ENMC3111
Computer Aided Engineering

Lecture #1

Ihab Abu Ajamieh, PhD.

Course details

Instructor: *Asst. Prof.* Ihab Abu Ajamieh.

Email: iabuajamieh@Birzeit.edu

Time: Thu 14:15 – 16:55 Aggad 344

Mon 14:15 – 16:55 Aggad 344

Office hours: You can email me, or the TA.

Lectures format: will include theory, examples, and H.W's

Introduction

Course description:

This course (Lab) provides you with

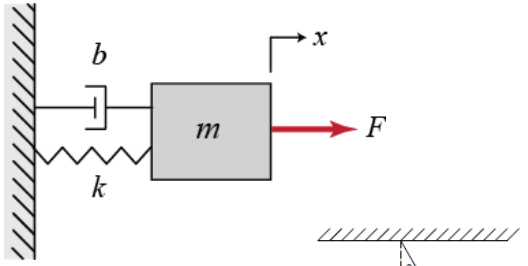
1. Programming language - MatLab:
    - MatLab introduction.
    - Matrices operations.
    - Loops and feedback systems.
    - Mini projects.
    - ⋮
2. The state-of-the-art Engineering software:
    - CAD: Solidworks: Sketching, 3D modeling, Assembly, and motion study.
    - CAM: Solidworks OR MasterCAM.
    - Numerical Analysis simulation: ANSYS or Solidworks.

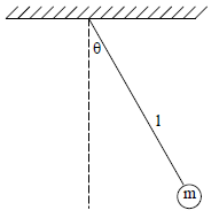
Introduction

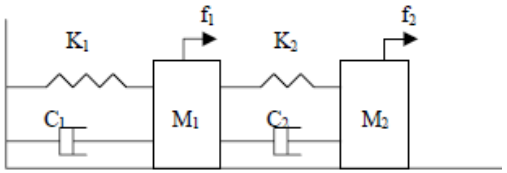
Course description:

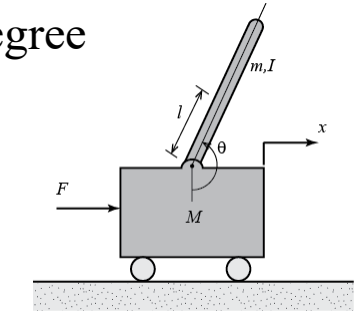
MatLab mini projects:

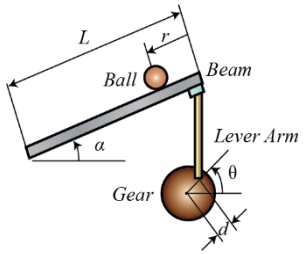
1. Spring-Mass-Damper system – free and forced.
2. Simple pendulum
3. Trajectory motion with aerodynamic drag.
4. Frequency response of systems having more than one degree of freedom.
5. Inverted pendulum control.
6. Ball and beam control.
7. Cruise control.

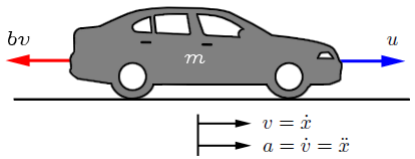












MATLAB

An Introduction

What is MATLAB?

MATLAB is a powerful software tool for:

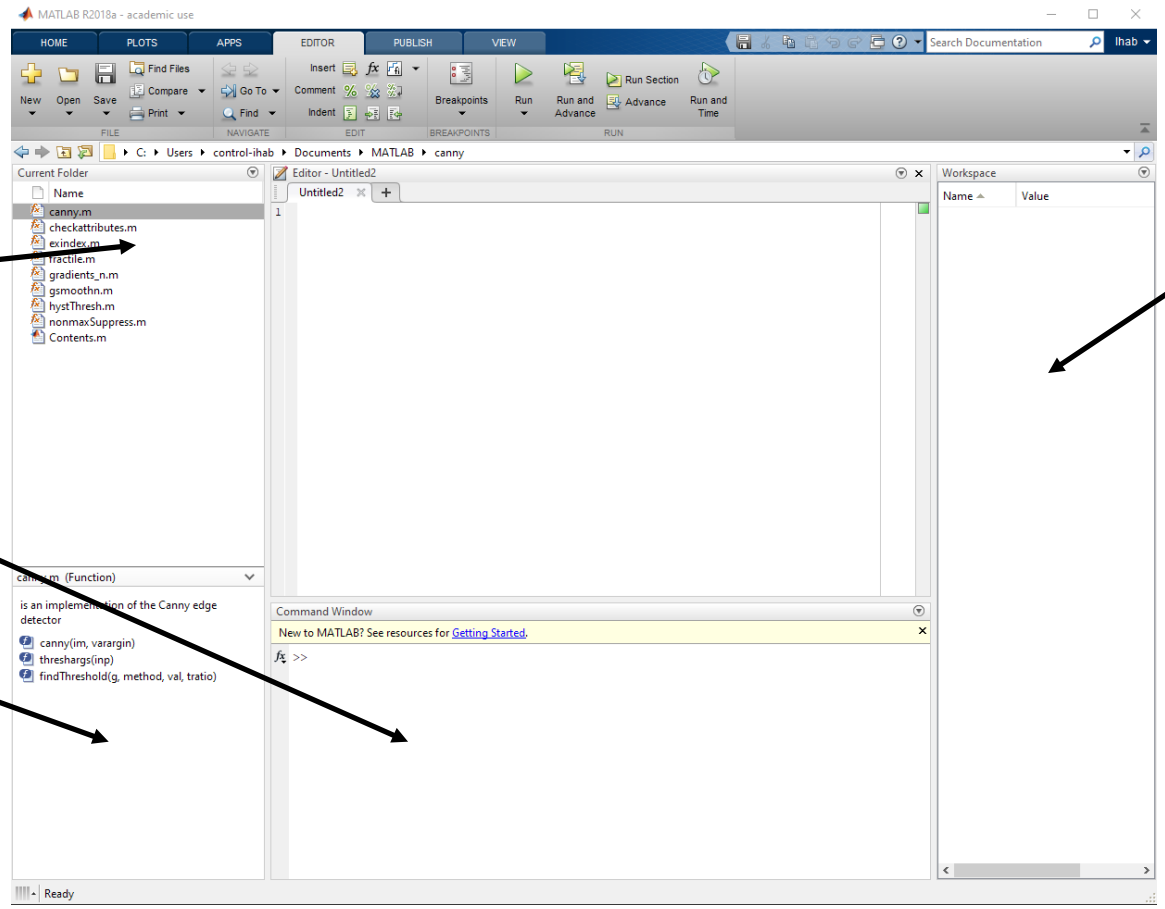
- Performing mathematical computations, image processing, signal processing …etc
- Analyzing and visualizing data (excellent graphics tools).
- Modeling physical systems and phenomena.
- Testing engineering designs.

