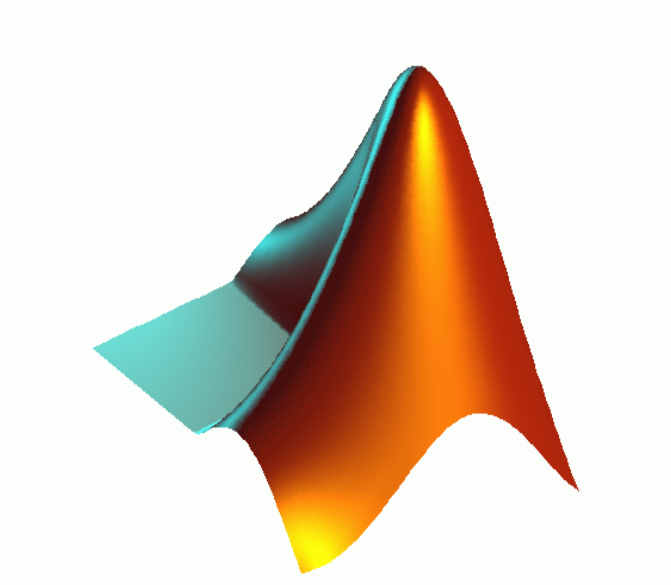
جامعة سبها

Introduction to Matlab



Dr Hasan Alkhadafe

- MATLAB stands for M*atrix Laboratory.*
- Matlab had many functions and toolboxes to help in various applications
- It allows you to solve many technical computing problems, especially those with matrix and vector formulas, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

The MATLAB System

MATLAB system consists of these main parts:

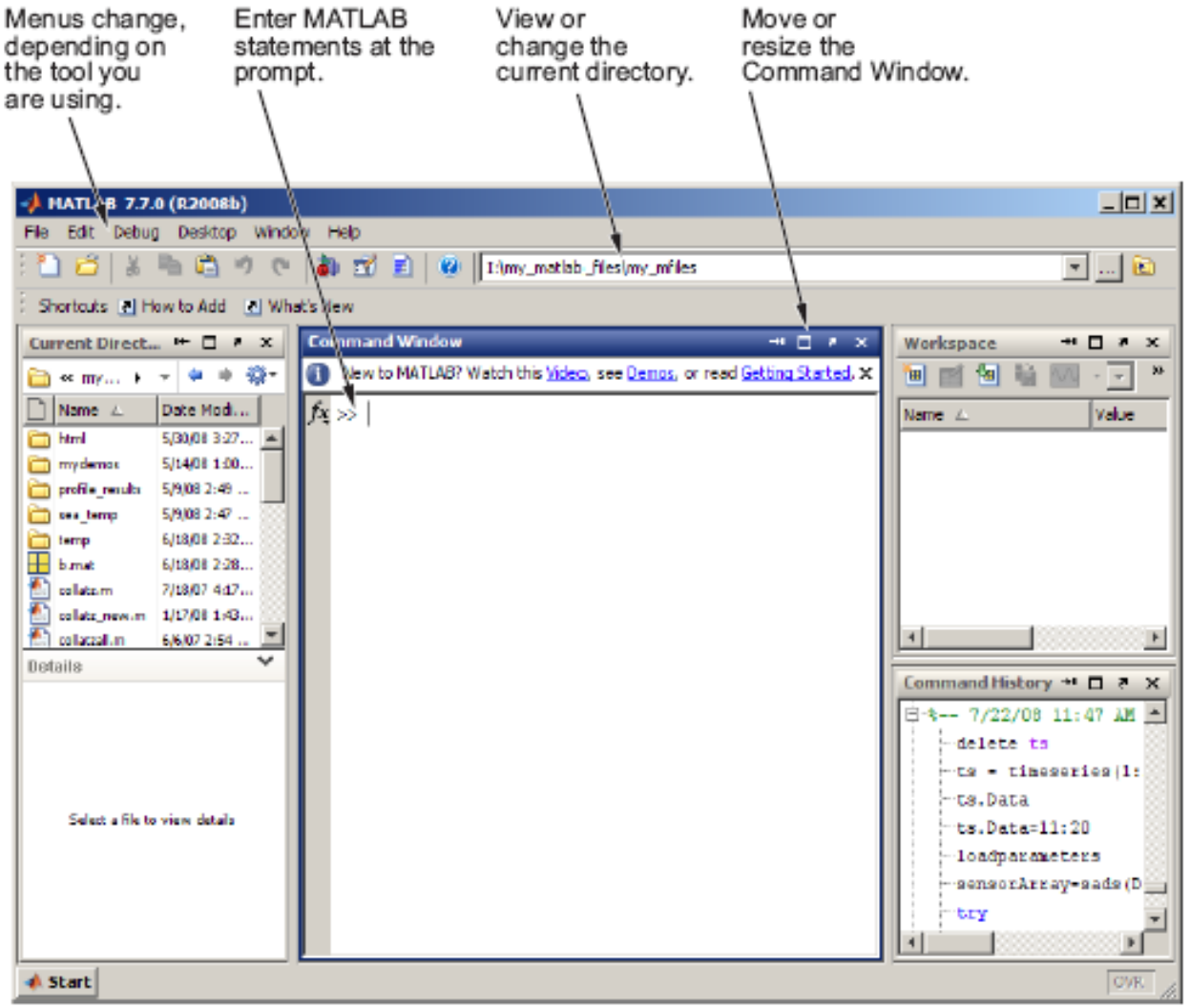
- **Desktop Tools and Development Environment**
  - Includes the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, browsers for viewing help, the workspace, files, and other tools
- **Mathematical Function Library**
  - vast collection of computational algorithms ranging from elementary functions, like sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

