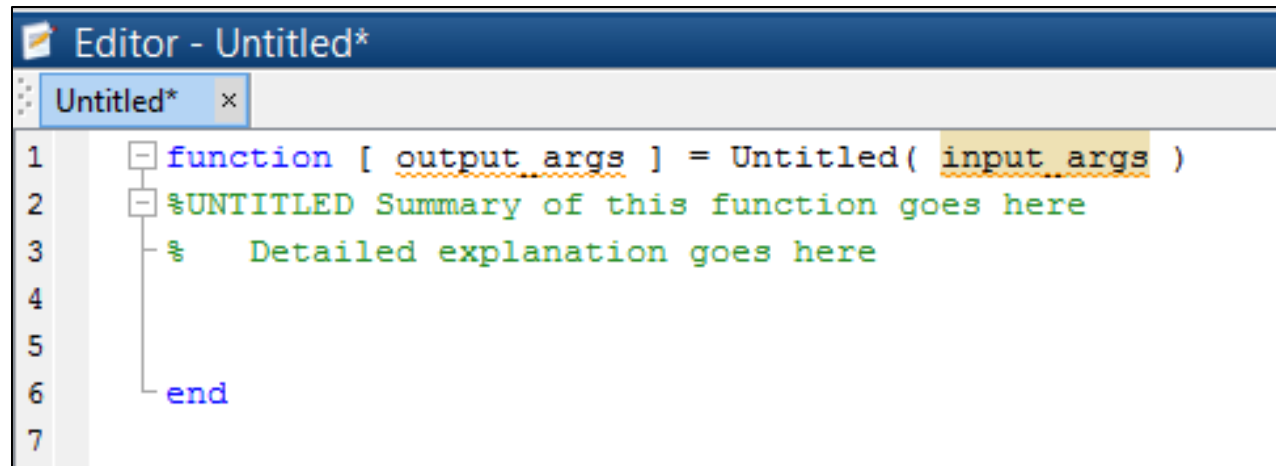
- You can call a Matlab function from:
  - A script file,
  - The Command Window,
  - Another function



# Creating Functions

**Menu**→**File**→**New**→**Function**.

- The Matlab Editor pops up and write your function as :



The screenshot shows the Matlab Editor window titled "Editor - Untitled\*". The editor contains the following code:

```
1 function [ output_args ] = Untitled( input_args )
2 %UNTITLED Summary of this function goes here
3 % Detailed explanation goes here
4
5
6 end
7
```

- Delete everything in the Editor and type the following code in the Editor

```
function z = add2(x, y)
%This function adds the numbers x and y
%and returns the value z which is the result of
%the addition of the two numbers
z = x + y;
end
```

- Save this function as add2.m
- The name of the file **MUST** be exactly the same as the name of the function and must be followed by the .m extension.

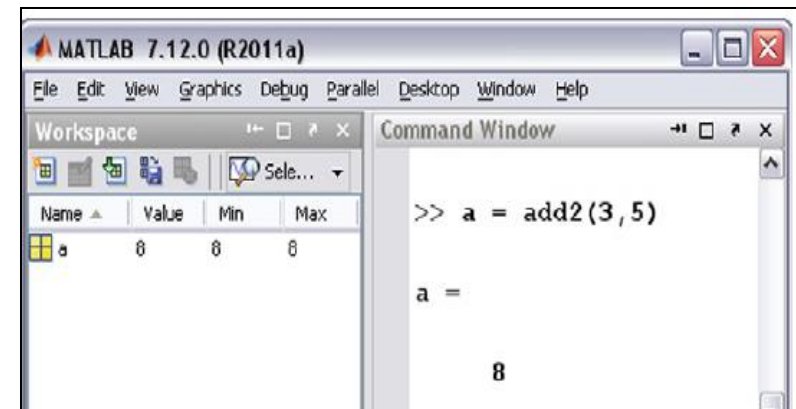
# Calling a Matlab Function

## Calling a Matlab Function from the Command Window

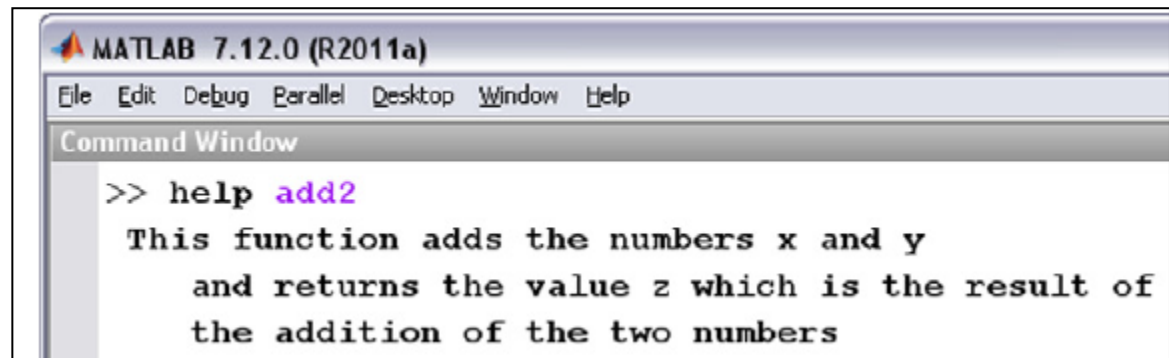
```
>> a = add2(3, 5);
```

- You must give the correct number of input arguments or you will get an error.
- Matlab executes the add2 function, and the returned result is assigned to the variable a.

- **Note that** the function arguments x, y and z do not actually appear in the Workspace window and they do not exist in Matlab memory, as they were only temporary function variables.



```
>> help add2
```



# Calling a Matlab Function

## Calling a Matlab Function from a Script File

```
a = 1;  
b = 2;  
c = add2(a, b)
```

## Calling a Matlab Function from Another Function

```
function d = add3(a, b, c)  
e = add2(a, b);  
d = add2(e, c);  
end
```

To call this function, at the **Command Prompt** type

```
>> z = add3(1, 2, 3)
```

Matlab responds with

```
z =
```

```
6
```

# A Matlab Function Returning Two Values

```
function [addition, subtraction] = add_sub(x,y)
addition = x + y;
subtraction = x - y;
end
```

To call this function:

```
>> [r, s] = add_sub (5,3)
```

The result of calling this function is

```
r =
    8
```

```
s =
    2
```

# Scope of Matlab Variables in a Function

- A variable that is created within a function has a limited scope. This means that this variable can be only accessed or modified by this function.
- This variable is called a **local variable**.

## Example 1

Create a function that raises its input argument to the power  $r = 2$ .

Answer:

```
function c = pow(a)
r = 2;
c = a.^r;
end
```

Call this function

```
>> f = pow(3)
```

```
f =
    9
```

- The variables `a`, `r`, and `c` are local variables to the function `pow`. To check this for the variable `r`, type at the Command Prompt

```
>> r
```

Matlab responds with

```
??? Undefined function or variable 'r'.
```

# Scope of Matlab Variables in a Function

- A variable created in the Command Window cannot be accessed by a function.

## Example:

```
function c = pow(a)
c = a.^r;
end
```

Call this function from the **Command Window** as follows:

```
>> r = 2;
>> f = pow(2)
```

Matlab responds with

```
???Undefined function or variable 'r'.
```

```
Error in ==> pow at 2
c = a.^r;
```

Even though we have created the variable `r` in the Command Window, the `pow` function cannot access this variable.

- Similarly, a variable created in a script file cannot be accessed by a function.





**Thanks for attention**