Industry Applications

- **Aircraft/Defense:** control and guidance system design and simulation, communications.
- **Robotics:** design and control.
- **Automotive:** cruise control, stability enhancement, fuel injection systems, hybrid power-train, sound suppression …
- **Communications:** voice over internet, cell-phone, satellite, antenna design, wireless, error coding …
- **Biotech, Pharmaceutical, Medical:** drug discovery and development, imaging procedures, cancer diagnosis …
- **Electronics:** chip design, acoustics, voice processing and recognition.
- **Industrial Automation and Machinery:** sensor design, machinery design and control.
- **Utilities and Energy:** power conversion and control.
- **Computers:** security systems, printer design.
- **Financial:** portfolio management and risk, commodity trading, currency markets.

MATLAB Basics

MATLAB Desktop



Current Directory

- View folders and m-files

Command Window

- type commands

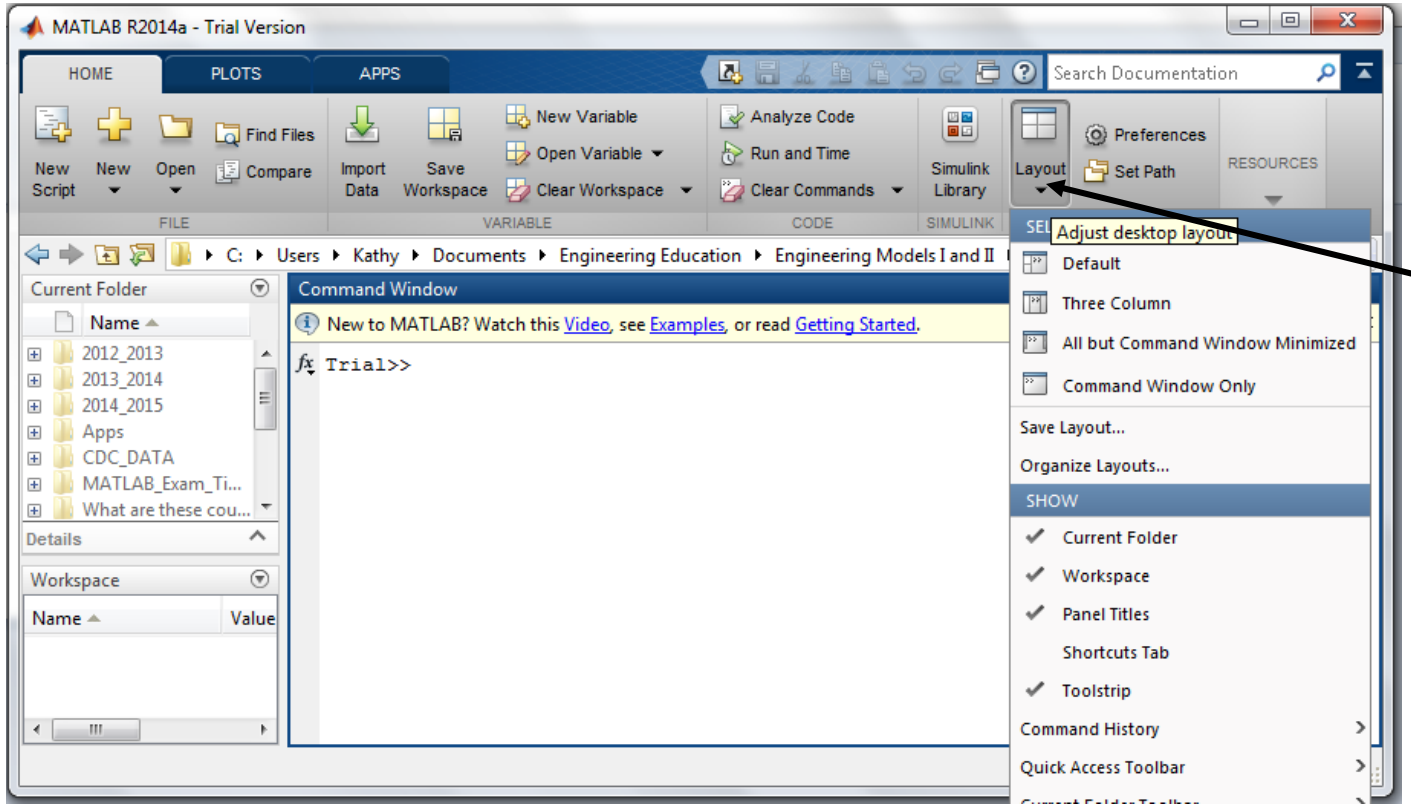
Command History

- view past commands
- save a whole session using diary

Workspace

- View program variables
- Double click on a variable to see it in the Array Editor

MATLAB Desktop



You can select what is on your desktop by Clicking on Layout. Go down to Command History and select docked.

