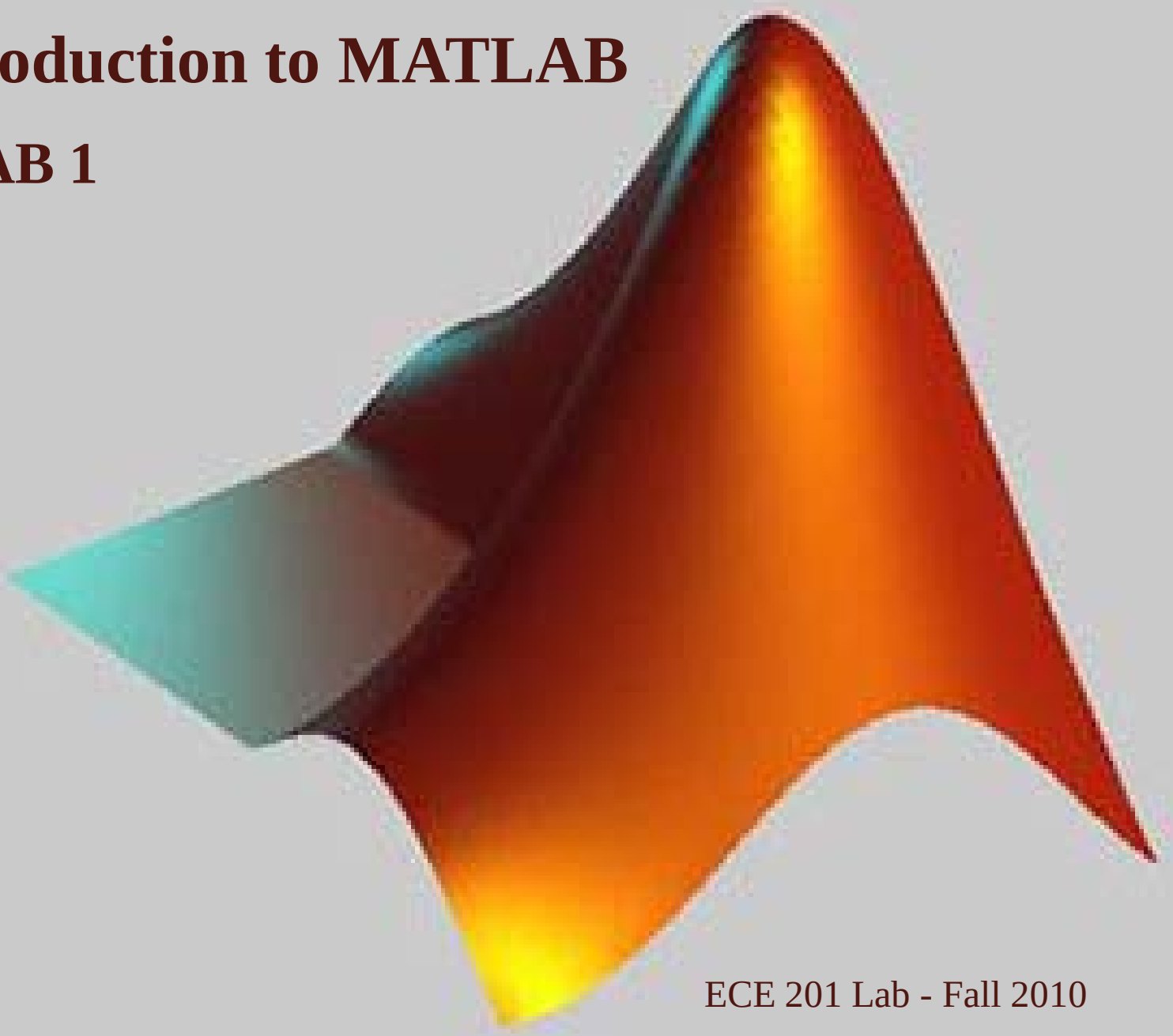
Introduction to MATLAB

LAB 1



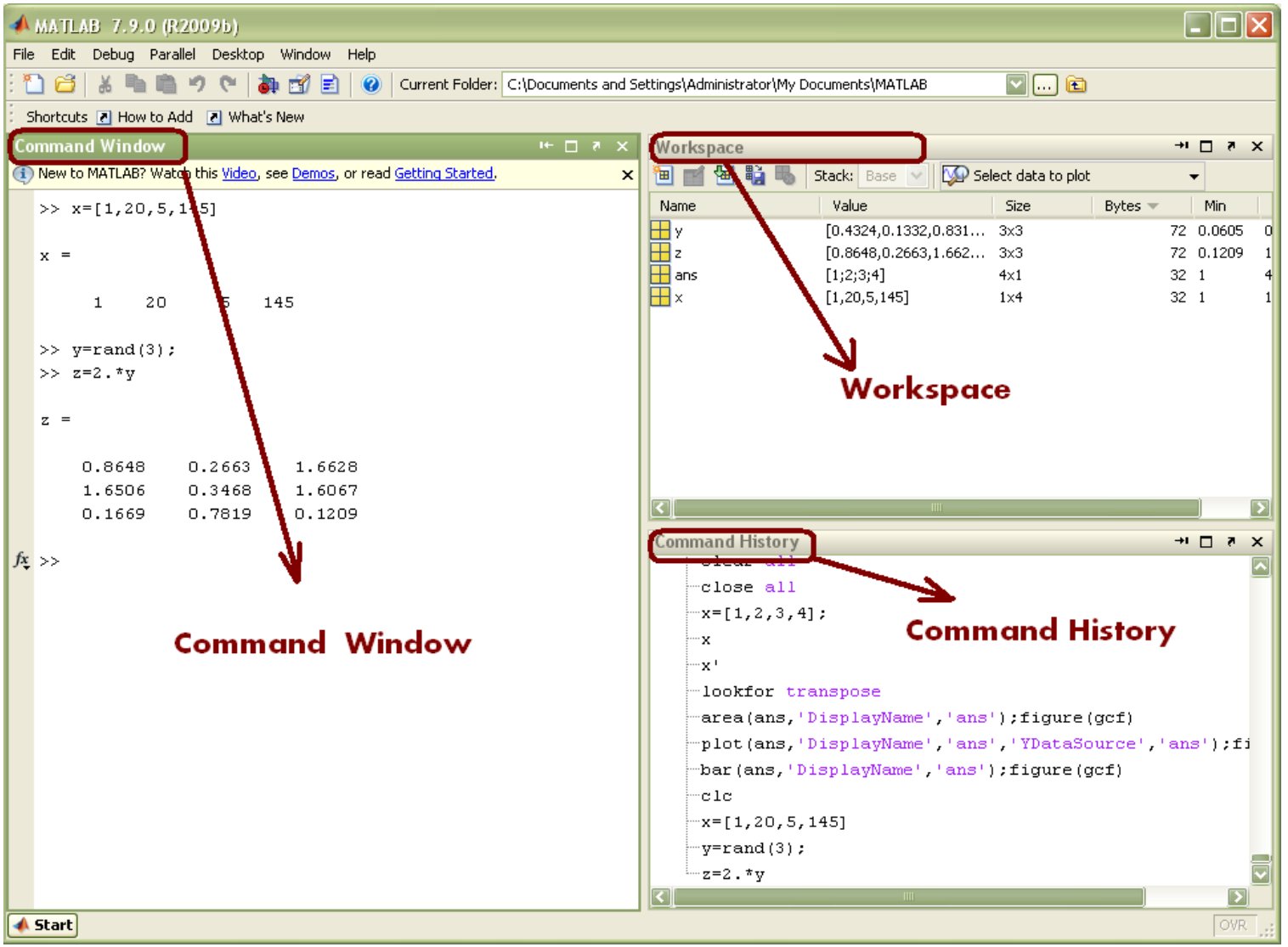
Basics of MATLAB

- MATrix LABoratory
- A super-powerful graphing calculator
- Matrix based numeric computation
- Embedded Functions
