



TECHNOLOGY  
SUPPORT

# *MATLAB*

## SHORTCOURSE HANDOUT



TEXAS TECH UNIVERSITY  
Technology Support

Texas Tech University | Heide Mansouri

## Table of Contents

Introduction .....	4
Course Objectives .....	4
Texas Tech University MATLAB Campus License .....	5
The MATLAB System .....	5
Overview of the MATLAB Desktop .....	6
Command Window .....	7
Workspace Window .....	8
Ways to Get Help, using the Command-line .....	9
To Perform Simple Computations .....	10
Variable Names .....	12
Some MATLAB function descriptions .....	13
Commands for Managing variables and the workspace .....	13
To Check for MATLAB Keywords: .....	14
Built-In Functions .....	14
Making Mistakes .....	15
Numeric Display Format .....	15
Preferences .....	16
Precedence of Arithmetic Operations .....	17
Some MATLAB Special Characters .....	17
To Suppress Output with Semicolon (;) .....	17
Creation of Matrices and Vectors .....	18
Overwriting Variables .....	20
Using the Colon Notation (:). .....	20
Workspace Window Setting .....	21
Opening the Variable Editor .....	22
Accessing Single Elements in a Matrix .....	25
Creating Matrices, using Functions .....	27
Building Elementary Matrices and Arrays (all lowercases) .....	28
Adding and Subtracting Matrices .....	29
Basic Arithmetic Operations in MATLAB .....	29

Element -by-Element (Element-wise) Operations..... 30

Matrix Transpose ..... 31

Vector Products and Transpose ..... 32

Multiplying Matrices..... 32

Roots of a Polynomial..... 33

Solving the System of Linear Equation  $AX=b$ ..... 34

Creating Symbolic Variables and Expressions ..... 35

The syms command: ..... 36

Using the solve function ..... 37

Integration ..... 38

Symbolic Integration, using *int* Function..... 40

To compute integrals using the int function..... 41

Definite Integrals Example: ..... 42

Graphics Window ..... 42

Specifying Graph’s Line, Mark, and Color Styles..... 45

Cylinder Function..... 47

To save your graph: ..... 47

Cones are special cylinders: ..... 47

Creating Symbolic Variables and Functions (Calculus Example)..... 50

Importing Excel Spreadsheet Data, using the Import Tool..... 52

Some Important Commands ..... 55

The M-Files..... 55

Saving and loading Files..... 56

The Difference between \*.m and \*.mat files in MATLAB ..... 57

Online Resources..... 57

# MATLAB ShortCourse

---

Copyright 2012-2018 Heide Mansouri, Texas Tech University. ALL RIGHTS RESERVED. Members of Texas Tech University or Texas Tech Health Sciences Center may print and use this material for their personal use only. No part of this material may be reproduced in any form without written permission from Heide Mansouri, the [author](#).

**Credit:** This document is adapted mostly from MATLAB User's Guide Documentation.

## Introduction

MATLAB, short for MATrix LABoratory, is an interactive matrix-based software package; designed for engineering and scientific applications. The MATLAB Desktop is a set of tools for managing files, variables, and applications associated with MATLAB.

This ShortCourse is designed to acquaint you with the basics of **MATLAB 9.4, Release 2018a** under **Windows**. The *mathematical background needed* for this ShortCourse, is **college algebra**. This ShortCourse is meant to serve as a quick way to learn MATLAB. For more detailed information, please consult the official MATLAB documentation.

## Course Objectives

After completing this ShortCourse, you should be able to

- Perform Basic MATLAB Operations;
- Perform simple computations;
- Work with variables and functions;
- Work with **Matrices** and **Vectors** ( in MATLAB, all variables are matrices);
- Create **matrices** and **vectors**;
- Perform some **basic linear algebra**;
- Work with array operators;
- Use the **Symbolic Integration** to calculate **integrals**;
- Create **Plots**;
- Solving the System of Linear **Equation  $AX=b$** ;
- Compute integrals using the **int** function; and
- Import **Excel Spreadsheet Data**, using the **Import Tool**.

### Texas Tech University MATLAB Campus License

- The latest version of MATLAB software and its top **50 toolboxes** is available at no cost to TTU students, faculty, and staff members for download @ <http://www.depts.ttu.edu/itts/software/matlab>.
- MATLAB's semiannual cycle for releasing updates are **March**, and **September** of each year.
- If you have any questions or need assistance, please contact IT Help Central at 742-HELP (4357) or [ithelpcentral@ttu.edu](mailto:ithelpcentral@ttu.edu).
  - Support is available for TTU customers from the MathWorks <http://www.mathworks.com/support>.

### The MATLAB System

The MATLAB system consists of five main parts:

## Texas Tech University MATLAB License

Texas Tech University has a Total Academic Headcount License for MATLAB, Simulink and several add-on products. These products are available campus wide for use by faculty, academic researchers and students. Contact Technology Support for details.

- |   |   |   |
|---|---|---|
| <ul style="list-style-type: none"><li>• <b>MATLAB</b></li><li>• <b>Simulink</b></li><li>• Bioinformatics Toolbox</li><li>• Communications System Toolbox</li><li>• Computer Vision System Toolbox</li><li>• Control System Toolbox</li><li>• Curve Fitting Toolbox</li><li>• Data Acquisition Toolbox</li><li>• Database Toolbox</li><li>• DSP System Toolbox</li><li>• Econometrics Toolbox</li><li>• Embedded Coder</li><li>• Financial Toolbox</li><li>• Fixed-Point Toolbox</li><li>• Fuzzy Logic Toolbox</li><li>• Global Optimization Toolbox</li><li>• Image Acquisition Toolbox</li></ul> | <ul style="list-style-type: none"><li>• Image Processing Toolbox</li><li>• Instrument Control Toolbox</li><li>• Mapping Toolbox</li><li>• MATLAB Coder</li><li>• MATLAB Compiler</li><li>• Model Predictive Control Toolbox</li><li>• Neural Network Toolbox</li><li>• Optimization Toolbox</li><li>• Parallel Computing Toolbox</li><li>• Partial Differential Equation Toolbox</li><li>• RF Toolbox</li><li>• Robust Control Toolbox</li><li>• Signal Processing Toolbox</li><li>• SimBiology</li><li>• SimDriveline</li><li>• SimElectronics</li></ul> | <ul style="list-style-type: none"><li>• SimMechanics</li><li>• SimPowerSystems</li><li>• SimRF</li><li>• Simscape</li><li>• Simulink 3D Animation</li><li>• Simulink Coder</li><li>• Simulink Control Design</li><li>• Simulink Design Optimization</li><li>• Simulink Fixed Point</li><li>• Simulink Verification and Validation</li><li>• Spreadsheet Link EX</li><li>• Stateflow</li><li>• Statistics Toolbox</li><li>• Symbolic Math Toolbox</li><li>• System Identification Toolbox</li><li>• Wavelet Toolbox</li><li>• xPC Target</li></ul> |
|---|---|---|

1. **MATLAB Language** - is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.
2. **MATLAB Desktop Tools** - is the set of tools that help you work and manage the variables in your workspace and importing/exporting data. Desktop Tools

- include the **MATLAB desktop** and **Command Window**, an **editor** and **debugger**, a **code analyzer**, and **browsers** for viewing the **Workspace**.
3. **MATLAB Graphics** - includes high-level commands for **two-dimensional** and **three-dimensional** data visualization, image processing, animation, and presentation graphics.
  4. **MATLAB Mathematical Function Library** -is collection of computational algorithms ranging from elementary functions like **sum**, **sine**, **cosine**, and complex arithmetic, to more sophisticated functions like **matrix inverse**, and more.
  5. **MATLAB Application Program Interface (API)** - is a library that allows you to write C/C++ and FORTRAN programs that interact with MATLAB.

### Starting MATLAB

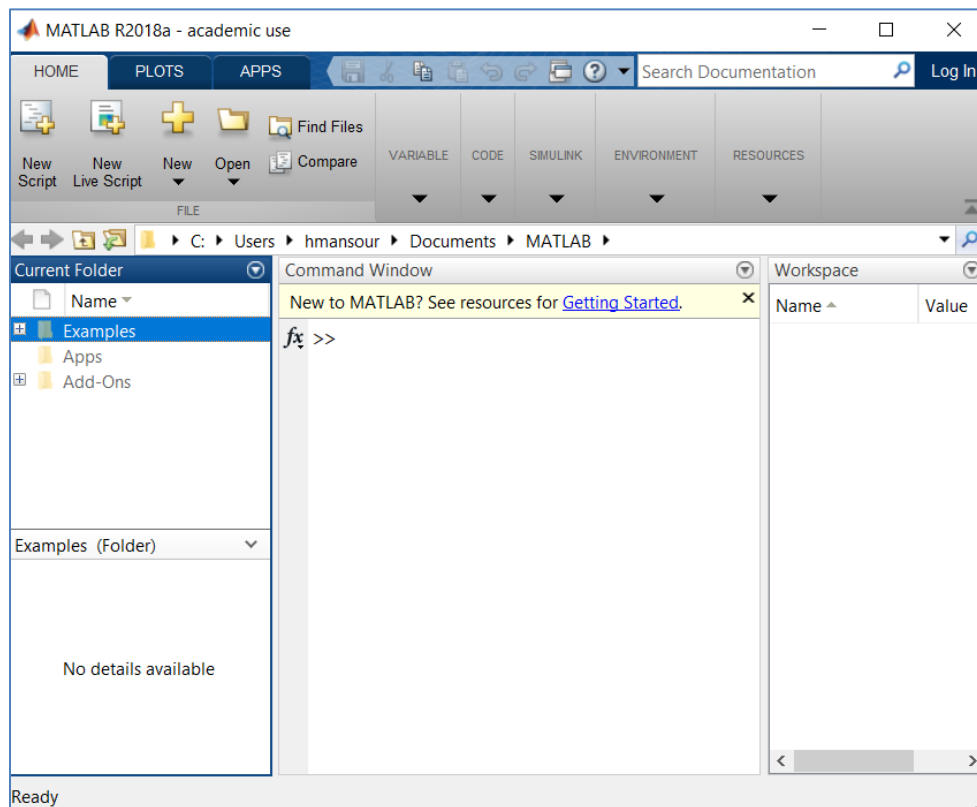
- **Start -> ALL Programs -> MATLAB -> MATLAB R2018a**
- MATLAB displays the **Command Window** (in the center) that is used to enter commands and display text-only results.
- The **Current Folder Pane** is on the left.
- The **Graphics** window and **Editing** window will automatically open, when needed.
- The **MATLAB Workspace** consists of the set of variables built up and stored in memory during a MATLAB session. The **Workspace** browser enables you to view, change, and plot **MATLAB workspace values**.
- The **Command History Window** displays a log of statements you ran in the current and previous MATLAB sessions. All entries remain until you delete them.

### Overview of the MATLAB Desktop

- The following components appear at the top of the MATLAB desktop:
  - **Toolstrip** contains the **HOME**, **PLOTS** and **APPS** tabs. Additional tabs, such as **EDITOR**, **PUBLISH** and **VARIABLE**, appear in the Toolstrip as needed to support your workflow.
  - **Quick access toolbar** displays frequently used options such as **cut**, **copy**, **paste** and **help**.
  - **Current folder** toolbar enables you to specify the current working directory.