- **The Language**
  - The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.
- **Graphics**
  - MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as editing and printing these graphs. It also includes functions that allow you to customize the appearance of graphics as well as build complete graphical user interfaces on your MATLAB applications.
- **External Interfaces**
  - The external interfaces library allows you to write C and Fortran programs that interact with MATLAB.

Simulink

- Used to model, analyze and simulate dynamic systems using block diagrams.
- Fully integrated with MATLAB , easy and fast to learn and flexible.
- It has comprehensive block library which can be used to simulate linear, non–linear or discrete systems – excellent research tools.
- C codes can be generated from Simulink models for embedded applications and rapid prototyping of control systems.

Main Matlab Window



Working with Matrices and Arrays

- Since Matlab makes extensive use of matrices, the best way for you to get started with MATLAB is to learn how to handle matrices.
  - Separate the elements of a row with blanks or commas.
  - Use a semicolon ; to indicate the end of each row.
  - Surround the entire list of elements with square brackets, [ ].

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

- MATLAB displays the matrix you just entered:

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

- Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can simply refer to it as A.

- Keep in mind, variable names are case-sensitive

- When you do not specify an output variable, MATLAB uses the variable *ans*, short for *answer,* to store the results of a calculation.
- **Subscripts**

The element in row i and column j of A

is given by A(i,j).

So to compute the sum of the elements in the fourth column of A, we have:

A(1,4) + A(2,4) + A(3,4) + A(4,4)

Which produces:

ans = 34

- **The Colon Operator**
- For example: 1:10

  is a row vector containing the integers from 1 to 10:

  1 2 3 4 5 6 7 8 9 10

- To obtain non-unit spacing, specify an increment. For example: 100:-7:50 will give you

  100 93 86 79 72 65 58 51

- Subscript expressions involving colons refer to portions of a matrix. For example: A(1:k,j)

  refers to the first k elements of the jth column of A.

- **Numbers**

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify the power. Imaginary numbers use either i or j as a suffix. Examples of legal numbers are:

3	-99	0.0001
9.6397238	1.60210e-20	6.02252e23
1i	-3.14159j	3e5i

MATLAB software stores the real and imaginary parts of a complex number.

- **The Load Function**

  The load function reads binary files containing matrices generated by earlier MATLAB sessions, or reads text files containing numeric data.

- **M-Files**

  You can create your own programs using *M-files*, which are plain text files containing MATLAB code. Use the MATLAB Editor or another text editor to create a file containing the same statements you would type at the MATLAB command line. Save the file under a name that ends in .m

- **Arrays**

  Arithmetic operations on arrays are done element by element. This means that addition and subtraction are the **same** for arrays and matrices, but that *multiplicative* operations are **different**. MATLAB uses a dot, or decimal point, as part of the notation for multiplicative array operations.