Naming Rules for Variables

1. Variable names must begin with a letter
2. Names can include any combinations of letters, numbers, and underscores
3. Maximum length for a variable name is 63 characters
4. MATLAB® is case sensitive. The variable name A is different than the variable name a.
5. Avoid the following names: i, j, pi, and all built-in MATLAB® function names such as length, char, size, plot, break, cos, log, …
6. It is good programming practice to name your variables to reflect their function in a program rather than using generic x, y, z variables.

Creating Variables & Assigning Values

When typing at the MATLAB command prompt (>>):

```
x = 10.57;
```

Several things happen with this simple MATLAB command:

- A variable, **x**, of type double is created
- A memory location for the variable **x** is assigned
- The value **10.57** is stored in that memory location called **x**.

Creating Variables & Assigning Values

When typing at the MATLAB command prompt (>>):

```
x = 73.65
```

Several things happen with this command:

- The old value for **x** (10.57) is replaced by the new value (**73.65**)
- Also, since the semicolon was left off the end, we see the result in the command window (as well as in the workspace window)

Creating Variables & Assigning Values

Variable do not have to be numbers. When typing at the MATLAB command prompt:

```
month = 'August'
```

Several things happen with this command:

- A variable, **month**, of type string (character array) is created
- A memory location for the variable **month** is assigned
- The string **August** is stored in that memory location called **month**.
- Notice that to enter a string, we must put single quotes around it '**month**'

Displaying Variables

We can display a variable (i.e., show its value) by simply typing the name of the variable at the command prompt (leaving off the semicolon).

We can also use a function called **`disp`** to display variables. Type the following commands at the command prompt:

```
>>  disp('The value of x is:'); disp(x)
```

Arithmetic Operators and Order of Operations

- Addition (+), Subtraction (-), Multiplication (*), Division (/), Power (^)
- Order of Operations (same rules you should already know from math class and using a calculator)
  1. Complete all calculations inside parenthesis or brackets using the precedent rules below
  2. Powers (left to right)
  3. Multiplication and Division (left to right)
  4. Addition and Subtraction (left to right)

Arithmetic Operators and Order of Operations

Some Examples:

```
>>   10/5*2

>>   5*2^3+4*(2)

>> -1^4

>>   8^1/3
```

Exercise 1

1. In MATLAB create two variables: a = 4 and b = 17.2
2. Now use MATLAB to perform the following set of calculations:

$$5a$$

$$\sqrt[3]{b + 9.8}$$

$$10\sqrt{5a + 16}$$

Some MATLAB® Functions

Function	MATLAB®		Function	MATLAB®
cosine	cos or cosd		square root	sqrt
sine	sin or sind		exponential	exp
tangent	tan or tand		logarithm (base 10)	log10
cotangent	cot or cotd		natural log (base e)	log
arc cosine	acos or acosd		round to nearest integer	round
arc sine	asin or asind		round down to integer	floor
arc tangent	atan or atand		round up to integer	ceil
arc cotangent	acot or acotd			

Note: cos(α) assumes α in radians; whereas, cosd(α) assumes α in degrees.
acos(x) returns the angle in radians; whereas, acosd(x) returns the angle in degrees.

π radians = 180 degrees

Other Useful Stuff

- **`clear`** clears all variables in the MATLAB® workspace
- **`clear a, b`** just clears variables a and b
- **`clc`** clears the command window
- **`i`** and **`j`** are defined in MATLAB to be $\sqrt{-1}$. If you define these variables to be something else, you lose the ability to work with complex numbers. So, avoid using i and j as variables.
- **`pi`** is defined in MATLAB as **`3.14159`**….

Help!

- The **`help`** command provides information about a function. Type **`help cos`** at the command prompt. This only works if you know the name of the function you want help with.

- You can also click on Help in the MATLAB toolbar and search for information by keyword(s).