## MATLAB ShortCourse Handout

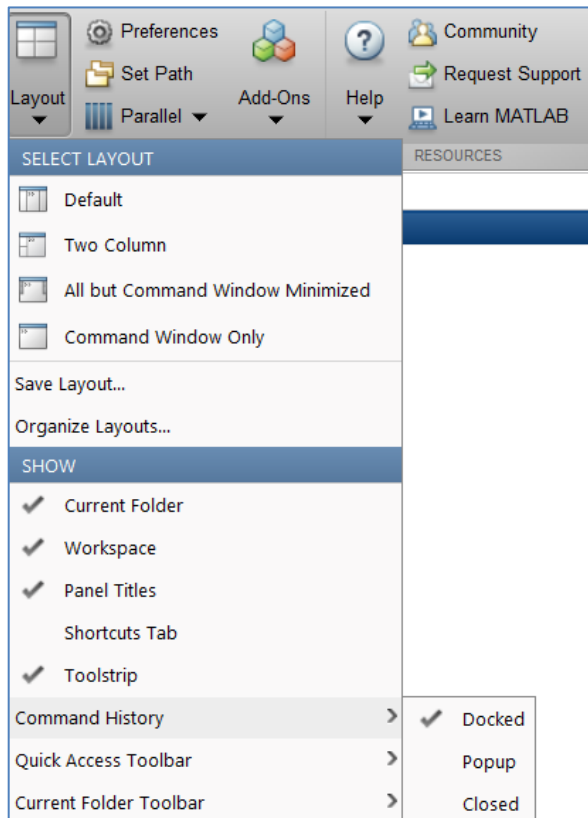
- **Search Documentation** box allows you to search the documentation.
- **The desktop** (in its default layout) displays these windows:
  - **Current Folder** enables you to access the contents of the current working directory. Use the options in the current folder toolbar to change the current working folder.
  - **Command Window** lets you execute commands at the command line, indicated by the prompt (`>>`).
  - **Workspace** allows you to explore data that you create or import from files. You can view the contents of the workspace using ***whos*** command.
  - **Command History** allows you to view or rerun commands that you entered at the command line.



### Command Window

- Use the **Command Window** to enter data, run MATLAB code, and display results.
- Enter statements at the prompt `>>`, which is also known as the **Command-line**.

- You can perform calculations in the Command Window similar to the way you perform calculations on a scientific calculator.
- To reset the MATLAB Desktop to default layout, in the **HOME** tab, choose **Layout**, and then select **Default**.



### Workspace Window

- Workspace is a graphical user interface that allows you to view and manage the contents of the workspace in MATLAB.
- To edit variables, double-click the variable in the Workspace browser. The variable displays in the **Variables Editor**, where you can view the full contents and make changes.
- You add variables to the workspace by using functions, running MATLAB code, and loading saved workspaces.
- From the Workspace browser, you can select variables to **view**, **modify**, or **plot**.
- Workspace variables do not persist after you exit MATLAB. Therefore, use your data for later use with the save command,

*save myfile.mat*

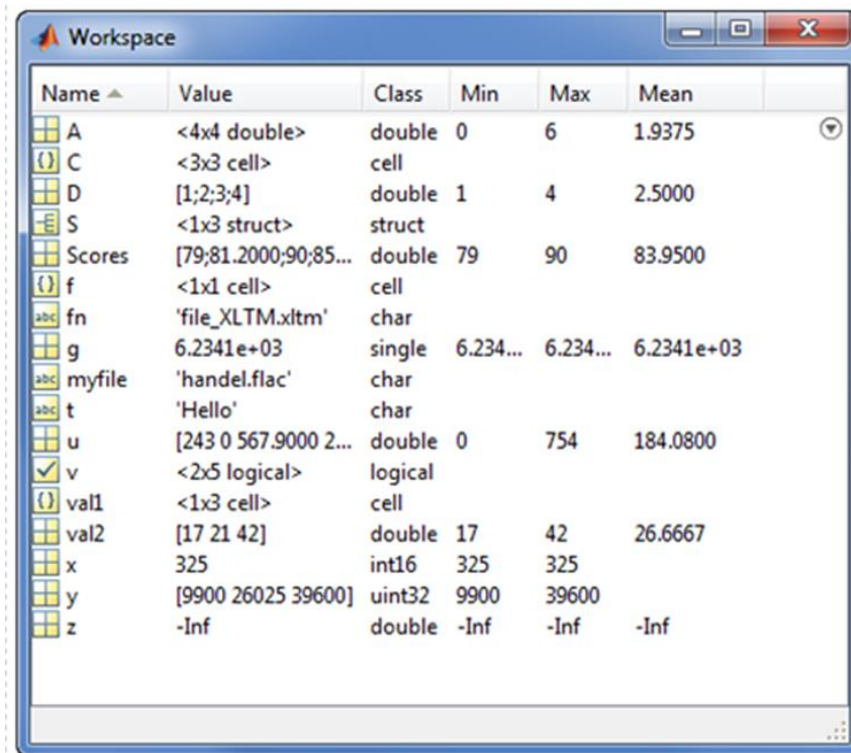


## MATLAB ShortCourse Handout

- Saving preserves the workspace in your current working folder in a compressed file with a **.mat** extension, called a **MAT-file**.
- To clear all the variables from the workspace, use the **clear** command.
- Restore data from a MAT-file into workspace using load command,

*load myfile.mat*

- To open the Workspace browser if it is not currently visible, do either of the following:
  - On the **HOME** tab, in the **Environment** section, click **Layout**. Then, under **Show**, select **Workspace**.
  - Type **workspace** at the Command Window prompt.



### Ways to Get Help, using the Command-line

- There are several commands that allow you to get help:
  - **demo** has demos of several options in MATLAB
  - **doc** provides more in-depth information
  - **clc** Window will clear the **Command Window**.

- You can transfer any command from the Command history window to Command Window by **double-clicking** (which also executes the command), or by clicking and dragging the line of code into Command Window.
- **quit** or **exit** terminates MATLAB.

**Examples:**

```
>> help sqrt <ENTER>
>> doc plot <ENTER>
>> help log <ENTER>
```

**Search Syntax and Tips**

- Find keywords in the documentation by entering text in the **Search box** on the **Desktop** or in the **Help** browser.



- When you view pages linked from the search results, search terms appear with highlights. To clear the highlights, press the **Esc** key.

Searches can include the following operators:

- ” “ **Exact phrase.** Example: "plot tools" finds pages that contain *plot tools*, in that sequence, with no words between them.
- \* **Wildcard.** Requires at least two non-wildcards characters, and cannot appear at the start of a keyword or in an exact phrase.  
*Example:* plot\* finds *plot*, *plot3*, and *plotting*.
- OR Boolean OR.** *Example:* plot OR graph finds pages with either *plot* or *graph*.
- NOT Boolean NOT.** *Example:* "plot tools" NOT "time series" finds pages with *plot tools* but excludes pages with *time series*.
- AND Boolean AND.** Implied when no operator is present between keywords. Example: **plot AND tools** is equivalent to **"plot" "tools"**.

The **Help browser** search evaluates **NOT** operators first, **OR** operators second, and **AND** operators last. For example, "plotting tool" OR "plot tools" NOT "time series" AND workspace.

**To Perform Simple Computations**

- At **prompt >>** Type a command and then press the **Enter** key.

```

Command Window
>> s = 1 + 2

s =

    3

>> fun = sin(pi/4)

fun =

    0.7071
    
```

- In the second example the trigonometric function sine and the **constant  $\pi$**  are used.
- In MATLAB they are named **sin** and **pi**, respectively. **Note** that the results of these computations are saved in **variables** whose names are chosen by the user.
- If they will be needed during your current MATLAB session, then you can obtain their values typing their names and pressing the **Enter** key. For instance,

```

>> s

s =

    3
    
```

- To clear the **Command Window** type **clc** and then press the **Enter** key.

**Note:** MATLAB uses a default variable name **ans** if an expression is typed at the prompt and it is not assigned to a variable. To change this to a Result Variable (for example), us **UP Arrow** key (to recall previous line of command), and Then **LEFT Arrow** key to modify the command. This is very useful, especially if a long expression is entered with an error, and it is desired to go back to correct it.



Example:

```
>> mynum=3

mynum =

     3

>> mynum=4+2

mynum =

     6

>> mynum=mynum+1

mynum =

     7
```

### Variable Names

- A variable name starts with a letter (a-z or A-Z), followed by any number (0-9), letters, digits, or underscores ( `_` ).
- Have a maximum length of **31 characters**.
- MATLAB software is **case sensitive**, so **A** and **a** are not the same variable.
- You **cannot** define variables with the same names as MATLAB keywords, such as **if** or **end** (for a complete list, run the **iskeyword** command).

### Examples of variable names:

- xxxxxxxxxx
- pipeRadius
- widgets\_per\_box
- mySum
- Mysum. **Note:** mySum and mysum are different variables. (**MATLAB is case sensitive**).

## MATLAB ShortCourse Handout

Invalid Name	Reason	Valid Name
6x	Does not start with a letter	x6
end	MATLAB keyword	lastValue
n!	Includes a character that is not a letter, digit, or underscore (!)	n_factorial

### Some MATLAB function descriptions

ans	- Most recent answer.
pi	- 3.1415926535897....
inf	- Infinity.
computer	- Computer type.
version	- MATLAB version number
clock	- Wall clock.
date	- Calendar
zeros	- Zeros matrix
ones	- Ones matrix
eye	- Identity matrix
magic	- Magic square
pascal	- Pascal matrix
sin	- Sine
cos	- Cosine
exp	- Exponential
log	- Natural logarithm
log10	- Common logarithm
sqrt	- Square root
abs	- Absolute value
inv	- Matrix inverse
roots	- Find polynomial roots

**Note:** Functions are all lower cases!

### Commands for Managing variables and the workspace

who	- List current variables.
whos	- List current variables, long form.
load	- Retrieve variables from disk.
save	- Save workspace variables to disk.
clear	- Clear variables and functions from memory.
size	- Size of matrix.
length	- Length of vector.
disp	- Display matrix or text.

**To Check for MATLAB Keywords:**

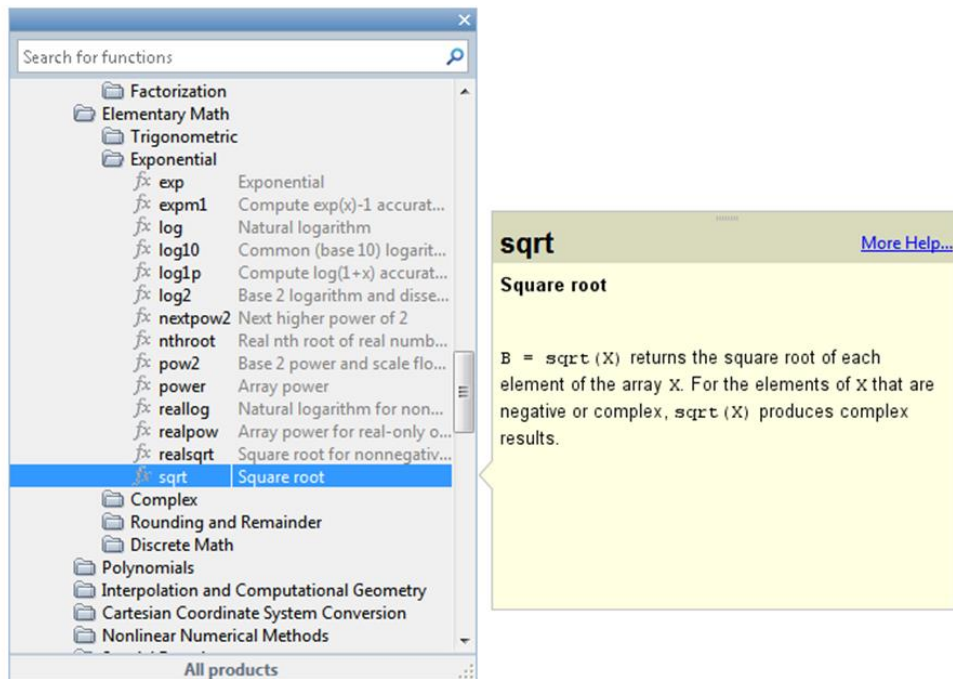
```
>> iskeyword

ans =

    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'spmd'
    'switch'
    'try'
    'while'
```

**Built-In Functions**

Functions are files that can accept input arguments and return output arguments.



### To use the Built-in Functions

```
>> log(256)
ans =
    5.5452
>> log10(256)
ans =
    2.4082
>> log2(256)
ans =
    8
```

**Note:**

- $\log(x)$  computes the natural logarithm of  $x$
- $\log_{10}(x)$  is the base 10 logarithm
- $\log_2(x)$  is the base 2 logarithm

### Making Mistakes

- When you make a mistake, you **cannot** just overwrite your command on the same line after you have executed it.
- Instead, you can enter the correct command on a new line, and then execute the new version.
- Or- recall previous commands by pressing the **up- and left-arrow keys** on your keyboard. Press the arrow keys either at an empty command line or after you type the first few characters of a command.