- Also a programming language
- User defined functions
- Commands executed line by line

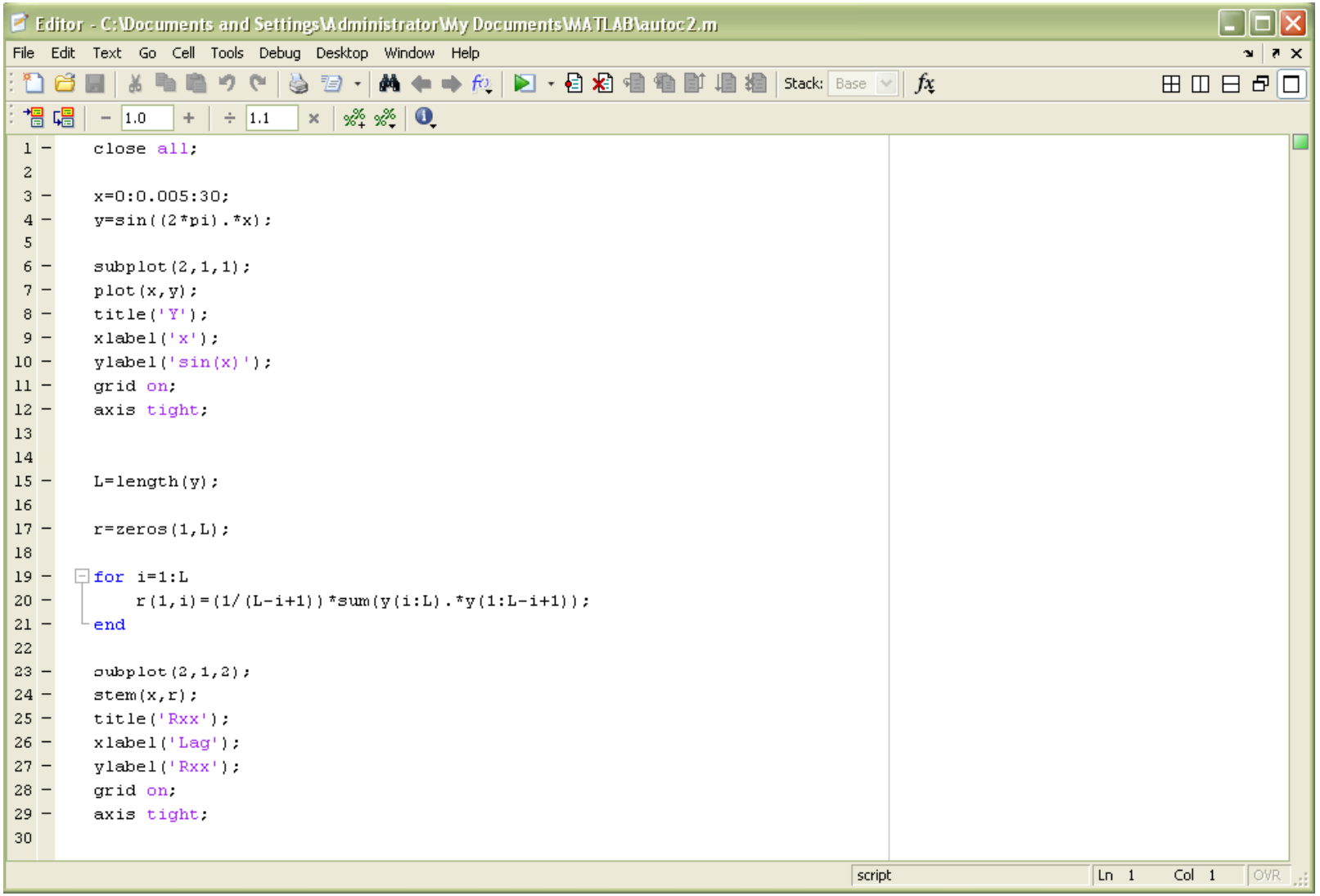
Outline

- LAB 1: User Interface, Vectors and Matrices
- LAB 2: Visualization
- LAB 3: MATLAB Scripts
- LAB 4: MATLAB Functions

