# Lecture 4

# Lecture 4: Produce and Customize Simple Plots

- **The Plot Function**
  - Example 1

```
>> x = 1:6;
>> y = [1  5  3  9  11  8];
>> plot(x,y)
```

  - Example 2

plotonepoint.m

```
% This is a really simple plot of just one point

clc
clear

% Create coordinate variables and plot a red '*'
x = 11;
y = 48;
plot(x, y, 'r*')
% Change the axes and label theme
axis([9  12  35  55])      % define the range of x and y
xlabel('Time')
ylabel('Temperature')
% Put a title on the plot
title('Time and Temp')
```

**The number of elements in vector storing x coordinates must be the same as that for the y coordinates.**

**If only one vector is passed to plot function, the values will be used as the coordinates of y axis**

**The variable names do not have to be x and y.**

**The plot can be created either in the command window or with a script**

# Lecture 4: Produce and Customize Simple Plots

## ➢ Customizing a Plot: Color, Line Types, Marker Types

- Possible colors

| | |
|---|---|
| b | blue |
| c | cyan |
| g | green |
| k | black |
| m | magenta |
| r | red |
| y | yellow |

- Line types

| | |
|---|---|
| - - | dashed |
| - . | dash dot |
| : | dotted |
| - | solid |

- Plot symbols

| | |
|---|---|
| o | circle |
| d | diamond |
| h | hexagram |
| p | pentagram |
| + | plus |
| . | point |
| s | square |
| * | star |
| v | down triangle |
| < | left triangle |
| > | right triangle |
| ^ | up triangle |
| x | x-mark |

- If no line type is specified, a solid line is drawn between the points

# Lecture 4: Produce and Customize Simple Plots

## ➢ Plot Functions

- figure: creates a new, empty Figure Window when called without any arguments. Calling it as figure(n) where n is an integer is a way of creating and maintaining multiple Figure Windows, and of referring to each individually.
- Subplot: create axes in tiled positions. subplot(m,n,p) breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc.
- hold: is a toggle that freezes the current graph in the Figure Window, so that new plots will be superimposed on the current one. Just hold by itself is a toggle, so calling this function once turns the hold on, and then the next time turns it off.
- legend: displays strings passed to it in a legend box in the Figure Window.
- grid: displays grid lines on a graph. Called by itself, it is a toggle that turns the grid lines on and off. Alternatively, the commands grid on and grid off can be used.

# Lecture 4: Produce and Customize Simple Plots

- **Plot Functions**
  - Example

plot2figs.m

```
% This creates 2 different plots in 2 different Figure Windows, to demonstrate
some plot features

clc
clear

x = 1:5;
y1 = [2  11  6  9  3];
y2 = [4  5  8  6  2];
% Put a bar chart in Figure 1
figure(1)
bar(x, y1)
% Put plots using different y values on one plot with a legend
figure(2)
plot(x, y1, 'k')
hold on
plot(x, y2, 'ko')
grid on
legend('y1', 'y2')
```

What if we remove 'figure(1)' and 'figure (2)'

What if we remove 'hold on'?

# Lecture 4: File Input/Output

- **Introduction to File Input/Output (Load and Save)**
  - There are basically three different operations, or modes, on files:

    read from

    written to

    appended to
- **Write Data to a File**
  - The **save** function can be used to write data from a matrix to a data file, or to append to a data file. The form is,

    *save filename matrixvariablename -ascii*

    Example: *>> mymat = rand(2,3)*

    *mymat =*

    *0.4565 0.8214 0.6154*

    *0.0185 0.4447 0.7919*

    *>> save testfile.dat mymat –ascii*

    -ascii is used when creating a text or data file.

    The data structure in the file is the same as that of the matrix
  - The type command can be used to display the contents of the file.

    *>> type testfile.dat*
  - If the file already exists, the save function will overwrite it.

# Lecture 4: File Input/Output

- **Appending Data to a Data File**
  - Once a text file exists, data can be appended to it. The format is the same, with the addition of the qualifier –append.

    Example:

    *>> mymat = rand(2,3)*

    *mymat =*

    *0.4565 0.8214 0.6154*

    *0.0185 0.4447 0.7919*

    *>> save testfile.dat mymat –ascii* ***-append***

    *>> type testfile.dat*

.

# Lecture 4: File Input/Output

- **Read from a File**
  - Once a file has been created, it can be read into a matrix variable, the load function will read from the file filename.ext and create a matrix with the same name as the file.
  - Example: Load from a File

```
>> clear
>> load testfile.dat
>> who
Your variables are:
testfile
>> testfile
```

.