  Example: A.*A

  the result is an array containing the squares of the integers

```
ans =
   256     9     4   169
    25   100   121    64
    81    36    49   144
    16   225   196     1
```

- **Multivariate Data**

  MATLAB uses column-oriented analysis for multivariate statistical data. Each column in a data set represents a variable and each row an observation. The (i,j)th element is the ith observation of the jth variable.

  As an example, consider a data set with three variables:

  • Heart rate • Weight • Hours exercise per week

  For five observations, the resulting array might look like

- D = [72 134 3.2
  81 201 3.5
  69 156 7.1
  82 148 2.4
  75 170 1.2]

- Now you can apply MATLAB analysis functions to this data set. For example, to obtain the mean and standard deviation of each *column*, use

```
                mu = mean(D),      sigma = std(D)
mu =       75.8     161.8     3.48
sigma =    5.6303     25.499     2.2107
```

- **Entering Long Statements**

  If a statement does not fit on one line, use an ellipsis (three periods), ... , followed by **Return or Enter** to indicate that the statement continues on the next line. For example,

```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
    - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

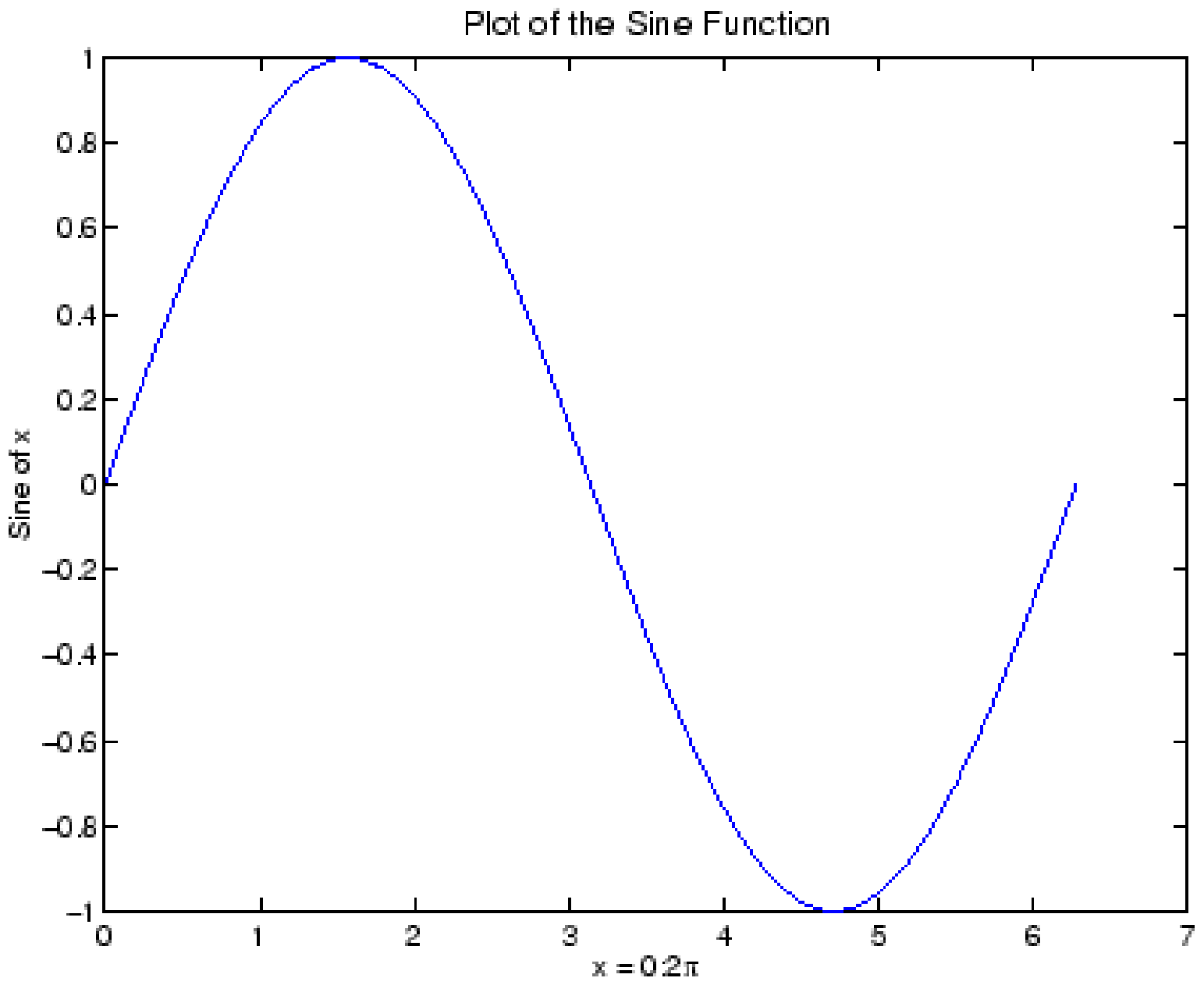
Graphics

- MATLAB provides a variety of techniques to display data graphically.
- Interactive tools enable you to manipulate graphs to achieve results that reveal the most information about your data.
- You can also edit and print graphs for presentations, or export graphs to standard graphics formats for presentation in Web browsers or other media.

Basic Plotting Functions

- The plot function has different forms, depending on the input arguments.
- If y is a vector, plot(y) produces a piecewise graph of the elements of (y) versus the index of the elements of (y).
- If you specify two vectors as arguments, plot(x,y) produces a graph of y versus x.
- You can also label the axes and add a title, using the 'xlabel', 'ylabel', and 'title' functions.

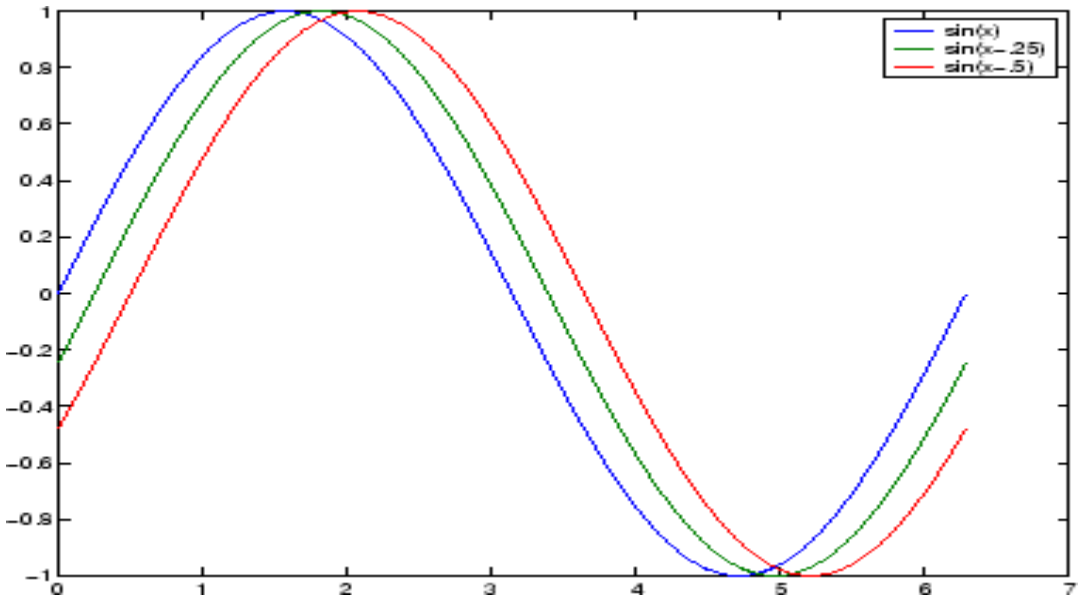
Example:
```
xlabel('x = 0:2\pi')
ylabel('Sine of x')
title('Plot of the Sine Function','FontSize',12)
```


Plot of the Sine Function
Sine of x
x = 0:2π
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7

- **Plotting Multiple Data Sets in One Graph**
  - Multiple x-y pair arguments create multiple graphs with a single call to plot.

For example:

```
x = 0:pi/100:2*pi;
y = sin(x);
y2 = sin(x-.25);
y3 = sin(x-.5);
```



- **Specifying Line Styles and Colors**

  It is possible to specify color, line styles, and markers (such as plus signs or circles) when you plot your data using the plot command:

  plot(x,y,'*color_style_marker*')

  For example: plot(x,y,'r:+')

  plots a red-dotted line and places plus sign markers at each data point.

- **Graphing Imaginary and Complex Data**

  When the arguments to plot are complex, the imaginary part is ignored *except* when you use a single complex argument.

  For example: plot(Z)

  which is equivalent to: plot(real(Z),imag(Z))

Adding Plots to an Existing Graph

When you type: hold on

MATLAB does not replace the existing graph when you issue another plotting command; it adds the new data to the current graph, rescaling the axes if necessary.

- **Figure Windows**

Graphing functions automatically open a new figure window if there are no figure windows already on the screen.

To make a figure window the current figure, type

figure(n)

where n is the number in the figure title bar. The results of subsequent graphics commands are displayed in this window.

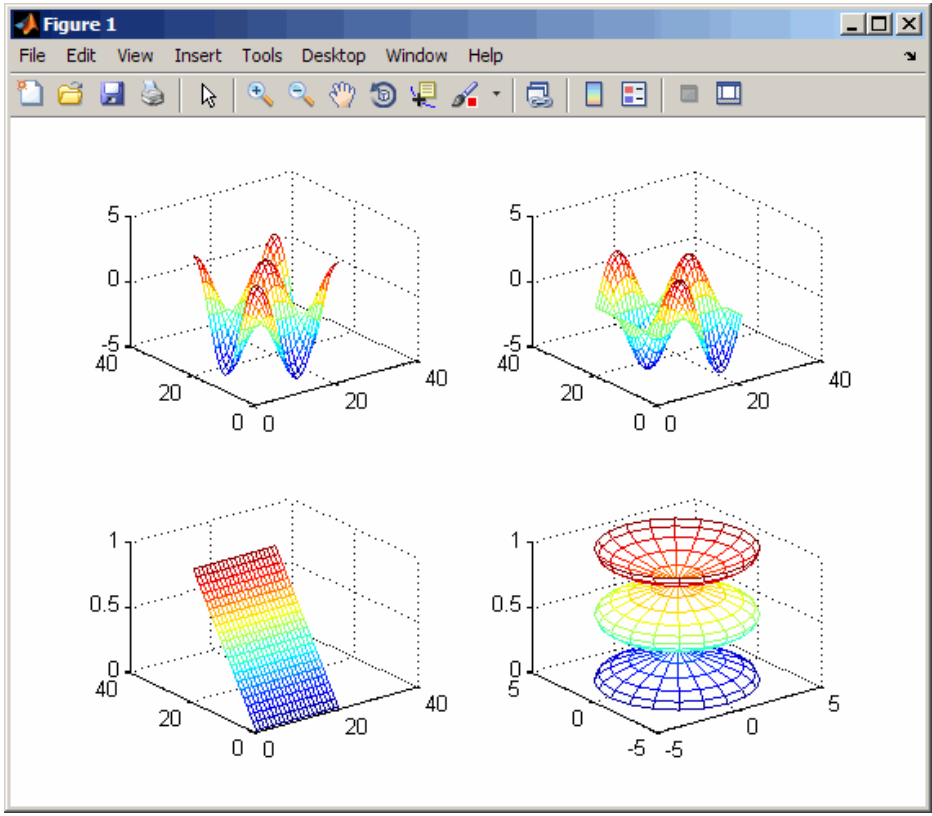
- **Displaying Multiple Plots in One Figure**

subplot(m,n,p)

This splits the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot.

- Example:

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```



Controlling the Axes

- **Setting Axis Limits & Grids**

The axis command lets you to specify your own limits:

axis([xmin xmax ymin ymax])

You can use the axis command to make the axes visible or invisible: axis on / axis off

The grid command toggles grid lines on and off:

grid on / grid off

Programming in Matlab

- Conditional Control
  - if, else, elseif
  - switch, case
- **Loop Control**
  - for, while, continue, break
- **Error Control**
  - try, catch
- **Program Termination**
  - return

Multidimensional Arrays

- One way of creating a multidimensional array is by calling zeros, ones, rand, or randn with more than two arguments.

  For example: R = randn(3,4,5);

  creates a 3-by-4-by-5 array with a total of 3*4*5 = 60 normally distributed random elements.

Scripts and Functions

- There are two kinds of M-files:

  - **Scripts**, which do not accept input arguments or return output arguments. They operate on data in the workspace. Any variables that they create remain in the workspace, to be used in subsequent computations

  - **Functions**, which can accept input arguments and return output arguments. Internal variables are local to the function.

- **Global Variables**
- If you want more than one function to share a single copy of a variable, simply declare the variable as global in all the functions. The global declaration must occur before the variable is actually used in a function.

Example:

```
function h = falling(t)
global  GRAVITY
h =  1/2*GRAVITY*t.^2;
```

Debugging

- Set breakpoints to stop the execution of code

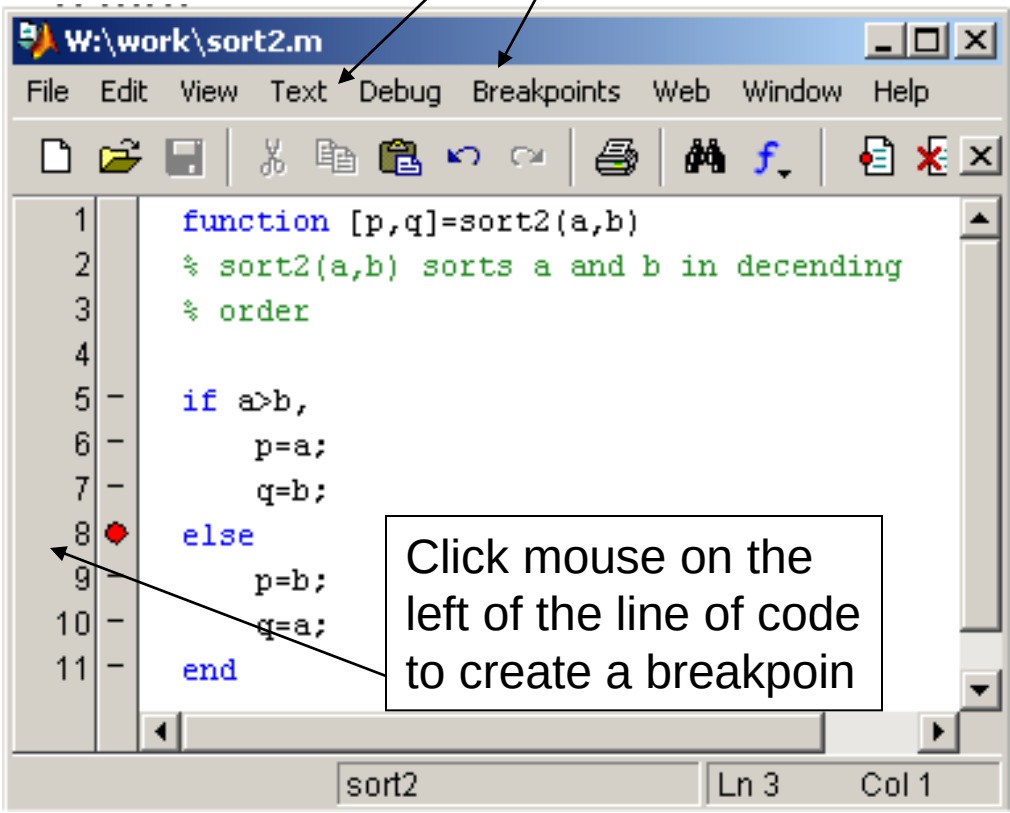
```
>> [i j]=sort2(2,4)
K>>
K>> whos
 Name      Size                Bytes  Class
  a         1x1                    8  double array
  b         1x1                    8  double array
Grand total is 2 elements using 16 bytes
K>> a
a =
     2
K>> return
i =