MATLAB Main Window



MATLAB Script Window



MATLAB's Built-In Help System

- MATLAB Help
  - Includes detailed documentation
  - Invoke by typing '`doc`' at command line
- Search commands by keyword
  - Type '`lookfor` *keyword*' at command line
- Lookup syntax of command
  - Type '`help` *functionname*' at command line

Interacting in the command window

- Type the commands at the command prompt (>>) in the command window
- Type ‘ clc ’ to clear all the previous commands and ‘ clear all ‘ to delete existing variables in the workspace
- Type ‘who’ or ‘whos’ to get information about the variables
- Semicolon (;) at the end of a command stops the results from being displayed
- Variable ‘ ans’ stores the last unassigned value (like a calculator)
- Example:

```
>> 53
   ans=
   53
>> x=53;
```

```
>> x=53
   x=
   53
```

Creating Row Vectors

- **Row Vector**: comma or space separated values between brackets

```
>> row=[20 1.5 -33 0];
>> row=[20,1.5,-33,0];
```

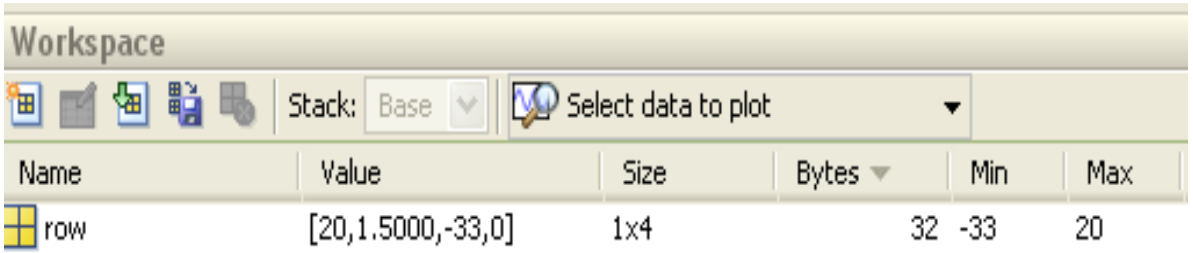
- **Command Window:**

```
>> row=[20 1.5 -33 0]

row =

   20.0000    1.5000  -33.0000         0
```

- **Workspace:**



Creating Column Vectors

- **Column Vector**: semicolon separated values between brackets

```
>> Column=[20;1.5;-33;0];
```

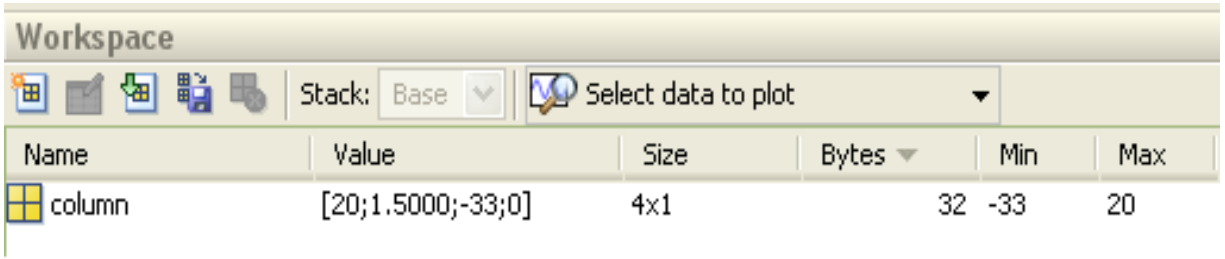
- **Command Window:**

```
>> column=[20;1.5;-33;0]

column =

   20.0000
    1.5000
  -33.0000
         0
```

- **Workspace:**



Creating Matrices

- Create Matrices like vectors

- Element by element:

```
mat=[5 3 -10;6 1.2 8];
```

- Large uniform matrices:

  - x=zeros(m,n); m: no. of rows n: no. of columns
  - y=ones(m,n);
  - z=rand(m,n); or z=rand(n);

- Vectors with values increasing linearly:

  - t= Start : Increment : End;
  - t= linspace (Start, End, Number of Samples);

Dimensions of Matrices

- Number of rows and columns:
  - [row_x , column_x] = size(x);
  - row_x = size (x,1);
  - column_x = size (x,2);
- Length of a matrix;
  - length_x = length (x);
- Number of elements in a matrix:
  - num_x = numel (x);

Accessing Elements Inside a Matrix

- Accessing by “row number” and “column number”
  - a = x (m , n);
  - x (m , n) = 3;
- Accessing by “element number”
  - a = x (*element_number*);
- Accessing an entire row:
  - r = x (m , :);
  - x (m , :) = [a b c d];
- Accessing part of a column
  - c = x (2:4 , n);
  - x (2:4 , 1) = [3 ; 4 ; 5];
- Be careful about dimensions!

Searching Inside a Matrix

- Use the “find” command to return the “element no.”
  - el_number = find (x == 3);
- Use the “find” command to find the row and column no.
  - [row_no , column_no] = find (x == 3);
- **NOTE** the “ == “ sign!