### Numeric Display Format

- The default format of numbers is called **short** (4 digits after the decimal point.)
- To display more digits click on **HOME** tab, click the **Preferences** button, and then change the **Numeric Format** for **Text Display**.
- You can also select a new format from within the **Command Window**. For instance, the following command **format long** changes a current format to the format long (15 digits after the decimal point).

## MATLAB ShortCourse Handout

```
>> fun = sin(pi/4)

fun =

    0.7071

>> format long
fun

fun =

0.707106781186547

>> format short
fun

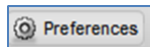
fun =

    0.7071
```

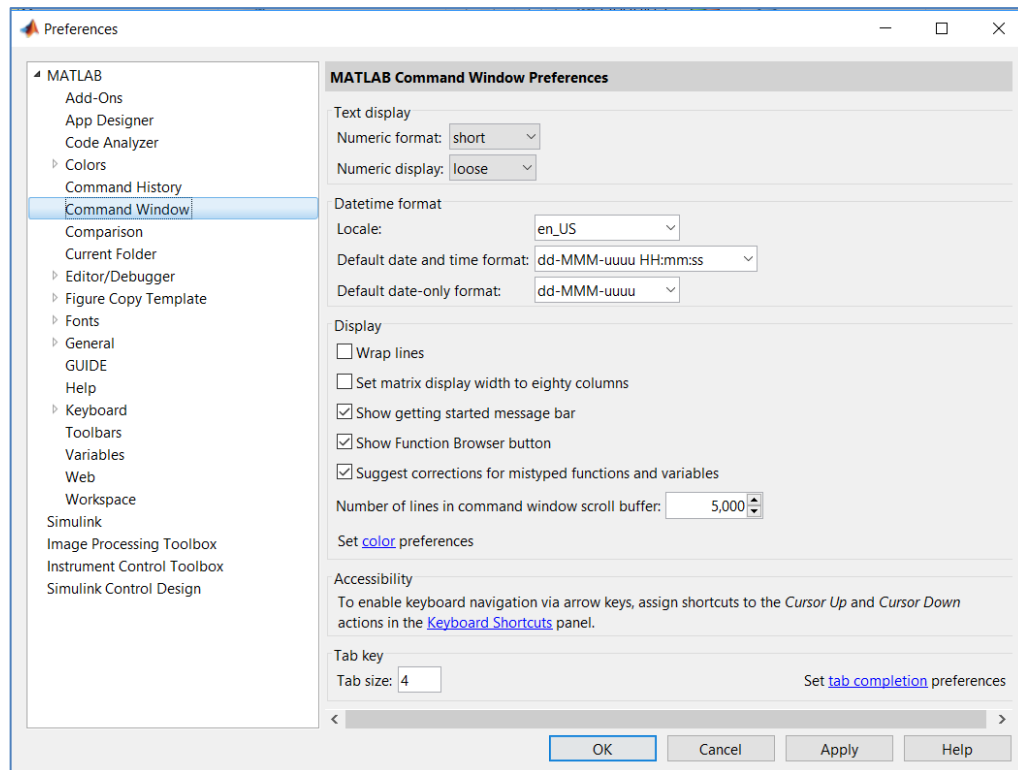
- To display **2 decimal digits**, use **format bank**.
- To close MATLAB type **exit**, in the **Command Window** and next press **Enter** or **Return** key.
- Or, from the **File** menu in the MATLAB's toolbar click on **Exit MATLAB** option. All unsaved information residing in the MATLAB **Workspace** will be lost.

**Note:** Changing display format does not change the accuracy of your results.

### Preferences



- HOME-> Preferences





## Precedence of Arithmetic Operations

Precedence	Operation
1	Parentheses, innermost first
2	Exponentiation, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

- [ ] Brackets are used to form vectors and matrices. [11 12 13; 21 22 23] is a 2-by-3 matrix. The semicolon ends the first row.
- = Used in assignment statements. B = A stores the elements of A in B.
- ' Matrix transpose. X' is the complex conjugate transpose of X.
- . Decimal point. 314/100, 3.14, and .314e1 are all the same.
- , Comma. Used to separate matrix subscripts and function arguments.
- ; Semicolon. Used inside brackets to end rows.
- : Colon. Create vectors, array subscripting, and for loop iterations.
- % Percent. The percent symbol denotes a comment; it indicates a logical end of line. Any following text is ignored.

## Some MATLAB Special Characters

### To Suppress Output with Semicolon (;)

Results of intermediate steps can be suppressed with *semicolons*. The variables created or calculated; will show in Workspace, but do not show in Command Window.

#### **Example:**

Assign values to x, y, and z, but only display the value of z in the command window:

```
>> x=5;
>> y=sqrt(59);
>> z=log(y)+x^0.25

z =

    3.5341
```

### Multiple Statements per Line

- Use commas or semicolons to enter more than one statement at once.
- Commas allow multiple statements per line without suppressing output.

```
>> a=5;b=sin(a),c=cosh(a)

b =

   -0.9589

c =

   74.2099
```

### Creation of Matrices and Vectors

- For manual entry, the elements in a vector are enclosed in square brackets []
- When creating a row vector, separate elements with a space or comma (,)
- To create a column vector, separate columns with a semicolon (;)

```
>> v = [7 3 9]
v =
     7     3     9
```

Separate columns with a semicolon

```
>> w = [2; 6; 1]
w =
     2
     6
     1
```

### Assigning Elements to a Matrix

When assigning elements to matrix, row elements are separated by **spaces**, or **commas**, and columns are separated by **semicolons (;)**

```
>> A = [1 2 3; 5 7 11; 13 17 19]

A =

     1     2     3
     5     7    11
    13    17    19
```

Or-

```
>> A=[
1 2 3
5 7 11
13 17 19]

A =

     1     2     3
     5     7    11
    13    17    19
```

Creates a 3-by-3 matrix and assigns it to a variable A.

### Transpose Operator ('), using a single quotation

The transpose operator converts a row vector to a column vector (and vice versa), and it changes the rows of a matrix to columns.

```
V =

     2     4     1     7

>> W=V'

W =

     2
     4
     1
     7

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

A =

     1     2     3
     4     5     6
     7     8     9

>> B=A'

B =

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

### Overwriting Variables

Once a variable has been created, it can be reassigned.

```
>> x=2;
>> x=x+2

x =

    4

>> y=[1 2 3 4]

y =

    1    2    3    4

>> y=y'

y =

    1
    2
    3
    4
```

### Using the Colon Notation (:)

To simplify the creation of large vectors, you define a vector by specifying the **first entry**, in **increment** (step value), and **last entry**, using the colon operator.

```
>> a=1:2:7

a =

    1    3    5    7
```

**Note:** in this case the brackets [] are not necessary to define the vector.

If you want to look at the first four entries of the vector, then you use the following notations:

```
>> s=1:4

s =

    1    2    3    4
```

**More examples:**

```
>> t=0:0.1:0.4

t =

    0    0.1000    0.2000    0.3000    0.4000

>> b=[1:6] '

b =

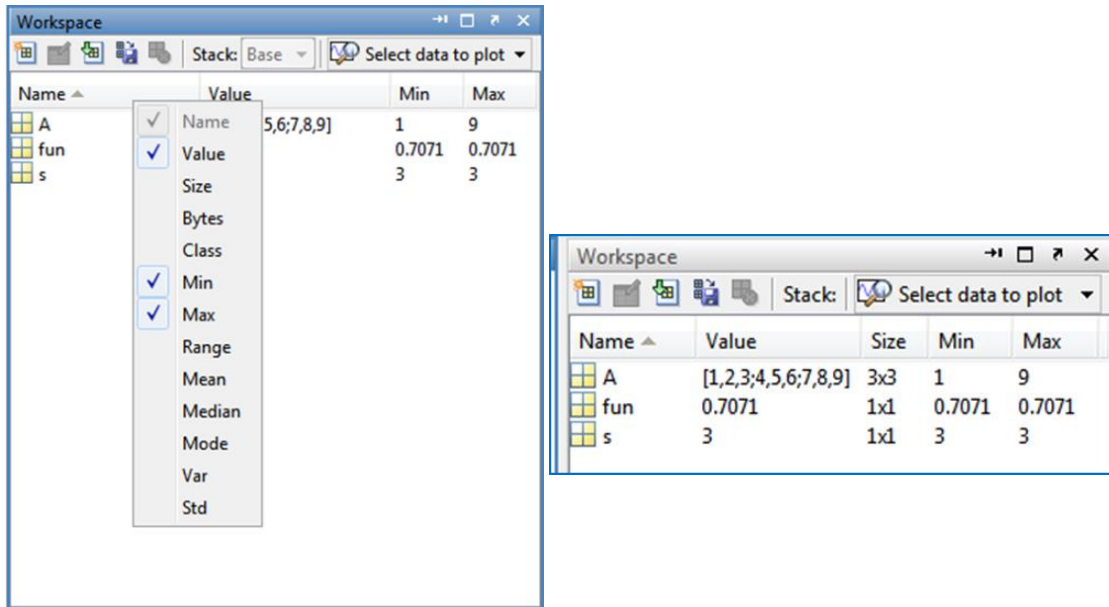
     1
     2
     3
     4
     5
     6
```

**MATLAB Data Types**

- MATLAB data (numerical variables) is placed into “**data container**” in the form of Workspace variables.
- All Workspace variables organize data into some form of array.
- 2-D numerical arrays (**matrices**) organize observations and measured variables by rows and columns, respectively.
- Multidimensional arrays – an extension of the normal 2-dimensional matrices.
- Cell and structure arrays – which organizes data of different types, sizes, units, etc.

**Workspace Window Setting**

- For each variable or object, the Workspace Browser shows: The **Name**, **Value**, **Class**, **Statistics**, **Min**, **Max**, **Range**, etc.
- To set the Workspace Window to show more about the variables, right-click on the bar with column labels.
- Check **size**.
- The yellow grid-like symbol indicates the variable “A” is a 3-by-3 matrix, and variables “fun” and “s” are 1-by-1 single values (one row by one column).



### About the Variable Editor

The Variable Editor enables you to:


- Display variables in the current workspace.
- View and edit values of one or two-dimensional arrays, character strings, cell arrays, structures, and objects and their properties. *Right-click the variable in the Workspace, then **Edit Value** (or-double-click the variable to Open the Editor Window).*
- View the contents of multidimensional arrays.
- Copy and paste data values.
- Edits you make in the Variable Editor immediately update the variable in the workspace.

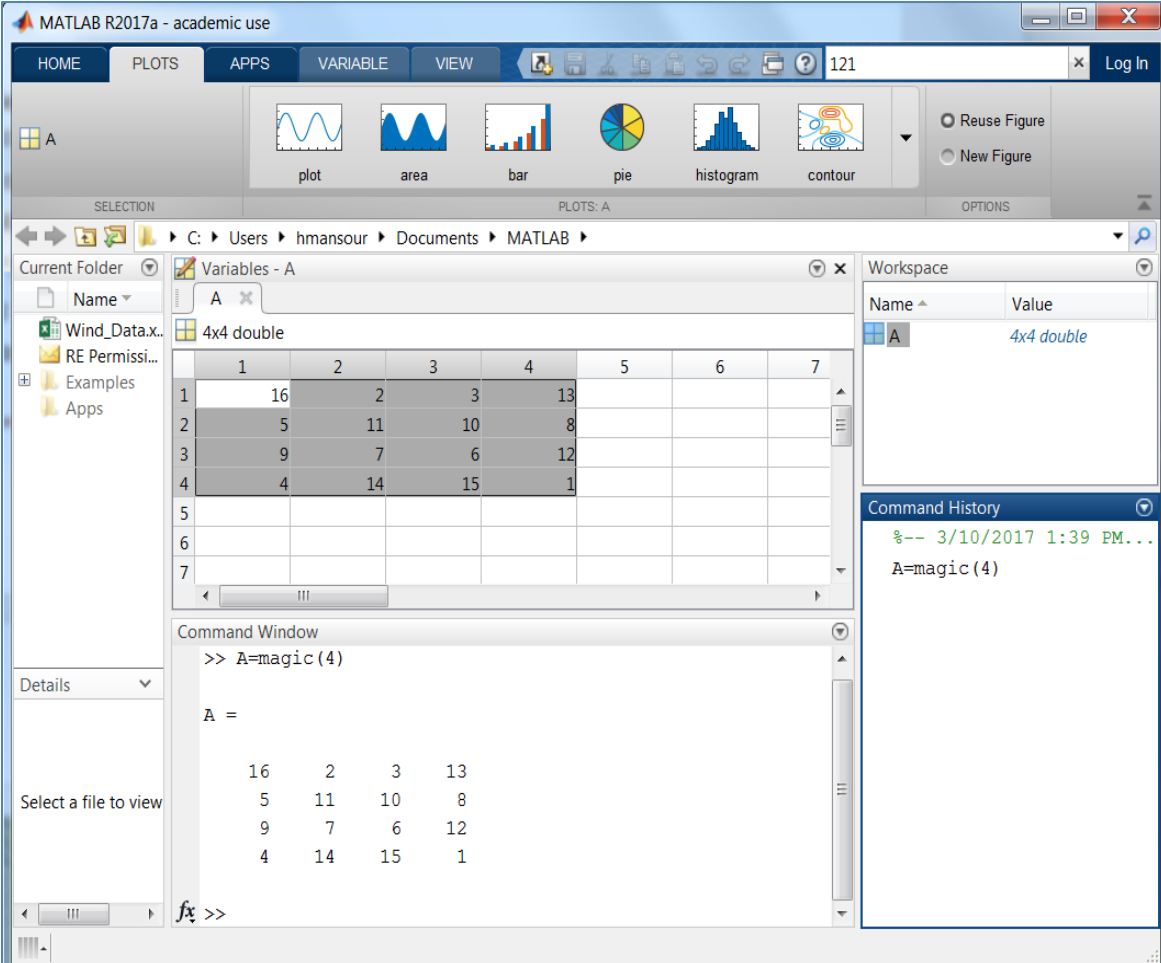
### Opening the Variable Editor

- If the Workspace browser is not open, select **Layout -> Workspace**.
- Create some workspace variables, for example:
 