# Lecture 4: User Defined Functions

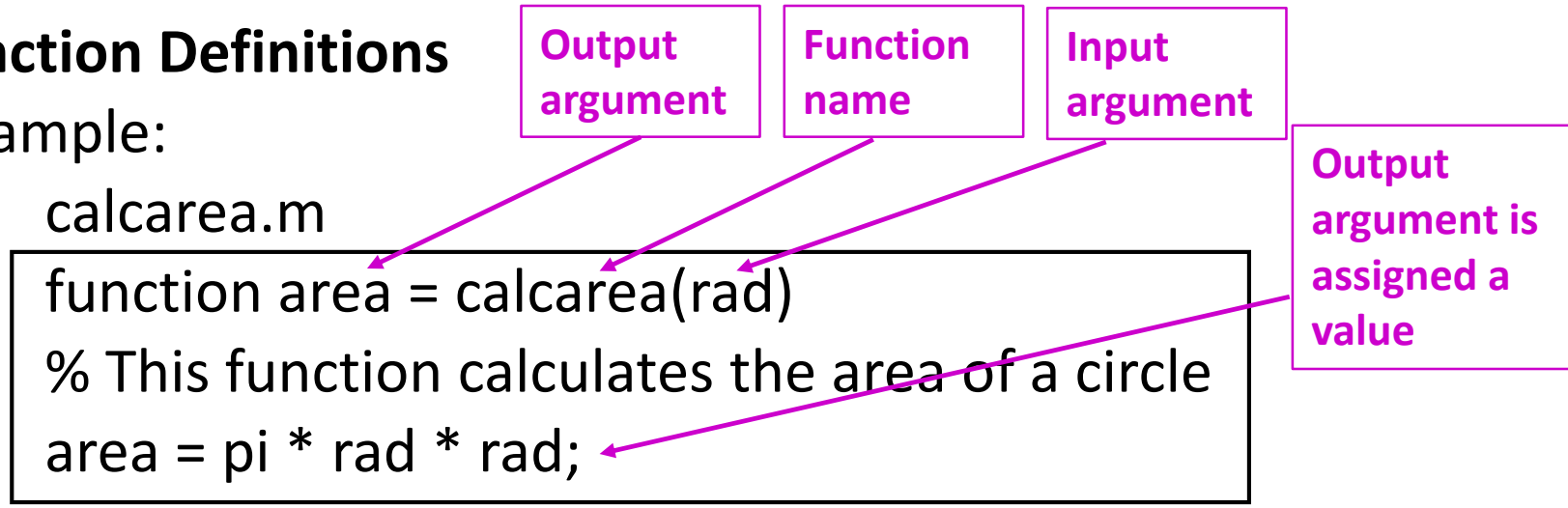
- **Function Definitions**
  - There are different ways to organize scripts and functions, but for now every function that we write will be stored in a separate script, or M-file.
  - A function in MATLAB that returns a single result consists of
    - ✓ The function head
      - The reserved word function
      - The output argument followed by the assignment operator =
      - The name of the function (important: This should be the same as the name of the M-file in which this function is stored in order to avoid confusion)
      - The input arguments in parenthesis; these correspond to the arguments that are passed to the function in the function call
    - ✓ A comment that describes what the function does (this is printed if help is used)
    - ✓ The body of the function, which includes all statements and eventually must assign a value to the output argument

# Lecture 4: User Defined Functions

➢ **Function Definitions**

Example:

calcarea.m




- The function can be displayed in the Command Window using the type command

  >> type calcarea

**No 'clc' and 'clear' in the user defined functions**

# Lecture 4: User Defined Functions

- **Calling a Function**
  - Technically, calling the function is done with the name of the file in which the function resides. In order to avoid confusion, it is easiest to give the function the same name as the filename.

    Examples:

```
>> calcarea(4)
ans =
    50.2655

>> area = calcarea(5)
area =
    78.5398
```

Here the variable name 'area' does not have to be the same as the name of the output argument of the function.

.

# Lecture 4: User Defined Functions

- **Calling a User-Defined Function from a Script**

  Example:

  script3.m

```
% This script calculates the area of a circle
% It prompts the user for the radius
radius = input('Please enter the radius: ');
% It then calls our function to calculate the area
% and then prints the result
area = calcarea(radius);
fprintf('For a circle with a radius of %.2f, ', radius)
fprintf('the area is %.2f\n', area)
```

```
>> script3
Please enter the radius: 5
For a circle with a radius of 5.00, the area is 78.54
```

# Lecture 4: User-Defined Functions

- **Summary**
  - Plot functions
  - File input/output
  - User defined functions

- **Homework (on Canvas)**
  - Due next Monday;