Mathematical Operations

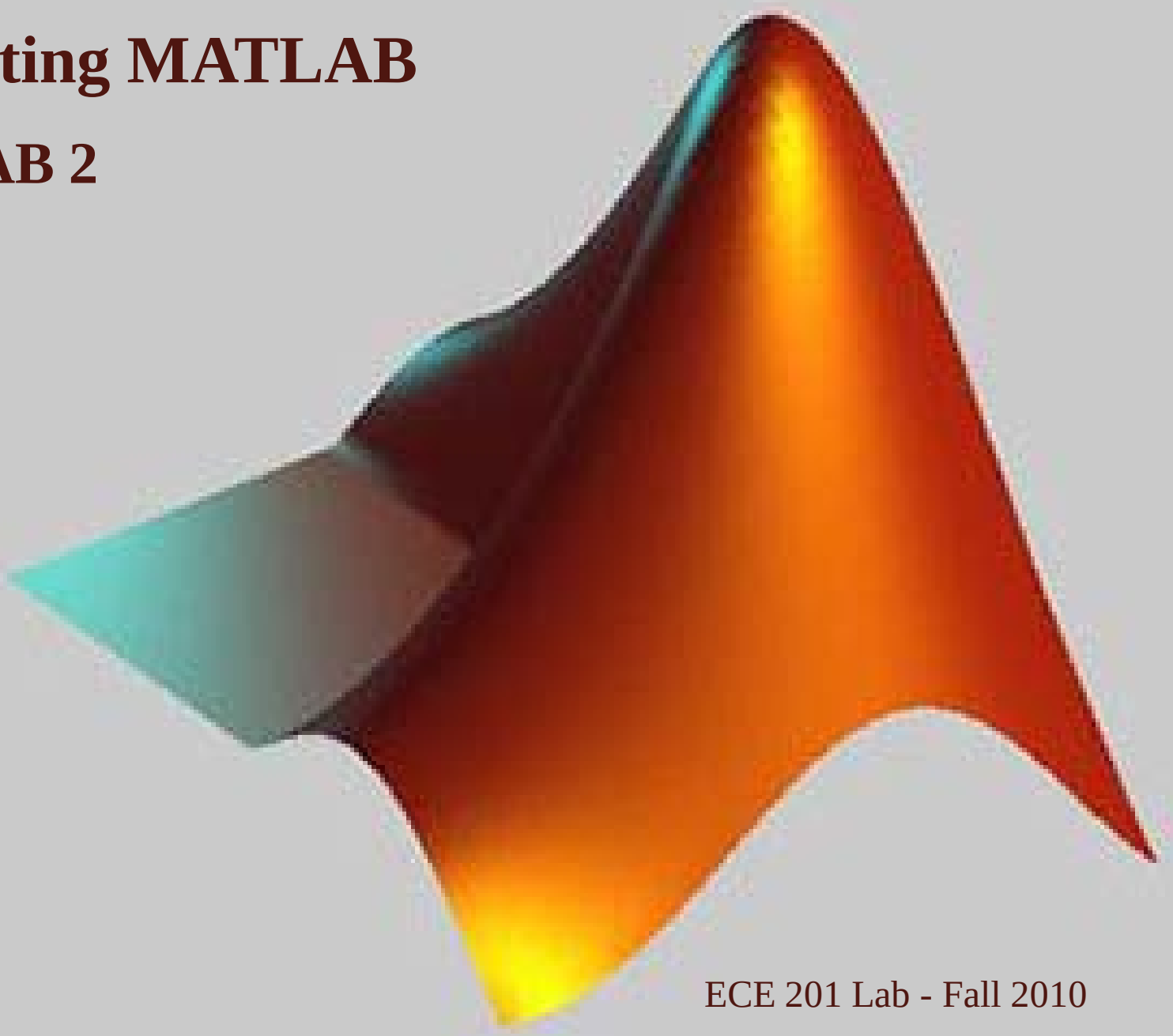
- Matrix Multiplication
  - `a = b * c ;` multiplies two matrices
  - `a = 3 * b ;` multiplies a scalar with a matrix
- Element by element Operations
  - `a = b .* c ;`
  - `a = b .^ c ;`
- Transpose operator turns a column vector to a row vector or vice versa:
  - `b = transpose (a);`
  - `b = a' ;`
- Take note of the dimensions!!

LAB 1 – Introduction to MATLAB

- Before starting the lab type " >> diary Lab1 " at command prompt
- After finishing everything type " >>diary off "
- Find the file " Lab1" in the current directory and open with any word processor
- Edit the file and submit it for Lab1 report at the beginning of next class

Plotting MATLAB

LAB 2



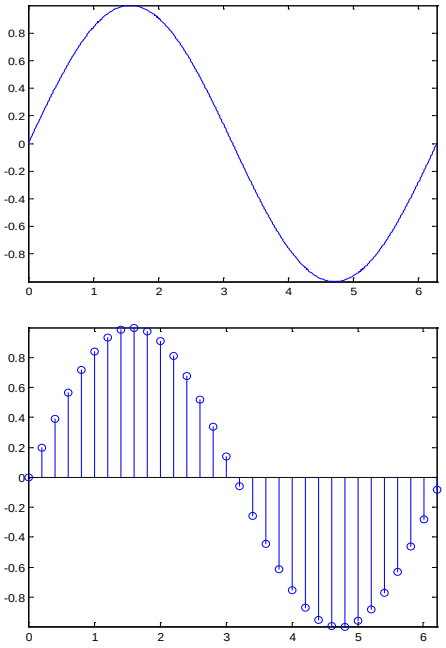
2-D Plots in MATLAB

- Different types of 2-D Plots:
  - Basic line plots:

    >> plot(x ,y)

  - Stem Plots:

    >> stem(x,y)



- Other types:

  Bar, Area, Stairs, Loglog etc.

Generating Functions/Signals

- Create independent vector first:

```
>>  x=0:0.1:5;
>>  w=linspace(0,2*pi,100);
```

- Create dependent vector based on the independent vector:

```
>>  y=exp(x+ 2);
>>  V= 4.*cos(w);
```

- The length of x and w will be the same as y and V respectively.