Script Files

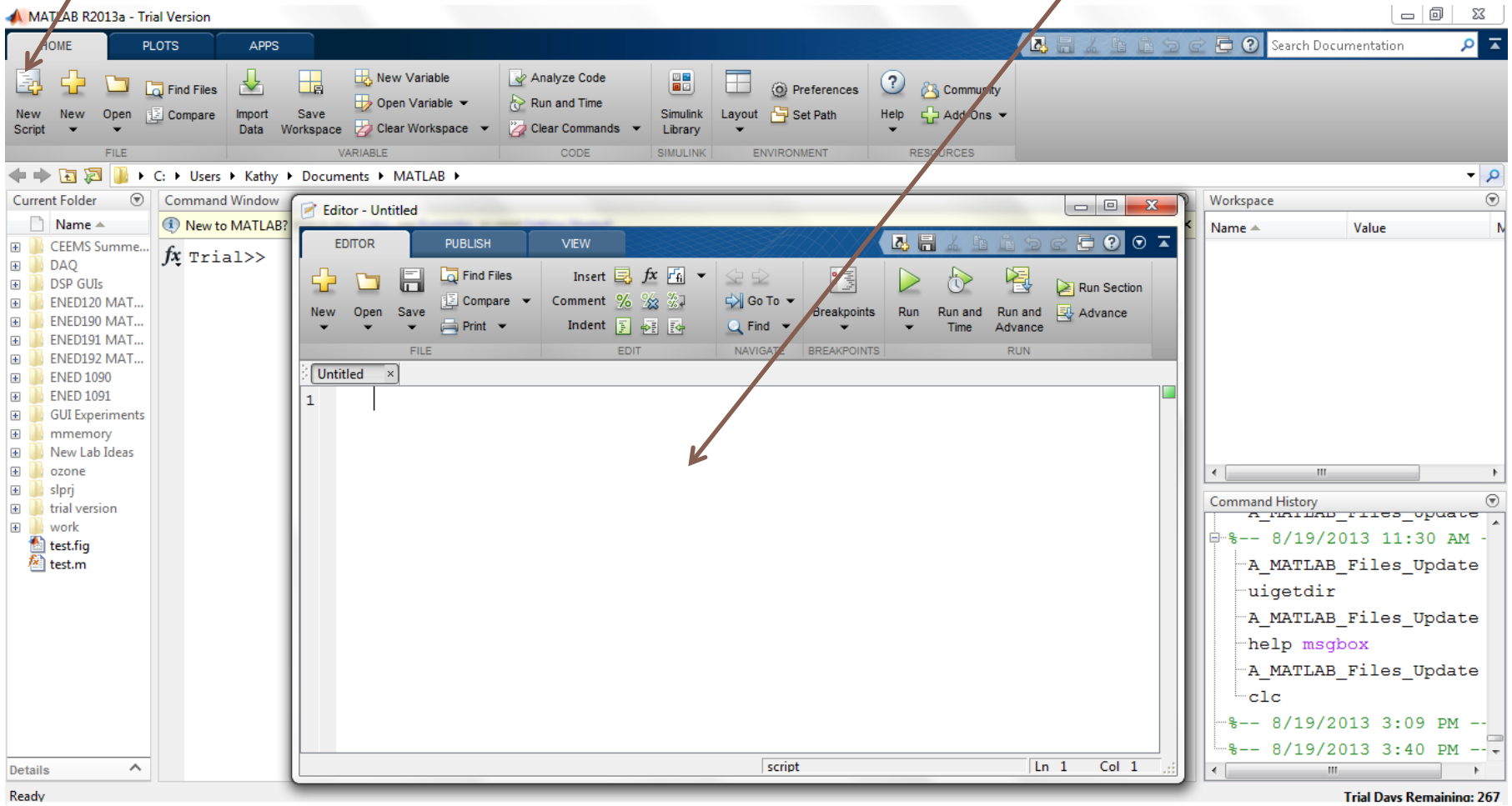
Script Files

- All of the pre-built commands that you use in MATLAB® are ***script files*** or ***functions*** (plot, mean, std, exp, cosd, …)
- MATLAB® allows the user to create his/her own customized m-files for specific applications or problems.
- A script file is simply a collection of executable MATLAB® commands. To create a new script file, click on the New Script icon on the left side of the Home Tab.
- Function files will be covered in Models II.

Script Files

Click on New Script

Creates Blank Script File



Script File: Procedure

1. Type a set of executable commands in the editor window.

2. Save the file in an appropriate folder. ***When you pick a name for the file you must follow the same rules that MATLAB has for naming variables.***

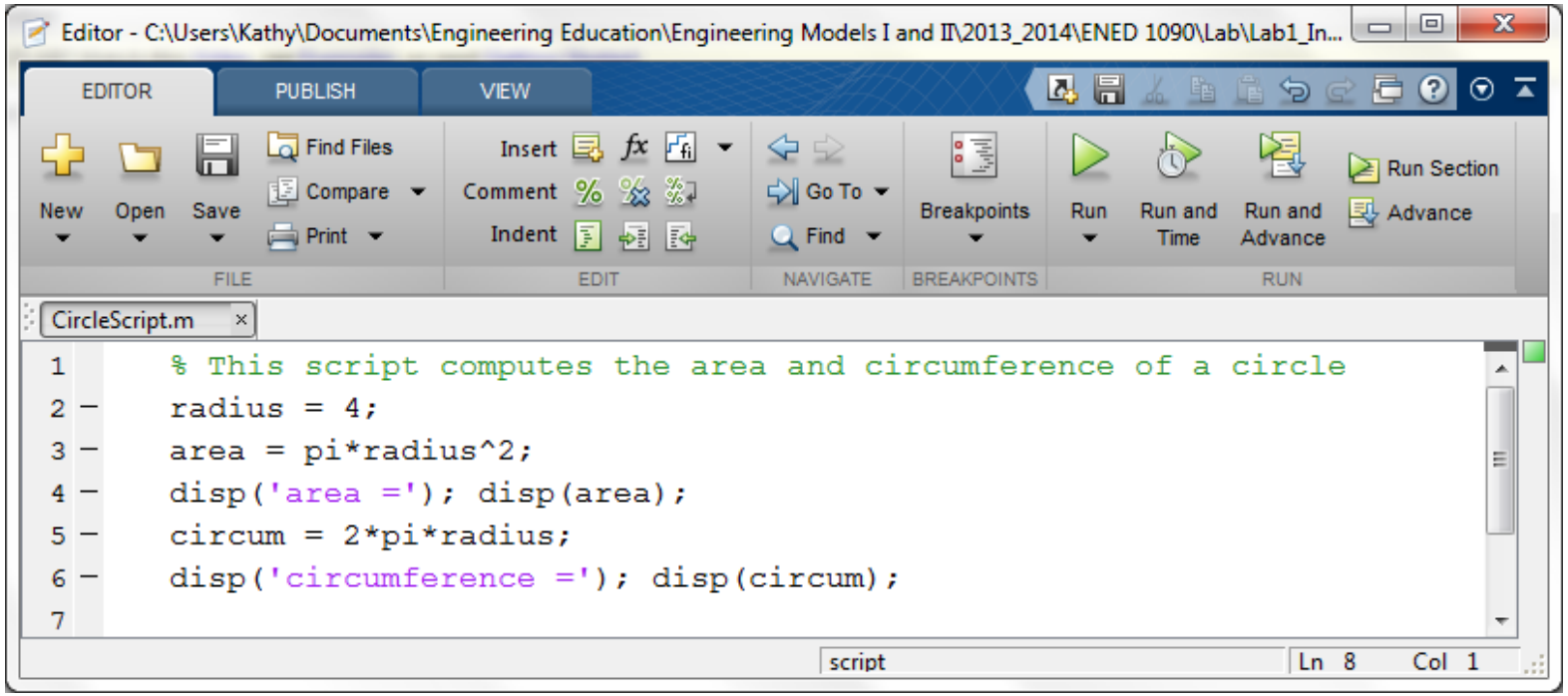
3. Set the current directory in MATLAB to the same place where you saved the Green Run Arrow in the Editor window or simply type the name of the file (without the .m extension) at the command prompt in the MATLAB command window.