```
A=magic(4);
x = 0:.1:4*pi;
y = sin(x);
```
- Then, in the Workspace browser, select the variable you want to open.
- Use (**Shift + click**) or (**Ctrl + click**) to select multiple variables, or use (**Ctrl + A**) to select all variables to open.

## MATLAB ShortCourse Handout

- Click the **Open Selection** button on the toolbar. 
  - For one variable, you can also open it by double-clicking it.
- The **Variable Editor** opens, displaying the values for the selected variable.
- The class and size of the value appear below the toolbar, and for some classes, include a link to the help for that class.
- Click the **Plot Selector** to create a graph of the selected variables.
- Click a tabbed document to display a different variable that is open in the Variable Editor.



The screenshot displays the MATLAB R2017a interface. The top toolbar includes tabs for HOME, PLOTS, APPS, VARIABLE, and VIEW. The VARIABLE tab is active, showing a plot selector with options: plot, area, bar, pie, histogram, and contour. Below the toolbar, the current folder is set to C:\Users\hmansour\Documents\MATLAB. The Variable Editor window shows a variable 'A' of type '4x4 double'. The values are displayed in a grid:

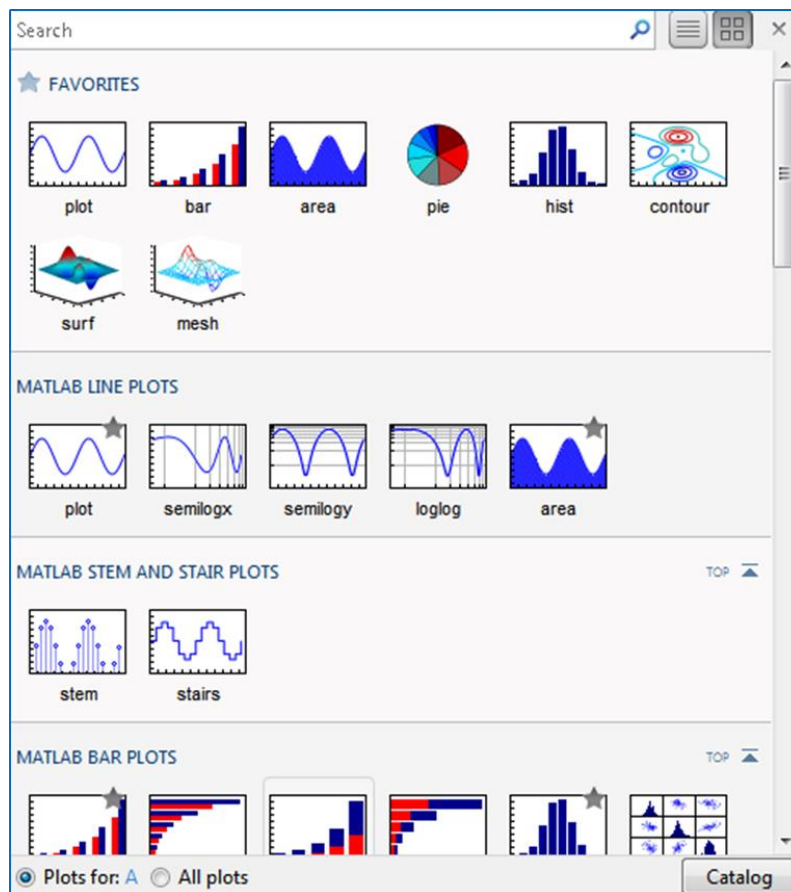
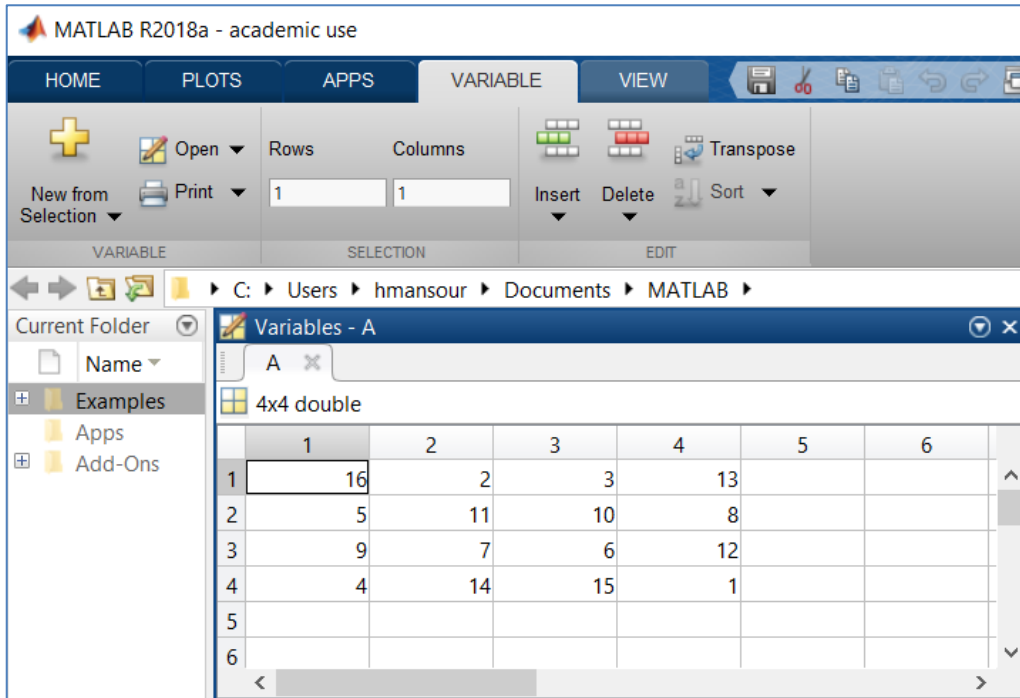
	1	2	3	4	5	6	7
1	16	2	3	13			
2	5	11	10	8			
3	9	7	6	12			
4	4	14	15	1			
5							
6							
7							

The Command Window shows the command `>> A=magic(4)` and the resulting matrix:

```
A =  
  
    16     2     3    13  
     5    11    10     8  
     9     7     6    12  
     4    14    15     1
```

The Command History window shows the command `A=magic(4)` executed on 3/10/2017 at 1:39 PM.

# MATLAB ShortCourse Handout





### Accessing Single Elements in a Matrix

- To reference a particular element in a matrix, specify its row and column number using the following syntax, where A is the matrix variable. Always specify the row first and column second: A (row, column). For example, for a 4-by-4 **magic square** A,

```
>> A=magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

- A **magic square matrix** is an n-by-n matrix constructed from integers 1 through  $n^2$  with equal rows, column, and diagonal sums, where  $n \geq 3$

```
>> sum(A)

ans =

    34    34    34    34

>> sum(diag(A))

ans =

    34
```

- You would access the element at row 4, column 2 with

```
>> A(4,2)

ans =

    14
```

### Accessing Multiple Elements

- For the following 4-by-4 matrix A, it is possible to compute the sum of the elements in the fourth column of A by typing

```
>> A=magic(4);
>> A(1,4)+A(2,4)+A(3,4)+A(4,4)

ans =

    34
```

- You can reduce the size of this expression using the colon operator.
- Subscript expressions involving colons refer to portions of a matrix.
- The expression  $A(1:m, n)$  refers to the elements in rows 1 through  $m$  of column  $n$  of matrix  $A$ .
- Using this notation, you can compute the sum of the fourth column of  $A$  more succinctly: `sum(A(1:4, 4))`

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

ans =

    34
```

### Specifying All Elements of a Row or Column, using Colon (:)

- The colon by itself refers to *all* the elements in a row or column of a matrix.
- Using the following syntax, you can compute the sum of all elements in the second column of a 4-by-4 magic square  $A$ :

```
>> A=magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

>> sum(A(:,2)) % All elements of column 2

ans =

    34
```

- By using the colon with linear indexing, you can refer to all elements in the entire matrix.

- This example displays all the elements of matrix A, returning them in a column-wise order:

```
>> A(:)
ans =
    16
     5
     9
     4
     2
    11
     7
    14
     3
    10
     6
    15
    13
     8
    12
     1
```

### Creating Matrices, using Functions

MATLAB has dozens of functions that create different kinds of matrices. To create a 3-by-3 symmetric, use the **pascal(3)**:

```
>> A=pascal(3)
A =
     1     1     1
     1     2     3
     1     3     6

>> B=magic(3)% not a symmetric matrix
B =
     8     1     6
     3     5     7
     4     9     2
```

**Note:** A **Pascal (n)** returns the Pascal matrix of order n: a symmetric positive definite matrix with integer entries taken from Pascal's triangle.

**Building Elementary Matrices and Arrays (all lowercases)**

Symbol	Explanations
<b>ones</b>	matrix of ones
<b>diag</b>	diagonal matrix
<b>inv</b>	inverse of a matrix
<b>det</b>	determinant of a matrix
<b>rank</b>	rank of a matrix
<b>cond</b>	condition number of a matrix
<b>eye(n)</b>	the n by n identity matrix
<b>tril</b>	lower triangular part of a matrix
<b>triu</b>	upper triangular part of a matrix
<b>zeros(n,m)</b>	the n by m matrix consisting of all zeros

```
>> A=eye(3)
```

```
A =
```

```

    1    0    0
    0    1    0
    0    0    1

```

```
>> B=zeros(3,2)
```

```
B =
```

```

    0    0
    0    0
    0    0

```

```
>> C=rand(3,1)
```

```
C =
```

```

    0.1419
    0.4218
    0.9157

```

```
>> triu(ones(4,4),-1) % returns the upper triangular portion of matrix ONES with 1 level below the diagonal
```

```
ans =
```

```

    1    1    1    1
    1    1    1    1
    0    1    1    1
    0    0    1    1

```

```
>> tril(ones(4,4),-1) % returns the lower triangular portion of matrix ONES with 1 level below the diagonal
```

```
ans =
```

```
0  0  0  0
1  0  0  0
1  1  0  0
1  1  1  0
```

### Adding and Subtracting Matrices

- Addition and subtraction of matrices is defined just as it is for arrays, element by element.
- Addition and subtraction require both matrices to have the same dimension, or one of them to be a scalar.