     4
j =
     2
```

local function workspace

exit debug mode

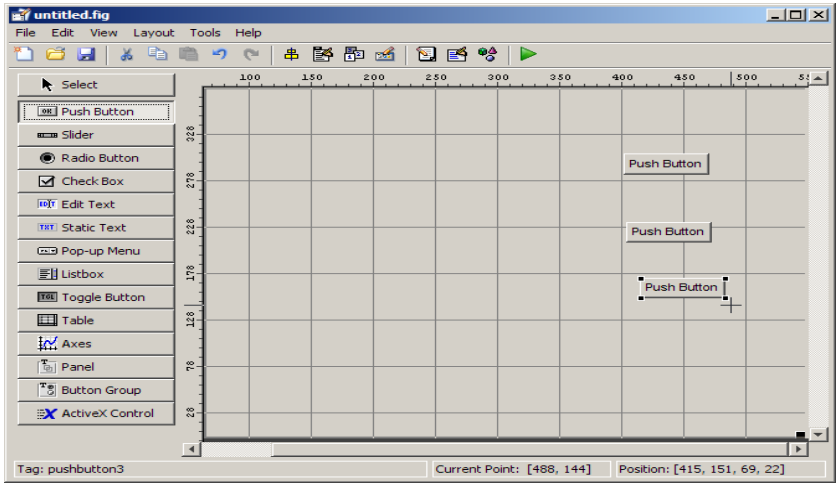
Debug menus



Click mouse on the left of the line of code to create a breakpoin

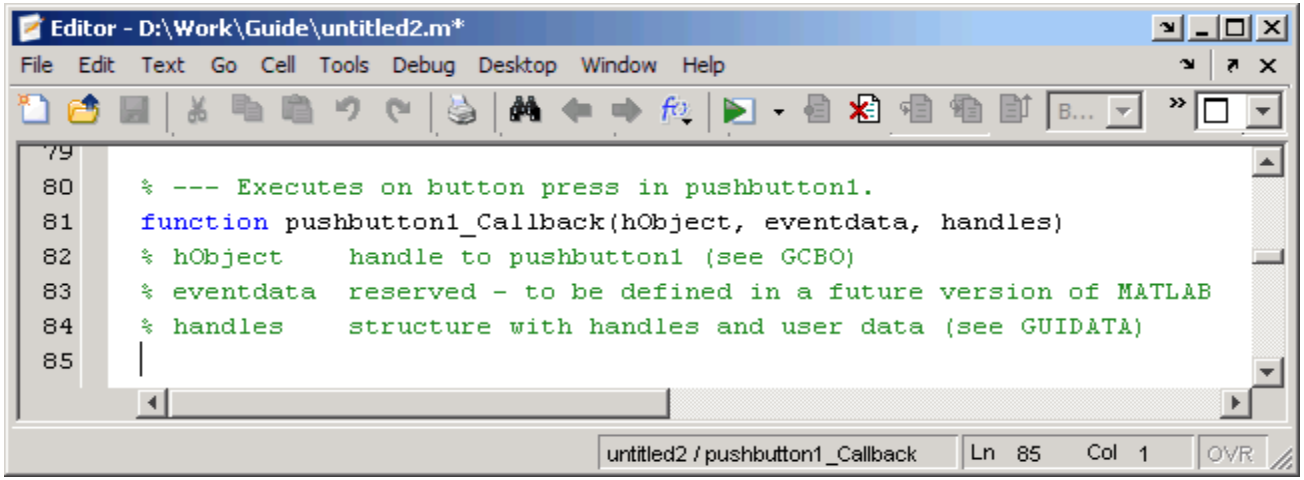
Graphical User Interfaces

- **GUIDE**, the MATLAB **G**raphical **U**ser **I**nterface **D**evelopment **E**nvironment, provides a set of tools for creating graphical user interfaces (GUIs). These tools greatly simplify the process of designing and building GUIs. You can use the GUIDE tools to perform the following tasks:
  - Laying out the GUI.



- Programming the GUI.

Example template for a push button



Resources

- Books
  - Two that seem to be good:
    - A Concise Introduction to MATLAB by William J. Palm III. 2008, McGraw-Hill, 418 pp.
    - Essential MATLAB for Engineers and Scientists (Fourth Edition) by Brian Hahn and Dan Valentine. 2010, Academic Press, 480 pp.
- Online tutorials and examples are everywhere
- Note that older books and tutorials may have options that are no longer available and functions that no longer work
  - Figures are the prime example