The file must be in your current directory or it will not run!

Exercise 2: New Script File

- Right click in the current folder window in MATLAB and create a new folder called Models I. This is where you should save all your files for the fall semester.
- Double click on the folder to make it your current folder.
- Clear your MATLAB workspace by typing **`clear`** at the command prompt.
- Click on New Script to open a blank script file.
- Type the commands on the next slide into the editor window then save the file as CircleScript in your newly created folder.

Exercise 2: Script File



Save the file as CircleScript in your newly created folder.

Note: Any line that starts with a % is a comment and turns green – it doesn't execute.

Exercise 2: Run the Script File

- Now run your script file by clicking on the Green Arrow in the m-file editor window.
- Notice that every single variable defined in the script file (radius, area, and circum) appears in the Workspace Window. Area and circum are also displayed in the Command Window because of the **`disp`** command.
- Clear the workspace window by typing **`clear`** at the command prompt.
- At the command prompt, type the name of your script file: >> **`CircleScript`**. Note, that the results are exactly the same as before.

Script Files

- A script file is simply a set of executable MATLAB commands saved together in a file.
- It behaves exactly the same as entering each command sequentially in the command window.
- Every single variable defined appears in the workspace.
- Script files can be really useful if you are going to execute a lot of MATLAB commands. For example, suppose you execute 15 commands at the command prompt and discover an error in the first command that affected the last 14 commands.
- In the command window, you would have to fix the error in the first command then run the other 14 commands over again. If these commands were in a script file, you could fix the first command and re-run the script file – much faster !!

Introduction to Arrays

Arrays

- An array is a collection of like elements.
- There are many engineering applications that use arrays.
- MATLAB® is an acronym for **Mat**rix **Lab**oratory.
- (A matrix is a two-dimensional array)
- MATLAB® stores data in arrays and performs all numerical computations using array operations.
- Therefore, to use MATLAB® effectively as a computing tool, one must understand arrays and operations with arrays.

1-d Arrays: Vectors

A vector is a one-dimensional array.

Examples:

A row vector with 4 elements $\mathbf{x} = [\ 0 \quad -1.5 \quad 4 \quad 7\ ]$

A column vector with 3 elements $\mathbf{y} = \begin{bmatrix} 5 \\ 2.9 \\ 3 \end{bmatrix}$

2-d Arrays: Matrices

A matrix is a two-dimensional array (like a table).

Examples:

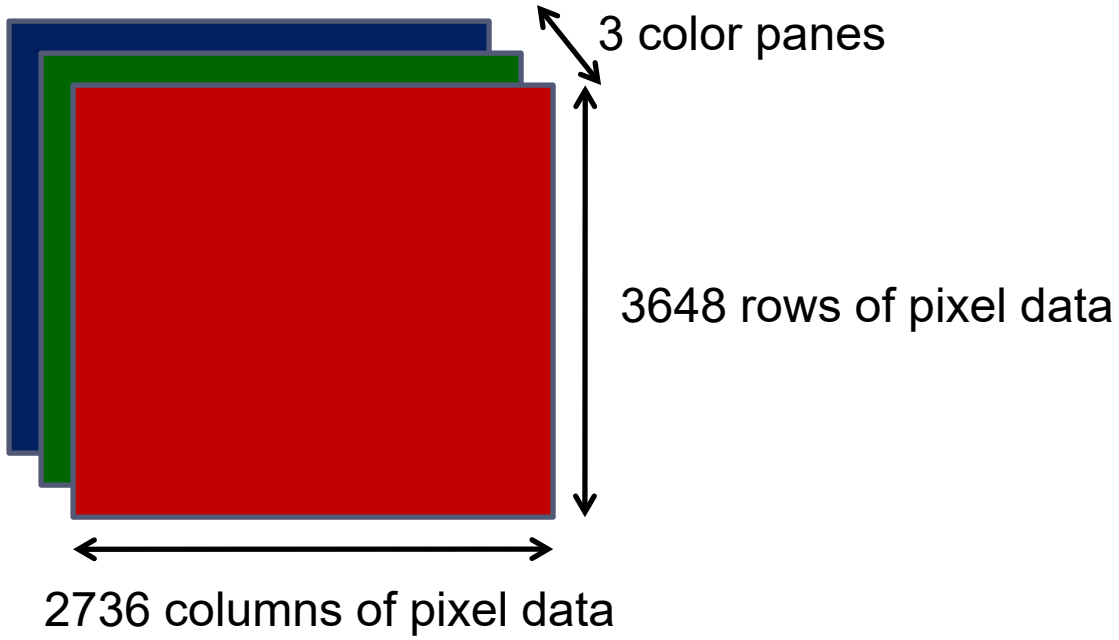
$$\mathbf{A} = \begin{pmatrix} 2.3 & 4.5 & -7.5 \\ 3.2 & -5.1 & 10 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 76 & 75 & 84 & 82 \\ 92 & 95 & 89 & 88 \\ 67 & 65 & 70 & 75 \end{pmatrix}$$

A has 2 rows and 3 columns (2x3 Matrix)

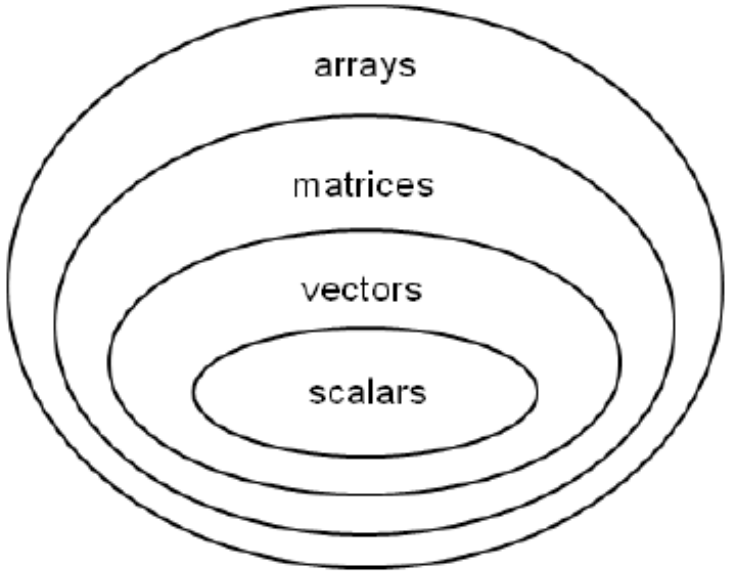
B has 3 rows and 4 columns (3x4 Matrix)

Multi-Dimensional Arrays

Arrays can have more than two dimensions. For example, a 3648 x 2736 jpg color image imported into MATLAB® would be a 3-dimensional array of size 3648 x 2736 x 3 where the 3rd dimension represents the RGB panes as illustrated below.



Introduction to Arrays

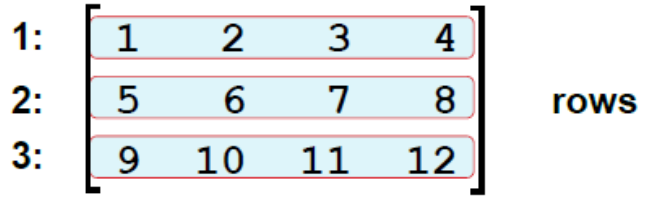


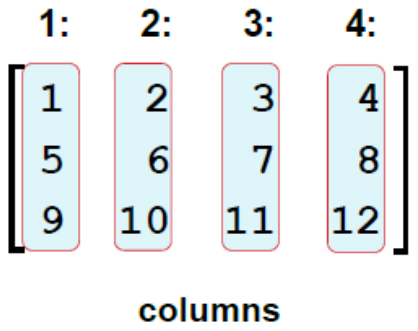
Creating 1-d Arrays (Vectors)

Creating Vectors in MATLAB®

```
>> a = [2.3  7.5  4.3  6]
% Creates a single row of numbers
      a =   2.3000    7.5000    4.3000    6.0000
```

```
>> b = [2.3;  7.5;  4.3;  6]
% Creates a single column of numbers
      b =
            2.3000
            7.5000
            4.3000
            6.0000
```





Creating Vectors in MATLAB®

Indexing:

```
>> X = [1:4; 5:8; 9:12];
>> X(2,3)
```

3:

2:

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

```
>> ans =

     7
```

Creating Vectors: Other Options


```
>> t = -1:0.5:3
t =
   -1   -0.5    0    0.5    1.0    1.5    2.0    2.5    3.0
```

START

INCREMENT = 0.5

MAX.