```
A = pascal(3);
B = magic(3);
X = A + B
```

```
X =
     9     2     7
     4     7    10
     5    12     8
```

```
Y = X - A
```

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

### Basic Arithmetic Operations in MATLAB

#### MATLAB Operators

assignment =	multiplication *	negation -
ellipsis ...	exponentiation ^	colon :
continuation ...	divided by /	transpose '
addition +	divided into \	
subtraction -	parentheses ( )	

**Note** that MATLAB has two divisions operators / - the right division and \ - the left division. They do not produce the same results

```
>> rd=45/3
```

```
rd =
```

```
15
```

```
>> ld=45\3
```

```
ld =
```

```
0.0667
```

### Element –by-Element (Element-wise) Operations

Arithmetic operations can be performed on each entry of an array, but using a period `.'` before certain operators such as multiplication, division, and exponentiation.

Then corresponding entries can be obtained, using the operations `.*`, `./`, and `.^` respectively.

#### Examples:

```
>> x=1:10;
```

```
>> x.^2
```

```
ans =
```

```
1 4 9 16 25 36 49 64 81 100
```

```
>> costs = Prices.*quantities;
```

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

```
>> quantities=[10 20 30 40 50];
```

```
>> item_costs=prices.*quantities;
```

```
>> total_cost=sum(item_costs)
```

```
total_cost =
```

```
550
```

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

ans =

     1     4     9
    16    25    36
    49    64    81

>> A^2

ans =

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

<b>Arithmetic Operators + - * / \ ^ '   Matrix and array arithmetic</b>	
Syntax	Description
A+B	Addition.
A-B	Subtraction.
A*B	Multiplication.
A.*B	Array multiplication. A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar.
A/B	Slash or matrix right division. B/A is roughly the same as B*inv(A). More precisely, B/A = (A'\B)'
A./B	Array right division. A and B must have the same size, unless one of them is a scalar.
A\B	Backslash or matrix left division. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation AX = B
A.\B	Array left division. A and B must have the same size, unless one of them is a scalar.
A^B	Matrix power
A.^B	Array power. A and B must have the same size, unless one of them is a scalar.
A'	Matrix transpose.
A.'	Array transpose

### Matrix Transpose

- Transposition turns a row vector into a column vector:

```
>> B=magic(3);
>> X=B'

X =

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

```
>> V=[2 0 -1]
V =
     2     0    -1
>> X=V'
X =
     2
     0
    -1
```

### Vector Products and Transpose

- If  $x$  and  $y$  are both real column vectors, then the product  $\mathbf{x}*\mathbf{y}$  is not defined, but the two products  $\mathbf{x}'*\mathbf{y}$  and  $\mathbf{y}'*\mathbf{x}$  are the same scalar.
- A row vector and a column vector of the same length can be multiplied in either order.
- The result is either a **scalar**( the *inner product*), or a **matrix** (the *outer product* ) :

```
u = [3; 1; 4];
v = [2 0 -1];
x = v*u

x =
     2

X = u*v

X =
     6     0    -3
     2     0    -1
     8     0    -4
```

### Multiplying Matrices

- MATLAB uses a single asterisk (\*) to denote matrix multiplication (*inner dimension of matrices should match*).
- The next two examples illustrate the fact that matrix multiplication is not commutative; that is,  $AB$  is usually not equal to  $BA$ :



```
>> A=pascal(3);
>> B=magic(3);
>> X=A*B

X =

    15    15    15
    26    38    26
    41    70    39

>> Y=B*A

Y =

    15    28    47
    15    34    60
    15    28    43
```

**Roots of a Polynomial**  $p_1X^n + \dots + p_nX + p_{n+1} = 0$ .

- MATLAB can compute the roots of a function **r = roots(p)** where
  - **p** is a row vector with the **n+1** coefficients of the polynomial
  - **r** is a column vector with the roots of the polynomial

**Examples:**

- The polynomial  $X^2 - 2X - 3$  is represented in MATLAB software as  $p = [1 -2 -3]$ . Remember to include all coefficients, even if 0.

```
>> p=[1 -2 -3]

p =

     1     -2     -3

>> r=roots(p)

r =

     3.0000
    -1.0000
```

- Given the roots of a polynomial, the polynomial itself can be calculated.

```
>> p=poly(r')
p =
    1.0000   -2.0000   -3.0000
```

### Solving the System of Linear Equation $AX=b$

If  $A$  is an  $n$ -by- $n$  matrix and  $B$  is a column vector with  $n$  elements, or a matrix with several such columns, then  $X = A \setminus b$  ( $A$  inverse times  $b$ ) is the solution to the equation  $AX = b$ .

Consider:

$$\begin{cases} x - 2y + 3z = 1 \\ 4x + 6z = -2 \\ 2x - y + 3z = -1 \end{cases}$$

Where  $X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$  is a column vector of unknowns  $x, y, z$ . Then, we need to solve

$AX = b$ , where

$$\mathbf{A} = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 0 & 6 \\ 2 & -1 & 3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}$$

One way to solve this system is  $X = \text{inv}(A) * b$ . A **better way**, from both and execution time and numerical accuracy standpoint, is to use the matrix division operator  $\mathbf{X} = \mathbf{A} \setminus \mathbf{b}$  (using the backslash operator, also called *left division*). This produces the solution using *Gaussian Elimination*, without forming the inverse.

```
A =
     1     -2     3
     4     0     6
     2     -1     3

>> b=[1;-2;-1;]

b =
     1
    -2
    -1

>> X=A\b

X =
    -2
     0
     1
```

```
>> A=[1 -2 3;4 0 6;2 -1 3]

A =
     1     -2     3
     4     0     6
     2     -1     3

>> b=[1; -2; -1]

b =
     1
    -2
    -1

>> C=inv(A)

C =
   -1.0000   -0.5000    2.0000
         0    0.5000   -1.0000
    0.6667    0.5000   -1.3333

>> X=C*b

X =
   -2.0000
         0
    1.0000
```

### Creating Symbolic Variables and Expressions

- **Symbolic Math Toolbox™** provides tools for solving and manipulating symbolic math expressions such as **differentiation**, **integration**, and **equation solving**.
- Symbolic expressions are used to simplify mathematical expressions.
- Symbolic objects enable you to perform mathematical operations in the MATLAB workspace analytically, without calculating numeric values (*gives us symbolic equations as results, rather than matrices or vectors*).
- You can use **sym** or **syms** to create symbolic variables in one function call (to declare variables as symbolic objects).

#### Example:

To declare  $f(x, y)$  as a symbolic function that accepts two arguments  $x$ , and  $y$ :

```
>> syms f(x,y)
>> f(x,y)=x+2*y

f(x, y) =

x + 2*y
```

Or-

```
>> syms x y
>> f(x,y)=x+2*y

f(x, y) =

x + 2*y
```

Then, to compute the function value at point  $x=1$  and  $y=2$ :

```
>> f(1,2)

ans =

5
```

**Note:** You can use **sym** or **syms** to create symbolic variables.

#### The syms command:

- Does not use parentheses and quotation marks: **syms x**
- Can create multiple objects with one call
- Serves best for creating individual single and multiple symbolic variables

#### The sym command:

- Requires parentheses and quotation marks: **x = sym('x')**
  - When creating a symbolic number with 15 or fewer decimal digits, you can skip the quotation marks: **f = sym(5)**
- Creates one symbolic object with each call.
- Serves best for creating symbolic numbers and symbolic expressions.
- Serves best for creating symbolic objects in functions and scripts.

**Using the solve function**

Symbolic math functions can be used to solve a single equation, a system of equations, and differential equations. You can use the double equation sign (==) to define an equation. Then you can solve the equation by calling the solve function. Solve function; solves the equation for specified variable.

For example: to solve  $x^2-1$

```
>> syms x
>> solve (x^2-1)

ans =

     1
    -1
```

Or-

```
>> syms x
>> r=solve(x^2-2*x-3)

r =

     3
    -1

>> solve(x+1==2,x)

ans =

     1

>> solve(x^2-2*x-3==0)

ans =

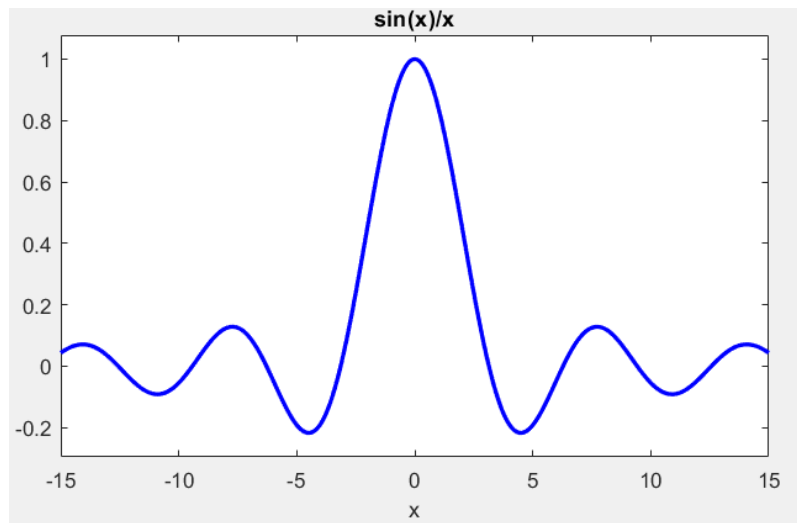
     3
    -1
```

**Note:** double equal sign (==) is a Relational Operator, and is used when the first input is equal to the second input (two-input mode).

**Symbolic Expressions Example**

- For example, create the following expression  $f$ , and the plot of  $f$  for the values of variable  $x$  from -15 to 15:

```
>> syms x
>> f=sin(x)/x;
>> ezplot(f, [-15,15]) %plots f over the domain -15,x,15
```



### Integration

- The ***int*** function can be used for definite integration by passing the limits over which you want to calculate the integral.
- If ***f*** is a symbolic expression, then ***int(f)*** attempts to find another symbolic expression, ***F***, so that ***diff(F) = f***. That is, ***int(f)*** returns the **indefinite integral** or **antiderivative** of ***f*** (provided one exists in closed form).
- Here are some examples of integration of expressions containing those variables.

**Note:** If MATLAB is unable to find an answer to the integral of a function ***f***, it just returns ***int(f)***.

### Some integration of expressions containing those variables

f	int(f)
<pre>syms x n f = x^n;</pre>	<pre>int(f)  ans = piecewise([n == -1, log(x)], [n ~= -1, x^(n + 1)/(n + 1)])</pre>
<pre>syms y f = y^(-1);</pre>	<pre>int(f)  ans = log(y)</pre>
<pre>syms x n f = n^x;</pre>	<pre>int(f)  ans = n^x/log(n)</pre>
<pre>syms a b theta f = sin(a*theta+b);</pre>	<pre>int(f)  ans = -cos(b + a*theta)/a</pre>
<pre>syms u f = 1/(1+u^2);</pre>	<pre>int(f)  ans = atan(u)</pre>
<pre>syms x f = exp(-x^2);</pre>	<pre>int(f)  ans = (pi^(1/2)*erf(x))/2</pre>

### Definite integration

Definite Integral	Command
$\int_a^b f(x)dx$	<code>int(f, a, b)</code>
$\int_a^b f(v)dv$	<code>int(f, v, a, b)</code>

**Examples of Integration**

f	a, b	int(f, a, b)
<pre>syms x f = x^7;</pre>	<pre>a = 0; b = 1;</pre>	<pre>int(f, a, b) ans = 1/8</pre>
<pre>syms x f = 1/x;</pre>	<pre>a = 1; b = 2;</pre>	<pre>int(f, a, b) ans = log(2)</pre>
<pre>syms x f = log(x)*sqrt(x);</pre>	<pre>a = 0; b = 1;</pre>	<pre>int(f, a, b) ans = -4/9</pre>
<pre>syms x f = exp(-x^2);</pre>	<pre>a = 0; b = inf;</pre>	<pre>int(f, a, b) ans = pi^(1/2)/2</pre>
<pre>syms z f = besselj(1,z)^2;</pre>	<pre>a = 0; b = 1;</pre>	<pre>int(f, a, b) ans = hypergeom([3/2, 3/2], [2, 5/2, 3], -1)/12</pre>

**Symbolic Integration, using *int* Function**

The *int* command can be used for definite integration by passing the limits over which you want to calculate the integral  $\int_a^b f(x)dx = f(b) - f(a)$ , we write int(x, a, b)

Syntax:

- int(expr, var) - computes indefinite integral of the expr with respect to variable var.