The *plot* Command

- Plotting a single variable:
  - x : row vector of numbers
  - `plot(x)` : plots the values of x against the element number
  - Element number is on the horizontal axis, and the values on the vertical axis
- Plotting two variables against each other
  - x and y : row vectors of the same size
  - `plot(x,y)`: plots the values of y against the corresponding values of x
  - x is displayed on the horizontal axis and y on the vertical axis

Plot Options

- Change the line color , marker style and line style by adding a string argument:

```
>> plot(x, y, 'r*--') ;
```

  - -- changes the line style
  - r changes the color
  - * changes the marker style

- Plot without connecting the dots by omitting line style argument

- Default: blue continuous line with no markers

- Find the appropriate characters by typing 'help plot' in the command window

Figure Properties

- Changing axis limits

```
>> axis ([x1 x2 y1 y2]);
>> xlim ([x1 x2]);
>> ylim ([y1 y2]);
```

- Putting labels

```
>>  xlabel('string');
>>  ylabel('string');
```

- Displaying a text inside a plot

```
>>  text (x,y,'string');
```

Figure Properties - continued

- Creating a plot's title:

```
>> title('string');
>> title(['sting1', num2str(n),'string2']);
```

- Displaying a grid:

```
>> grid on;
```

- Setting the axis limits to the range of the data:

```
>> axis tight;
```

Figure Windows

- Plotting multiple functions inside the same window:
  - After plotting the first function type 'hold on'
  - The previous plot is not erased

- Creating a new figure:

>> figure(2) : will open a new figure window, titled 'Figure 2'. Next commands will affect this window.

- Closing all the open figure windows:

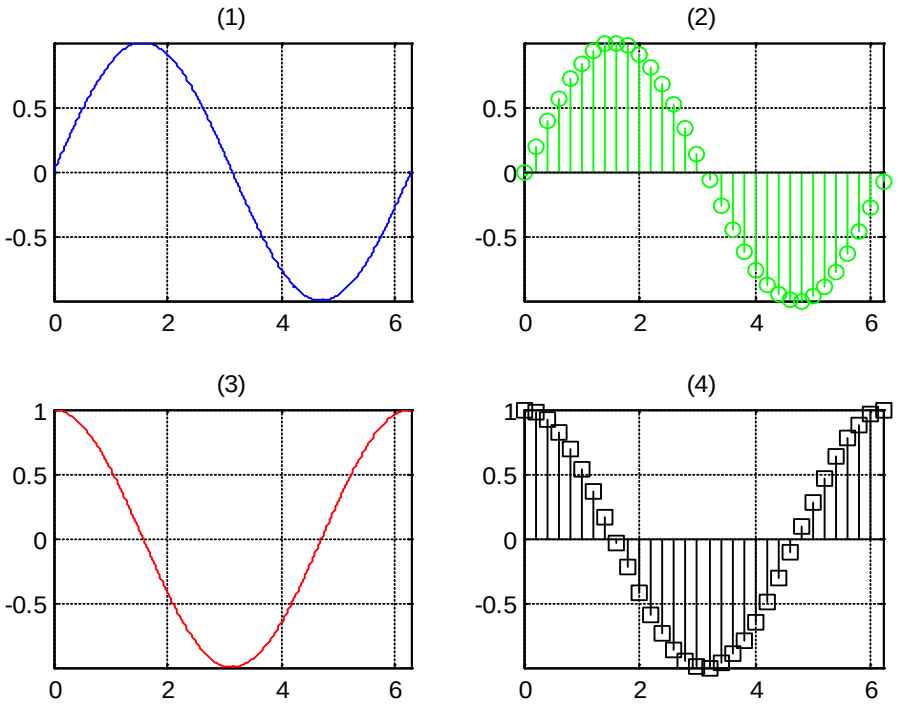
```
>> close all;
```

Multiple Plots in One Figure

- Creating multiple plots inside the same figure window

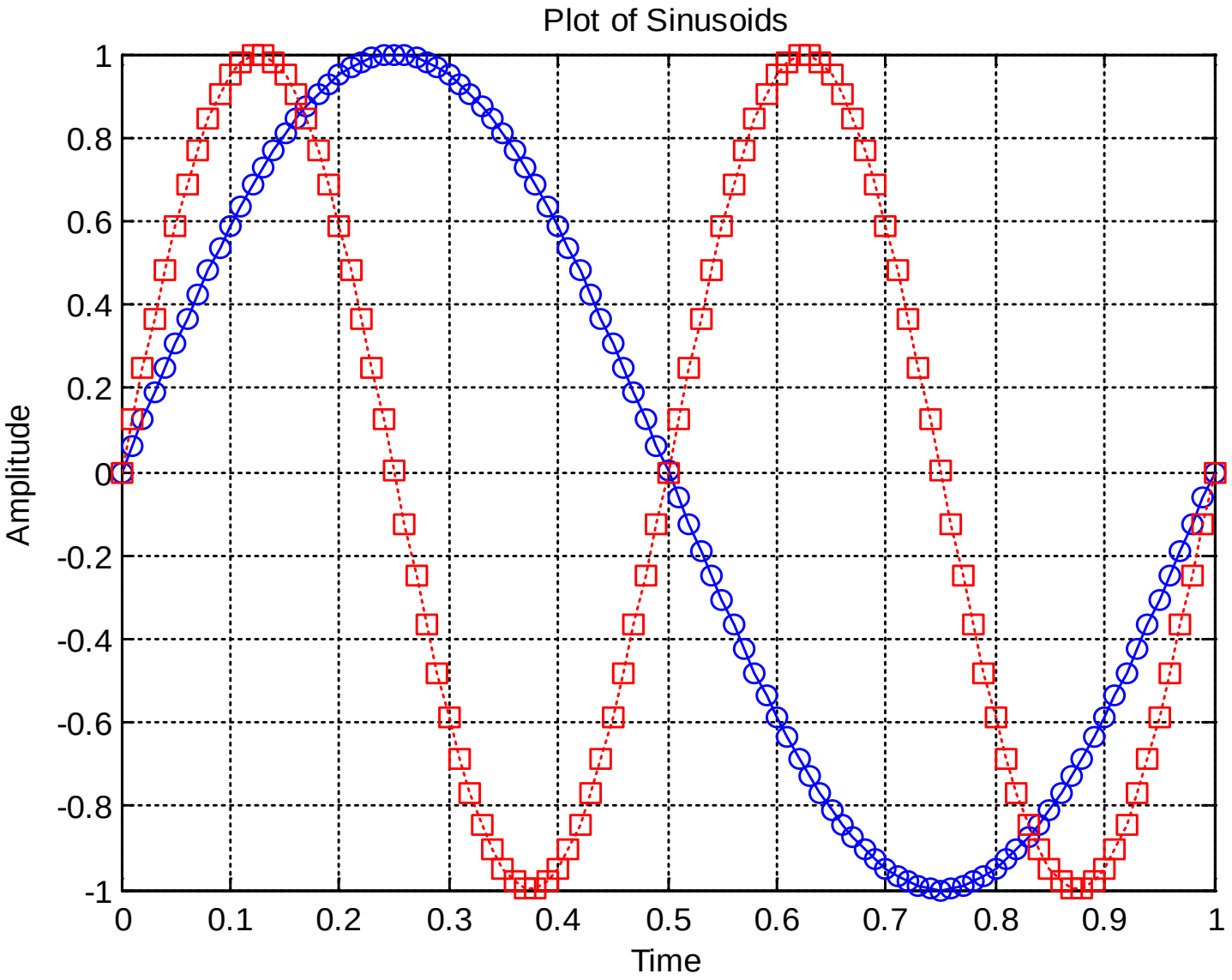
```
>> subplot (m,n,p);
```

-m: number of rows
-n: number of columns
-p: plot number



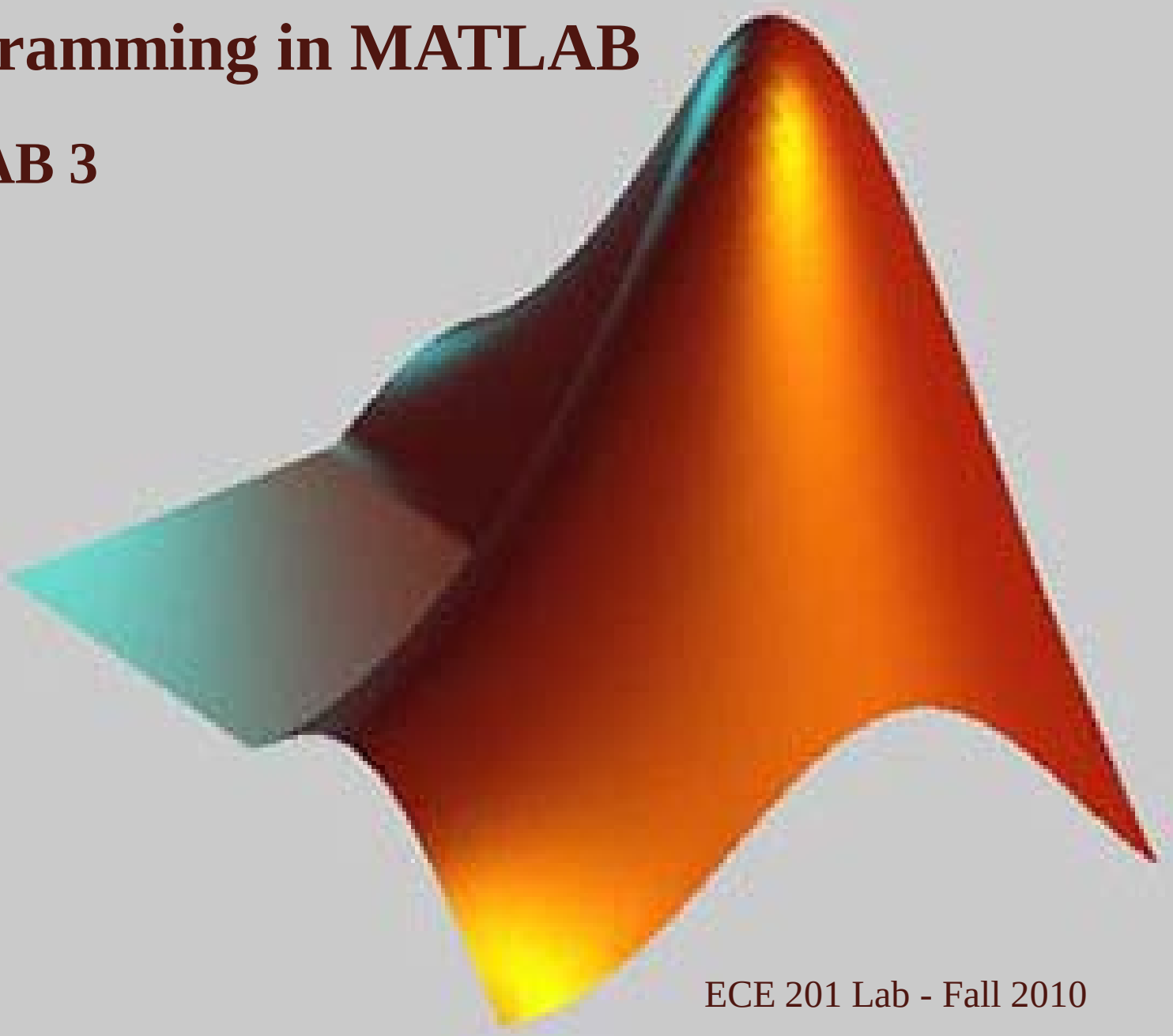
Example

```
>>  t = 0:0.01:1;
>>  x = sin (2*pi*t);
>>  y = sin (4*pi*t);
>>  figure (1);
>>  plot (t , x , '- b o');
>>  hold on;
>>  plot (t , y , ': r s' );
>>  grid on;
>>  xlabel ('Time');
>>  ylabel ('Amplitude');
>>  title (' Plot of Sinusoids');
```