Note: If you leave out the increment (middle value), MATLAB will set increment = 1

```
>> t = -1:0.6:3
t =
   -1   -0.4    0.2    0.8    1.4    2.0    2.6
```

Creating Vectors: Other Options


```
>> t = linspace(-1,3,7)

t =

   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
```

START

7 Data Points

END

Entry by Entry Math Operators

Example

Suppose we create a vector, t, as follows:

```
>> t = 0:2:10
     t =     0     2     4     6     8    10
```

What if we wanted to square each entry in the vector, t?

```
>> t^2
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^)
instead.
```

Arithmetic Operators

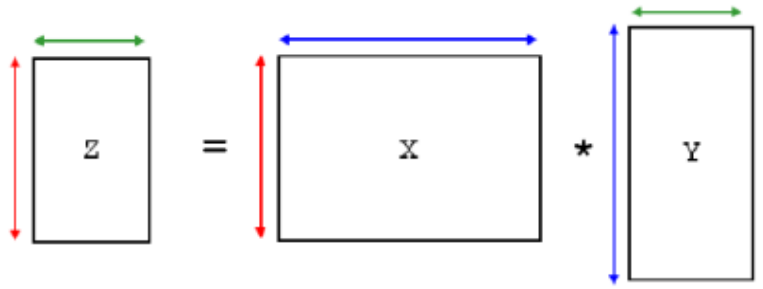
MATLAB® stores data in arrays and performs all numerical computations using array operations.

All of the arithmetic operators: + – * / ^
perform array operations and must follow the rules for arrays.

[0 2 4 6 8 10] * [0 2 4 6 8 10]

(1 x 6) * (1 x 6)

Inner Matrix Dimensions are not the same. Invalid Multiplication



Entry by Entry Operators

MATLAB provides entry by entry operators that allow the user to perform computations on each entry in an array.

The entry by entry operators are: .* .^ ./

Back to Example

```
>> t = 0:2:10
     t =     0     2     4     6     8    10
```

How do we square each entry in the vector, t?

```
>> t.^2

ans =

     0     4    16    36    64   100
```

Your Turn …

Try these commands (one at a time) in MATLAB.

Explain what each command does.

```
>> x = [5  -3  7  -10]
>> y = [1;  3;  -17]
>> t = 0:0.1:2
>> z = linspace(0,2,11)
>> q = [1  -2  3  9]; q^3
>> q = [1  -2  3  9]; q.^3
>> a = [1 2 3]; b = [4 5 6]; a*b
>> a = [1 2 3]; b = [4 5 6]; a.*b
>> a = [1 2 3]; b = [4 5 6 7]; a.*b
```

ENMC3111
Computer Aided Engineering

Lecture #2

Ihab Abu Ajamieh, PhD.

Introduction to Arrays

Arrays

- An array is a collection of like elements.
- There are many engineering applications that use arrays.
- MATLAB® is an acronym for **Mat**rix **Lab**oratory.
- (A matrix is a two-dimensional array)
- MATLAB® stores data in arrays and performs all numerical computations using array operations.
- Therefore, to use MATLAB® effectively as a computing tool, one must understand arrays and operations with arrays.

Why Arrays?

- MATLAB® stores data in arrays and performs all numerical computations using array operations.
- Therefore, to use MATLAB® effectively as a computing tool, one must understand arrays and operations with arrays.

1-d Arrays: Vectors

A vector is a one-dimensional array.

Examples:

A row vector with 4 elements $\mathbf{x} = [\ 0 \quad -1.5 \quad 4 \quad 7\ ]$

A column vector with 3 elements $\mathbf{y} = \begin{bmatrix} 5 \\ 2.9 \\ 3 \end{bmatrix}$

2-d Arrays: Matrices

A matrix is a two-dimensional array (like a table).

Examples:

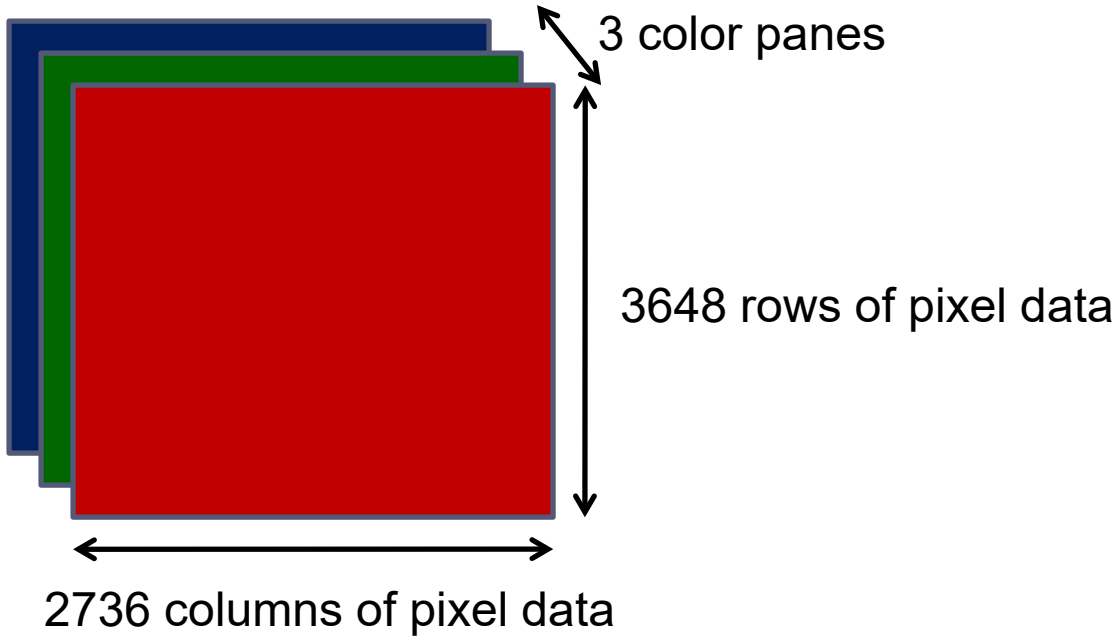
$$\mathbf{A} = \begin{pmatrix} 2.3 & 4.5 & -7.5 \\ 3.2 & -5.1 & 10 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 76 & 75 & 84 & 82 \\ 92 & 95 & 89 & 88 \\ 67 & 65 & 70 & 75 \end{pmatrix}$$