- `int(expr,var,a,b)` - computes the definite integral of `expr` with respect to `var` from `a` to `b`. For example to find  $\int_0^1 x^7 dx$

```
>> syms x
>> f=x^7

f =

x^7

>> int(f,0,1)

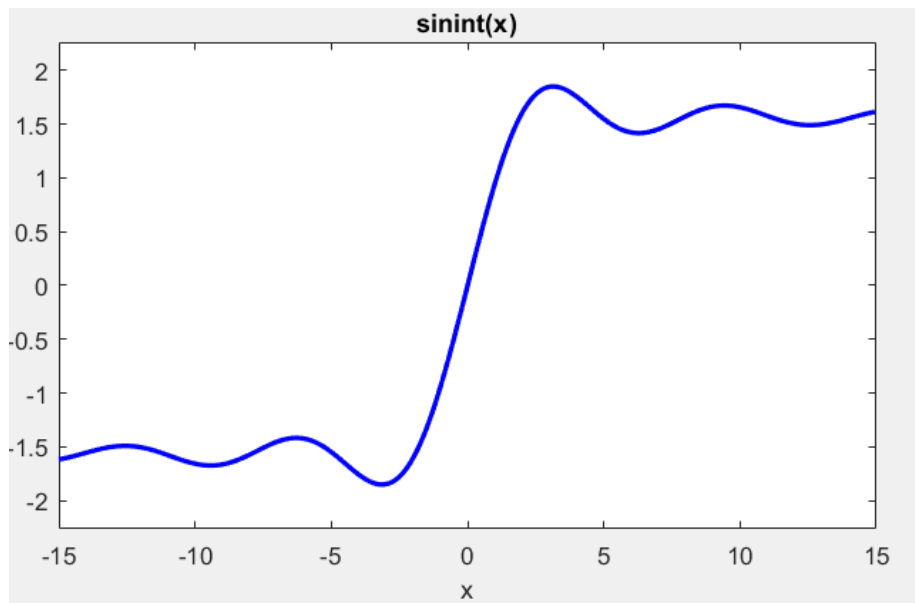
ans =

1/8
```

**To compute integrals using the `int` function**

- To find the integral of  $f = \sin(x)/x$  and visualize the sine integral function from -15 to 15:

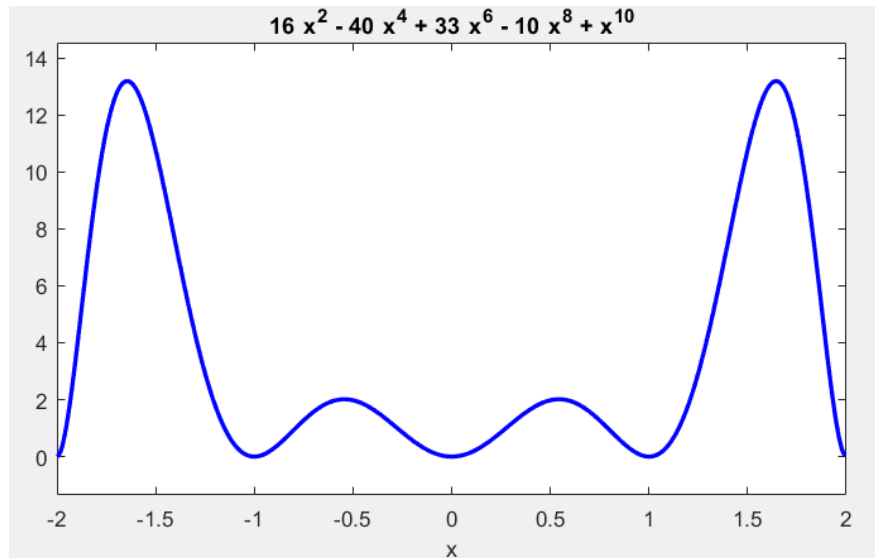
```
>> syms x
>> f=sin(x)/x; F=int(f,x); ezplot(F,[-15,15])
```



**Definite Integrals Example:**

- Let  $f(x) = x^{10} - 10x^8 + 33x^6 - 40x^4 + 16x^2$
- Use ezplot to plot  $f(x)$  for  $-2 \leq x \leq 2$
- Use the Symbolic Toolbox to find  $\int_{-2}^2 f(x)$

```
>> syms x
>> f(x)=x^10-10*x^8+33*x^6-40*x^4+16*x^2;
>> ezplot(f(x), [-2,2])
```



```
>> F=int(f,-2,2)

F =

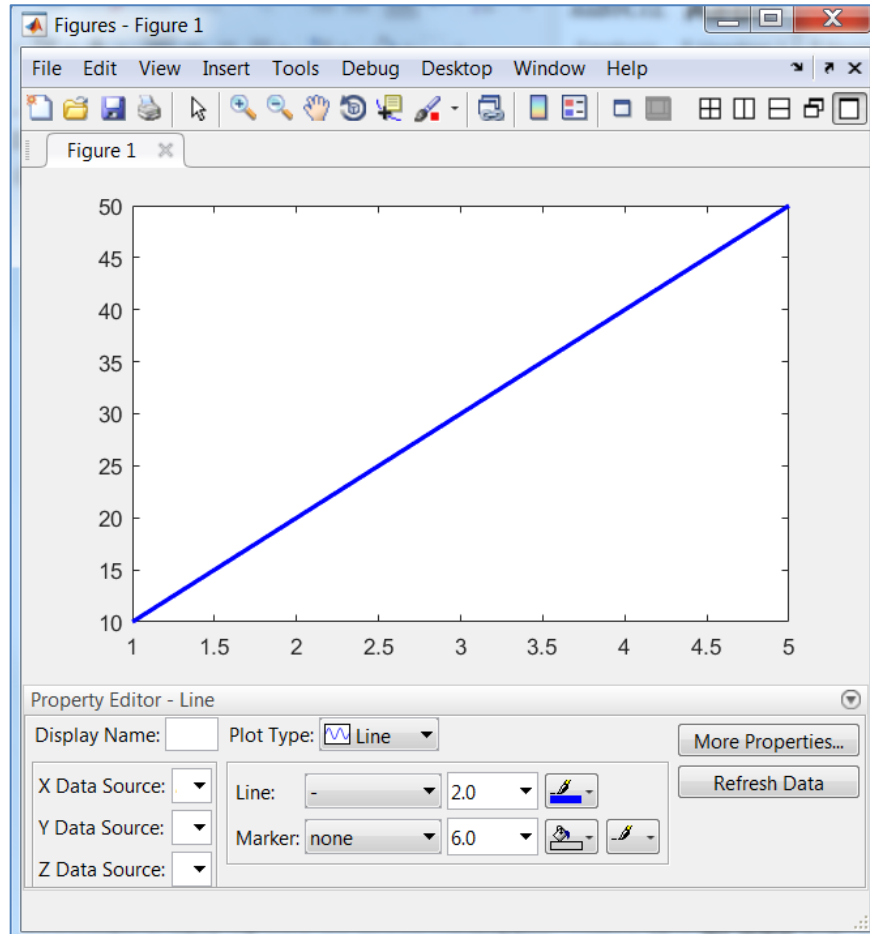
10240/693
```

**Graphics Window**

Let's create 2 arrays of x and y. A semicolon suppresses the output from the commands. Then we use the Plot Command: plot(x,y), to create a linear 2-dimensional x-y plot (y vector versus x vector).

## MATLAB ShortCourse Handout

```
>> x=[1,2,3,4,5];  
>> y=[10,20,30,40,50];  
>> plot(x,y)
```



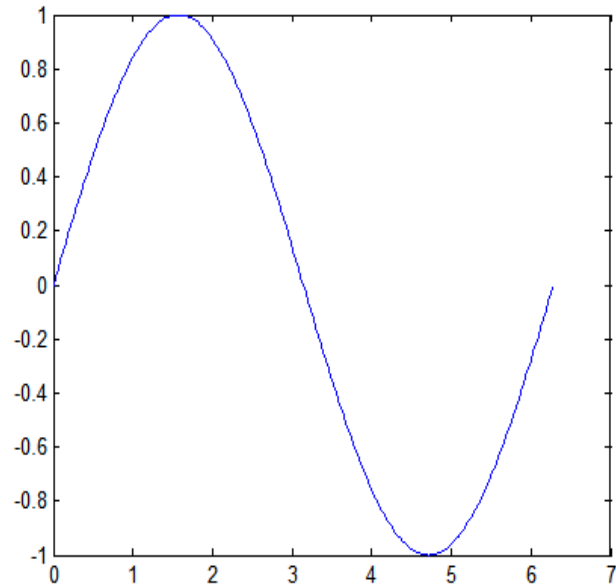
You can also generate a plot of one or more variables, using the **PLOTS** tab on the **Toolstrip**.

### Tips:

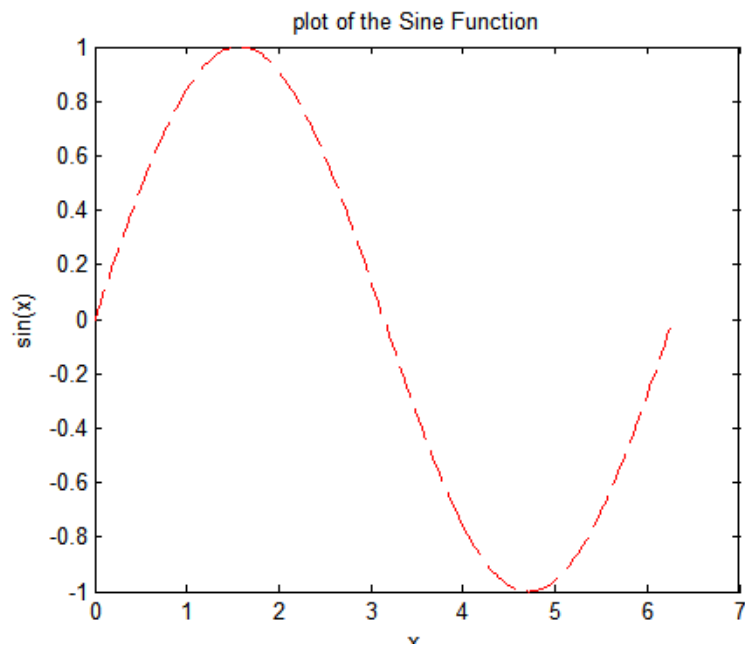
- Use **plot** function, to create linear 2-D plots.
- Use **plots3** function, to create 3-D plots.
- Use **ezplot** function, to plot an expression, equation, or function  $f$  (univariate).

**2-D Plots**

```
>> x=0:pi/100:2*pi;
>> y=sin(x);
>> plot(x,y);
>> xlabel ('X');
>> ylabel('sin(x)');
>> title('Plot of the Sine Function')
```



```
>> x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y,'r--')% using a red dash line
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function')
```



### Specifying Graph's Line, Mark, and Color Styles

Type	Values	Meanings
Color	'c' 'm' 'y' 'r' 'g' 'b' 'w' 'k'	cyan magenta yellow red green blue white black
Line style	'-' '--' '.' '-.' no character	solid dashed dotted dash-dot no line
Marker type	'+' 'o' '*' 'x' 's' 'd' '^' 'v' '> '< 'p' 'h' no character	plus mark unfilled circle asterisk letter x filled square filled diamond filled upward triangle filled downward triangle filled right-pointing triangle filled left-pointing triangle filled pentagram filled hexagram no marker


### 3-D Plots

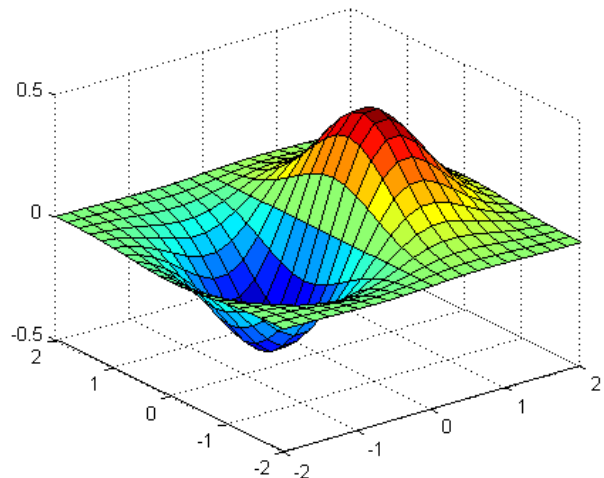
```
>> [X,Y] = meshgrid(-2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
>> surf(X,Y,Z)
```

**Note:**

**.\*** symbol is used for element-by-element multiplication.

You may want to enable zooming graph

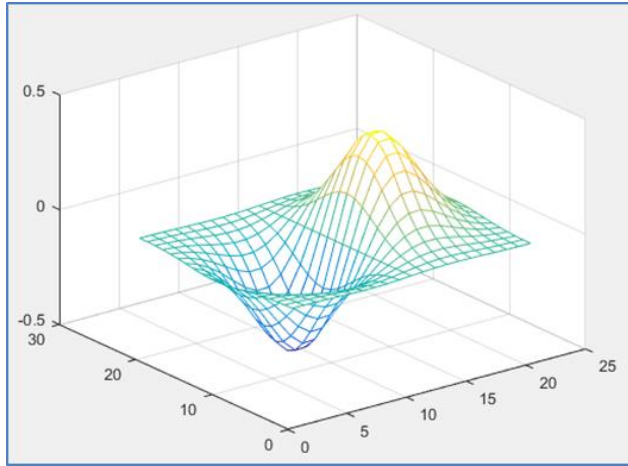
tools  to see greater detail in a small area of the graph.