Plot of Sinusoids
Amplitude
Time
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Programming in MATLAB

LAB 3



Creating a Script

- Scripts are
  - Written in MATLAB editor: *File > New > M-File*
  - Saved as MATLAB files (.m extension)
  - Executed line by line
- Scripts / Command line
  - All the commands written and then executed at once
  - Later editing possible on saved M-files
  - Adding comments to the program body (%)

input **and** disp **command**

- input: asks the user for an input and stores it in a variable

```
>> n = input ('Enter an integer number:');
```

- disp: displays a text

```
>> disp (['The number you entered is: ' , num2str(n)]);

>> disp ( 'End of Program');
```

if **Statement**

- **if :** execute statements if condition is true
- **elseif :** execute statement if additional condition is true
- **else :** execute statement if condition is false

```
>> if  a==b
            disp ( ‘ a equals b’ );
    elseif  a > b
            disp ( ‘ a greater than b’ );
    else
            disp ( ‘ a less than b’ );
    end
```

if **statement with multiple conditions**

- Conditions:

```
>> if   A == B
>> if   A ~ = B
>> if  A > = B
>> if  A < = B
```

- Combined conditions:

```
>> if  condition1 & condition2
>> if  condition1 ||  condition2
```

Loops

- For: executes a block of codes for a specified number of times

```
>>  for n = 1:10
      disp (['the value of n is' , num2str(n)]);
      end
```

- While: repeatedly executes a block of codes while the condition is true

```
>>   while  k<=10
       disp(k);
       k=k+ 1;
       end
```

Complex Numbers in MATLAB

- Generating complex numbers

```
>> c1 = complex ( real_part , im_part);
>> c2 = real_part +  i * im_part;
>> c3 = real part +  im_part i;
```

- Complex math:

```
>>  R=real(X);
>>  I=imag(X);
```

- Other commands – type help complex and look at the related commands

Plotting complex numbers

- Plot command

```
>> plot (z1, 'rs' );  -plots the point with a red square
```

- MATLAB plots the complex numbers in the complex plane
- Real part is displayed on the horizontal axis and imaginary part on the vertical axis

Working with .mat files

- Save variables in the workspace to a .mat file
  - Saving the entire workspace

```
>> save filename.mat
```

  - Saving some of the variables

```
>> save filename.mat x y z
```

- Load variables from .mat file to the workspace
  - Loading all the contents of the file to the workspace

```
>> load filename.mat
```