# MATLAB ShortCourse Handout

The command `mesh(Z)` creates a three dimensional plot of the elements of the matrix `Z`.

```
>> mesh(z)
```



**Try:**

```
mesh(eye(10))
```

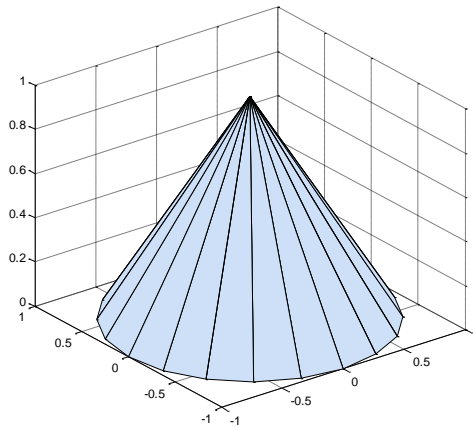
```
Surf(eye(10))
```

## Common Graphics Functions in MATLAB

Line Plots	Pie Charts, Bar Plots, and Histograms	Discrete Data Plots	Polar Plots	Contour Plots	Vector Fields	Surface and Mesh Plots	Polygons	Animation	
<code>plot</code> 	<code>area</code> 	<code>stairs</code> 	<code>polar</code> 	<code>contour</code> 	<code>quiver</code> 	<code>surf</code> 	<code>mesh</code> 	<code>fill</code> 	<code>animatedline</code> 
<code>plot3</code> 	<code>pie</code> 	<code>stem</code> 	<code>rose</code> 	<code>contourf</code> 	<code>quiver3</code> 	<code>surfc</code> 	<code>meshc</code> 	<code>fill3</code> 	<code>comet</code> 
<code>loglog</code> 	<code>pie3</code> 	<code>stem3</code> 	<code>compass</code> 	<code>contour3</code> 	<code>feather</code> 	<code>surf1</code> 	<code>meshz</code> 	<code>patch</code> 	<code>comet3</code> 
<code>semilogx</code> 	<code>bar</code> 	<code>scatter</code> 	<code>ezpolar</code> 	<code>contourslice</code> 	<code>streamslice</code> 	<code>ezsurf</code> 	<code>waterfall</code> 		
<code>semilogy</code> 	<code>barh</code> 	<code>scatter3</code> 		<code>ezcontour</code> 	<code>streamline</code> 	<code>ezsurf</code> 	<code>ezmesh</code> 		
<code>errorbar</code> 	<code>bar3</code> 	<code>spy</code> 		<code>ezcontourf</code> 	<code>streamribbon</code> 	<code>ribbon</code> 	<code>ezmeshc</code> 		
<code>ezplot</code> 	<code>bar3h</code> 	<code>plotmatrix</code> 			<code>streamtube</code> 	<code>pcolor</code> 			
<code>ezplot3</code> 	<code>histogram</code> 				<code>coneplot</code> 				
	<code>pareto</code> 								

### Cylinder Function

- The function cylinder is used for plotting a surface of revolution. The command `cylinder(r, n)` is used where, parameter `r` stands for the vector that defines the radius of cylinder along the z-axis and `n` specifies a number of points used to define circumference of the cylinder.
- For example, `>> cylinder([1 0])` Plots a cone with the base radius equal to one and the unit height.



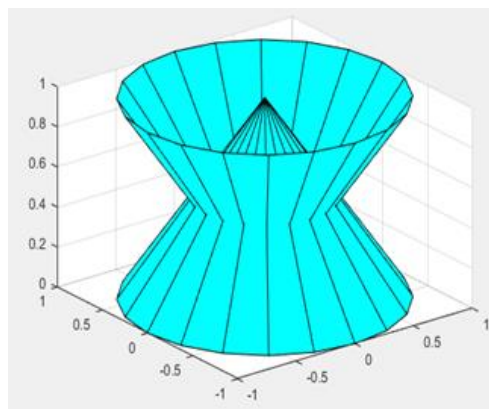
### To save your graph:

- **File -> Save As** type **(\* .fig)**
- Specify a name for your file -> click **OK**.

### Cones are special cylinders:

### Example:

```
>> cylinder([1 0])    % displays a cone
    hold on          % keep the plot
    cylinder([0 1])   % cone in opposite direction
```

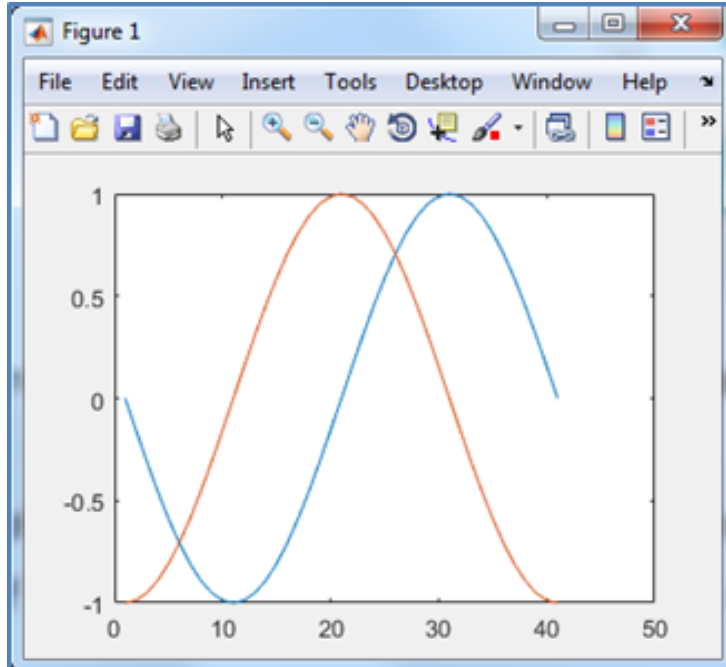


- The cylinders are produced by **cylinder** function, all have unit height.
- The argument of cylinder is a vector which contains the distance from the wall of the cylinder to the vertical axis at equally spaced points. This should explain why we get cones with [1 0] (distance 1 to axis at base and distance 0 at top) and why [0 1] is oriented in the opposite direction.

**Note:** “**hold on**” holds the current plot and all axis properties so that subsequent graphing commands add to the existing graph. “**hold off**” returns to the default mode whereby plot commands erase the previous plots and reset all axis properties before drawing new plots.

**Example:**

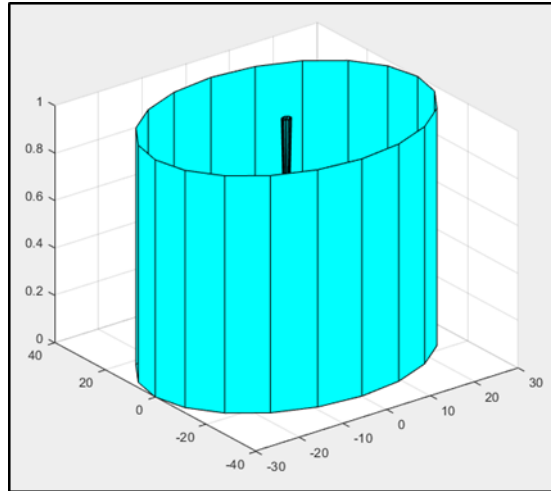
```
>> x=-pi:pi/20:pi;  
>> plot (sin(x))  
>> hold on  
>> plot(cos(x))  
>> hold off
```



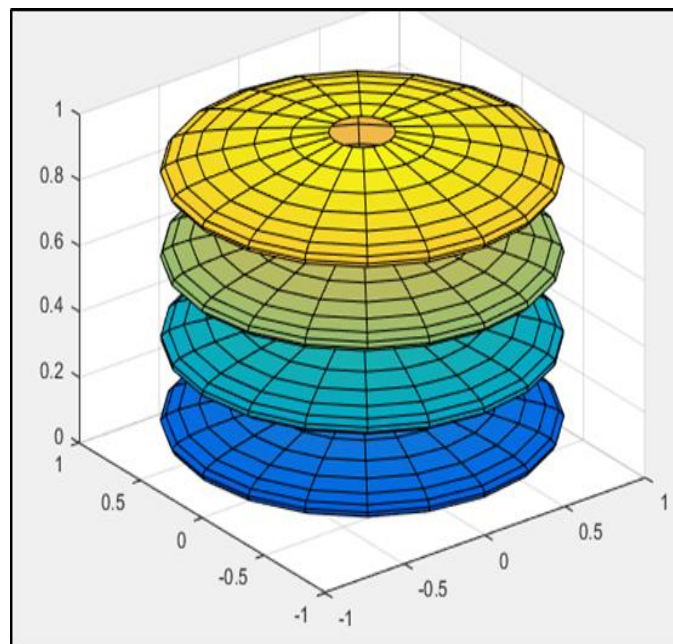


**Cylinder Function Examples:**

```
>> [x,y,z] = cylinder(30); % replace default by 30
>> surf(x,y,z)           % to get finer picture
```



```
>> t = 0:0.1:2*pi; % prepare sampling range
r = sin(2*t); % sample a sine
[x,y,z] = cylinder(r); % create cylinder
surf(x,y,z) % to get finer picture
```



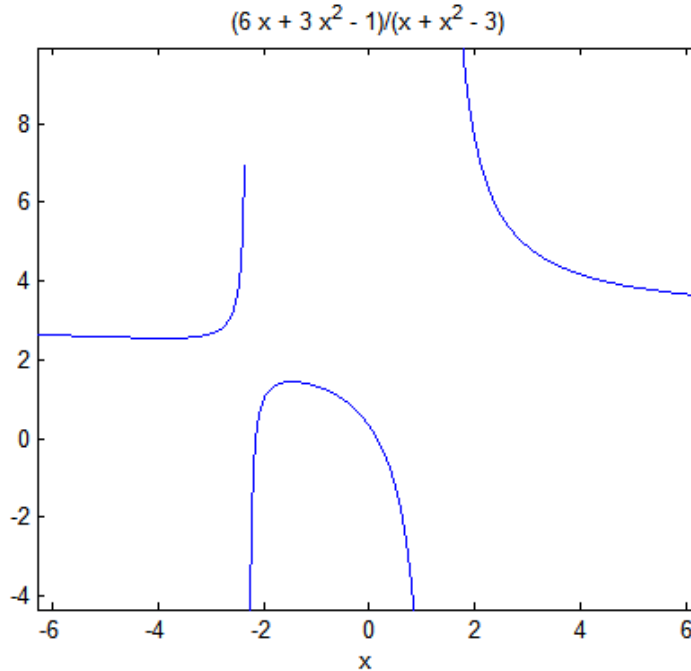
**Creating Symbolic Variables and Functions (Calculus Example)**

- To create the function  $f(x) = \frac{3x^2 + 6x - 1}{x^2 + x - 3}$  Enter the following commands (using **syms** to create symbolic variable x):

```
>> syms x
num = 3*x^2 + 6*x - 1;
denom = x^2 + x - 3;
f = num/denom
```

This returns  $f = (3*x^2 + 6*x - 1)/(x^2 + x - 3)$

You can plot the graph of  $f$  by entering **ezplot(f)**. This displays the following plot.