  - Loading some of the variables

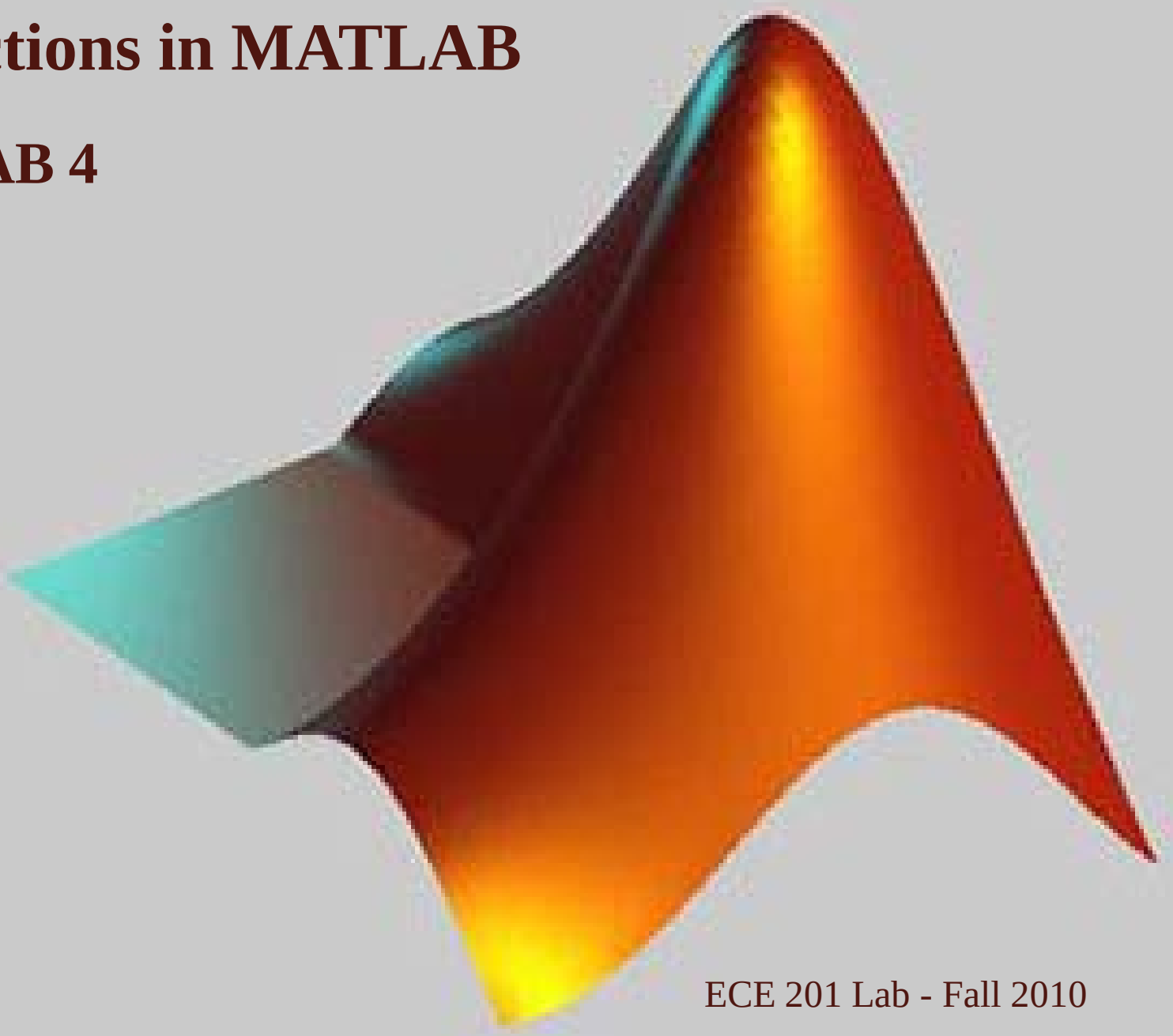
```
>> load filename.mat x y z
```

Example

```
>>   a = 9 +  2i;
>>   r = real (a);
>> while   r > 3
            r = r – 1;
      end
>>  b = r +  i*(imag(a)*3);
>>  if  imag(b) < 7 & real(b) = = 3
     disp( [‘ b = ‘ , num2str (real (b)) , ‘+  i* ‘ , num2str(imag(b))]) ;
     end
```

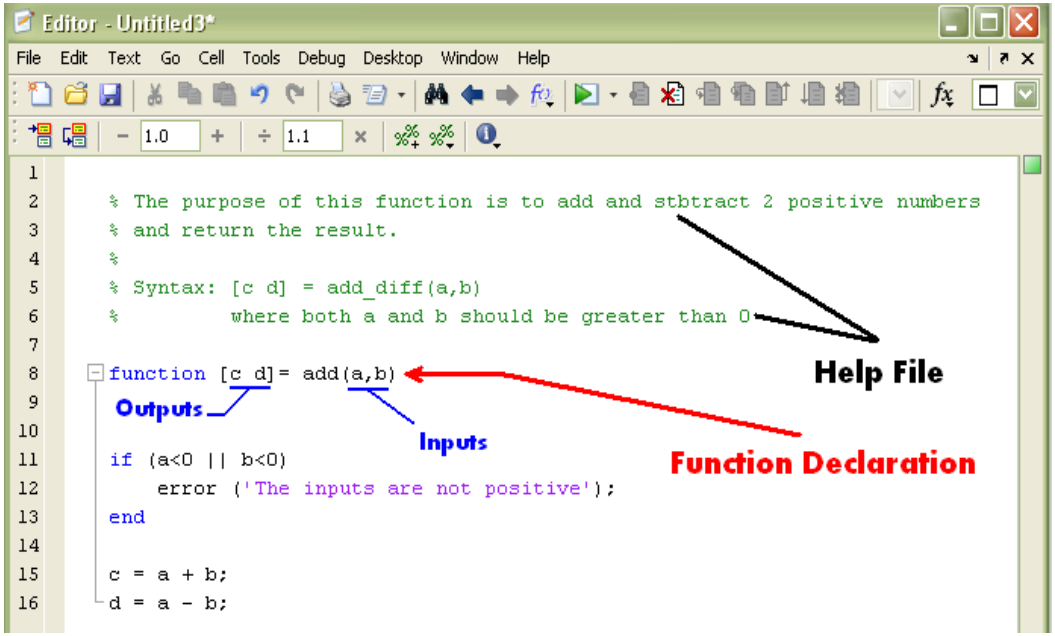
Functions in MATLAB

LAB 4



User-defined Functions

- The only difference with scripts: function declaration
- Advantages:
  - Repeatable operations written as functions
  - The actual code becomes cleaner



Basic Structure of a Function

- Help file: comments at the beginning of the M-file or after the function declaration (%)
- Function declaration:

function [out1, out2, …] = function_name (in1 , in2, …)

- Check for errors in input variables
- Commands to find the output variables

Example – Single Output Function

```
%  The purpose of this function is to add 2 positive numbers and return
%   the result
%   Syntax:       c  =  add (a , b)
%                    where both  a  and  b  should be greater than 0.

function  [ c ]  =  add  ( a , b )

if    ( a < 0  ||  b < 0  )
       error  ( ' The inputs are not positive ' );
end

c = a +  b;
```

- Calling the function (either in command window or in another M-file)

```
>>   x =  add ( 5 , 4 );
```

Example – Multiple Output Function

```
%  The purpose of this function is to add and subtract  2  positive
%   numbers and return both results.
%   Syntax :       [ c ,  d ]  =  add  ( a  ,  b )
%                    where both   a  and  b  should be greater than 0.

function [c, d] = add_diff (a, b)

if   ( a < 0  ||  b < 0 )
     error ( 'The inputs are not positive' );
end

c = a +  b;
d = a – b;
```

- Calling the function (either in command window or in another m-file

```
>>  [ x, y ] = add_diff  ( 5, 4 );
```

Example – Working with Vectors

```
% The purpose of this function is to accept a vector as an input,
% multiply it with a linear vector and output the result.
%  Syntax :    Y = multiply ( X, x1, x2 )
%              where both X and Y are vectors, x1 and x2 are start
%              and end of the linear vector.

function [ Y ] = multiply ( X, x1, x2 )

if  ( x2 < x1 )
    error ( 'The "end" value is less than the "start" value' );
end

N = length (X);
XX = linspace ( x1, x2, N);
Y = X .* XX;
```

Remember!

- Name of the M-file: the same as the function name
- Save the function M – file in the current directory
- No need to return the output variable - just ensure that the variable names are the same
- In most cases, no need to an input command, only input variables
- Do not "run" the function, "call" it!
- The "call" command: the same as the "function definition" line