**Sort Command**

The Sort command arranges the values of a vector into ascending order, and if it is used with a matrix; it sorts each column of a matrix in ascending order, unless you specify.

```

>> A = [ 3 7 5
0 4 2 ]

A =

     3     7     5
     0     4     2

>> sort(A)

ans =

     0     4     2
     3     7     5

>> sort(A, 'descend')

ans =

     3     7     5
     0     4     2
    
```

**Example:**

Suppose that the vector A represents the distribution of measurements:

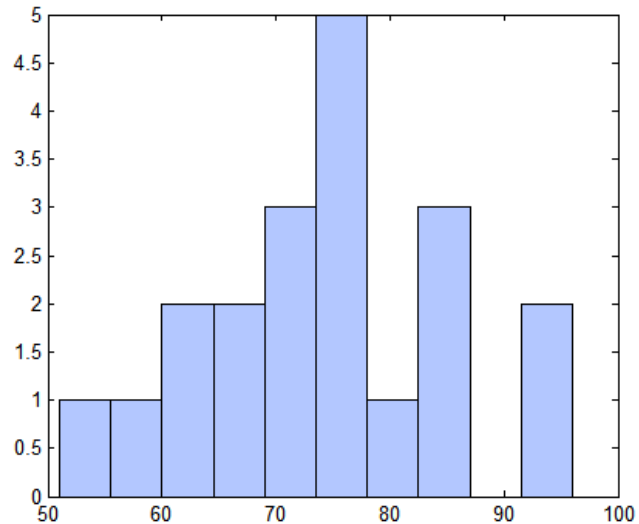
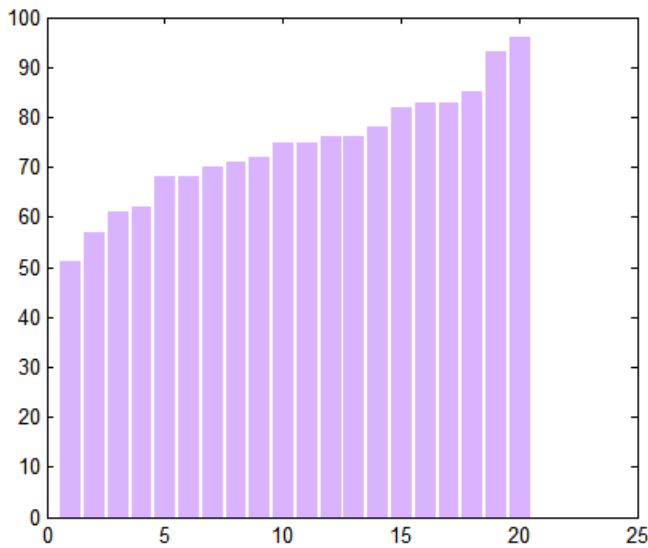
A=[68,83,61,70,75,82,57,51,76,85,62,71,96,78,76,68,72,75,83,93]

To create a bar graph, and a histogram of these measurements:

1. Sort the data to create a bar graph of scores
2. Create a histogram of the scores.

```

fx >> A=[68,83,61,70,75,82,57,51,76,85,62,71,96,78,76,68,72,75,83,93];
x=sort(A);
bar(x)
hist(x)
    
```

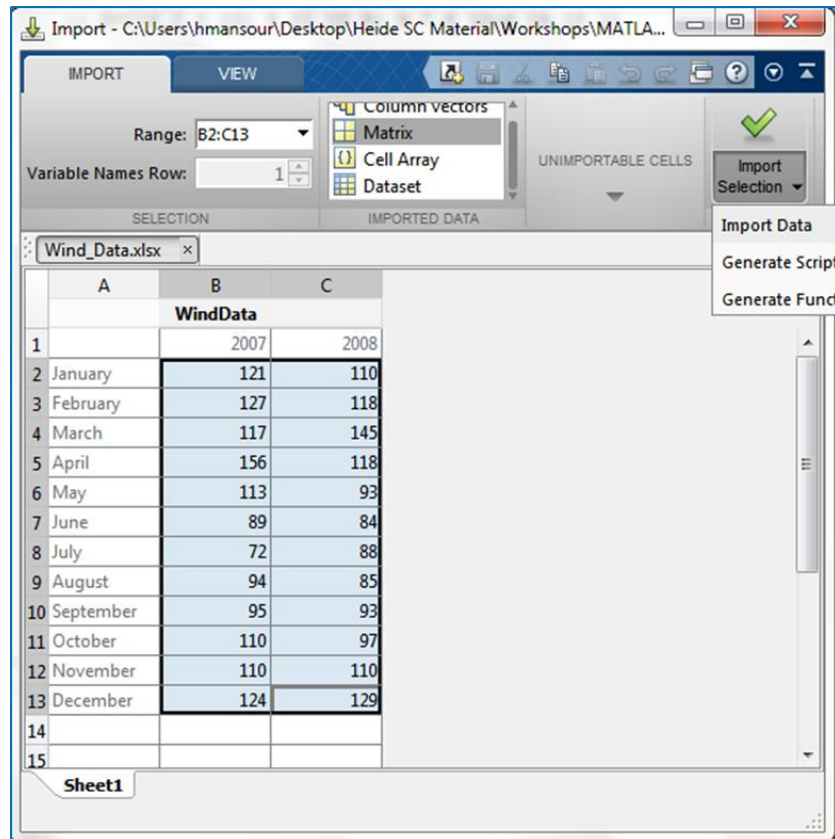


### Importing Excel Spreadsheet Data, using the Import Tool

- On the **HOME** tab, in the **VARIABLE** section, click Import Data
- Browse to find **Wind-Data.xlsx** file (In SC folder).
- Select **Import Data** -> **Select** just the data (not the name of the variables) to be imported (change the selection of data to **B2:C13**).
- Select the **Numeric Matrix** from the drop down **Imported data** list.
- **Import Matrix** -> **Import Data**. Once the Import Wizard has completed the import, the matrix name untitled will appear in Workspace window. **Close** the Import window.
- **Rename** this Matrix to **Data**.

**Example:** A sample of weather data collected for two successive years is given in the following table:

	2007	2008
January	121	110
February	127	118
March	117	145
April	156	118
May	113	93
June	89	84
July	72	88
August	94	85
September	95	93
October	110	97
November	110	110
December	124	129



- Then write a MATLAB script to find the average peak wind speed by month, and by year.
- Find the highest peak wind speed by the month of year. Open the **Script Editor**, and then type the following codes:

```
wd=WindData; %copy data into wd
disp('Monthly Average')
disp(mean(wd,2)) % return row vector containing the mean over 2 columns dimension for rows for each month
disp('Yearly Average')
disp(mean(wd)) % mean over column vectors for years
disp('Peak Wind and Corresponding Months')
[Peak Month]=max(max(wd'))
[Peak Year]=max(max(wd))
```

- Then click the **RUN** button, to run the script. Functions **sum**, **max**, and **mean** will be applied to each column and a row vector of results are returned.

## MATLAB ShortCourse Handout

Monthly Average

```
115.5000
122.5000
131.0000
137.0000
103.0000
 86.5000
 80.0000|
 89.5000
 94.0000
103.5000
110.0000
126.5000
```

Yearly Average

```
110.6667 105.8333
```

Peak Wind and Corresponding Month

Peak =

```
156
```

Month =

```
4
```

Peak Wind and Corresponding Year

Peak =

```
156
```

Year =

```
1
```

### Some Important Commands

- To run **MATLAB demos**:
  - Type **demo** in the command window, and press the **Enter** key.
- To clear Windows:
  - Type **clc** to clear the contents of the command window. This removes all lines in your Command Window.
  - Type **clf** to clear the contents of the figure window
- To clear Variables:
  - Type **clear** to clear the contents of the workspace. This deletes all stored variables in your Workspace.
- To close all of the figure generated:
  - Type close all.

**Note:** it is a good idea that at the beginning of every program to include:

- **close all** – closes all of the figures that you have generated in your program.
- **clear all** – deletes all stored variables in your workspace.
- **clc** – removes all lines in your command window.

This will clear all the work done in previous sessions of the program.

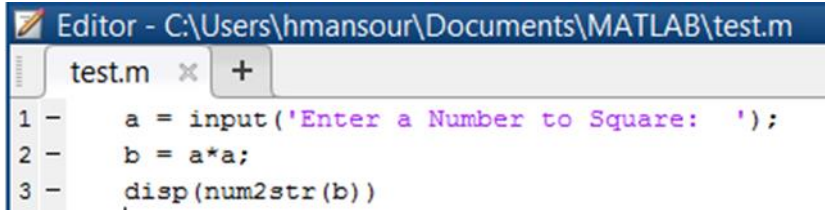
### The M-Files

- If you wish to execute repeatedly some set of commands, and possibly change input parameters as well, then you should create a **script M-File**.
- To make the **m-file** click on **New Script** button on the **File** group.
- Once you are done with typing your codes, click on **Save**, in the **MATLAB Editor** Window and select **Save As...**

**Note:** Your file should be saved in the directory that is in MATLAB's search path.

### Example:

Begin by creating the following script file (using the Script Window):



```

Editor - C:\Users\hmansour\Documents\MATLAB\test.m
test.m x +
1 - a = input('Enter a Number to Square: ');
2 - b = a*a;
3 - disp(num2str(b))
    
```

- Save it as **test1.m**.

- Click the **Run** button, and it will ask for a number.
- Type a number and press the **Enter** key – it will square it and display the result.
- **Or-** in Command window, type **test1** and press the **Enter** key, and follow the instruction.

### Script Example

The body mass index script, using the metric system:

- Let's Open another blank Script file (Home -> New Script), and then type the following program:

```
% File:   body_mass_index.m
% Author: Heide Mansouri
% Date:   Summer 2013
% Course: MATLAB ShortCourse
% This program determines a person's BMI (or body mass index)
% by dividing the weight of the person (in kilograms) by the
% (height of the person squared).
% Used Variables and what they mean:
% weight : a floating point number representing the weight in Kilograms
% height  : a floating point number representing the height in meters
% BMI: body mass index

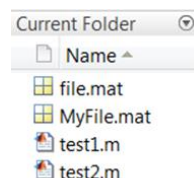
kg = input('Type your weight in kilograms: ');
m = input('Type your height in meters: ');
BMI= kg/ m^2;
disp(num2str(BMI)) % display the body mass index
```

- **File** -> **Save As** -> test2 -> Save.
- Click **Run**.
- Close The Editor window.

### Saving and loading Files

- **To save** your **Workspace** variables to a file (binary formatted **MAT-file**).
  - **Right-click** the Workspace -> **Save** -> Choose a name for your file (**file-name**) and click on **Save** button.

**Remember** that the file you save (**\*.mat**) will be stored in the *current directory* (C:\users\Username\Documents\MATLAB for example).





- Another way of saving your workspace is to type **save filename** in the **Command Window**. The command **save filename** saves only the variables.
- To load contents of the file named **filename** into MATLAB's workspace type **load filename** in the **Command Window**.

**Practice:**

```
a=[1 2 3 ]
save MyFile
close all
clear all
clc
load MyFile
```

**The Difference between \*.m and \*.mat files in MATLAB**

- Files with **.m** extension contain **MATLAB code**, either in the form of a script or a function.
- Files with **.mat** extension contain **MATLAB formatted data**, and can be loaded from or written to these files using the functions **load** and **save**.

**Quitting the MATLAB Program**

- To end your MATLAB session, select **File > Exit MATLAB** in the desktop
- Or- type **quit** in the Command Window.

**Online Resources**

- **Up and Running with MATLAB** video training courses  
<http://library.ttu.edu/lynda>
- **MATLAB Examples** <http://www.mathworks.com/examples/>
- **FREE MATLAB interactive course** from **MATLAB Academy**  
<http://matlabacademy.mathworks.com>

**Where to Get Help**

- Texas Tech University has site licenses for MATLAB software, and as a member of the TTU community, you have the right to have full 24 hours support from MATLAB.
- If you need help from me, please e-mail [heide.mansouri@ttu.edu](mailto:heide.mansouri@ttu.edu). Or call 834-2935 to make an appointment.

Please e-mail your comments or suggestions to: [heide.mansouri@ttu.edu](mailto:heide.mansouri@ttu.edu)