A has 2 rows and 3 columns (2x3 Matrix)

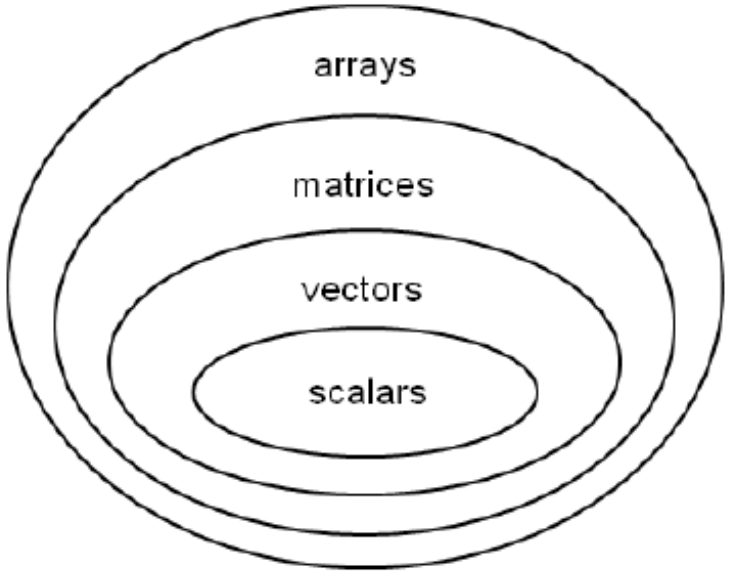
B has 3 rows and 4 columns (3x4 Matrix)

Multi-Dimensional Arrays

Arrays can have more than two dimensions. For example, a 3648 x 2736 jpg color image imported into MATLAB® would be a 3-dimensional array of size 3648 x 2736 x 3 where the 3rd dimension represents the RGB panes as illustrated below.



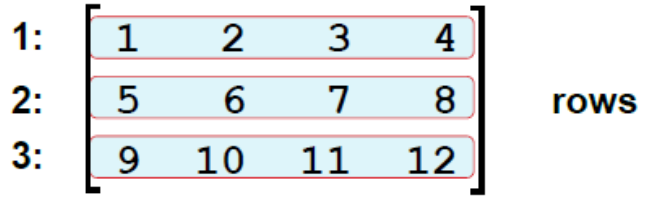
Introduction to Arrays



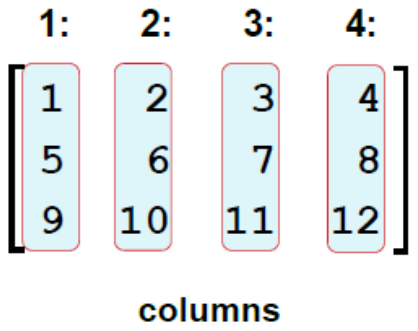
Creating 1-d Arrays (Vectors)

Creating Vectors in MATLAB®

```
>> a = [2.3  7.5  4.3  6]
% Creates a single row of numbers
      a =   2.3000    7.5000    4.3000    6.0000
```




```
>> b = [2.3;  7.5;  4.3;  6]
% Creates a single column of numbers
      b =
            2.3000
            7.5000
            4.3000
            6.0000
```



Creating Vectors in MATLAB®

Indexing:

```
>> X = [1:4; 5:8; 9:12];
>> X(2,3)
```

3:

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

```
>> ans =

     7
```

Creating Vectors: Other Options


```
>> t = -1:0.5:3
t =
   -1   -0.5    0    0.5    1.0    1.5    2.0    2.5    3.0
```

START

INCREMENT = 0.5

MAX.

Note: If you leave out the increment (middle value), MATLAB will set increment = 1

```
>> t = -1:0.6:3
t =
   -1   -0.4    0.2    0.8    1.4    2.0    2.6
```

Creating Vectors: Other Options


```
>> t = linspace(-1,3,7)

t =

   -1.0000   -0.3333    0.3333    1.0000    1.6667    2.3333    3.0000
```

START

7 Data Points

END

Vector operations

```
% Built-in matlab functions operate on vectors, if a matrix is given,
% then the function operates on each column of the matrix

a = [1 4 6 3]                    % vector
sum(a)                           % sum of vector elements
mean(a)                          % mean of vector elements
var(a)                           % variance
std(a)                           % standard deviation
max(a)                           % maximum

a = [1 2 3; 4 5 6]               % matrix
a(:)                             % vectorized version of the matrix
mean(a)                          % mean of each column
max(a)                           % max of each column
max(max(a))                      % to obtain max of matrix
max(a(:))                        % or...

A = zeros(3)                     % 3x3 zeros matrix
B = magic(4)                     % 4x4 matrix
C = ones(5)                      % 5x5 ones matrix.
D = rand(10,2)                   % 10x2 rand matrix.
E = eye(5)                       % 5x5 matrix - diagonal ones.
```

Entry by Entry Math Operators

Example

Suppose we create a vector, t, as follows:

```
>> t = 0:2:10
     t =     0     2     4     6     8    10
```

What if we wanted to square each entry in the vector, t?

```
>> t^2
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^)
instead.
```

Arithmetic Operators

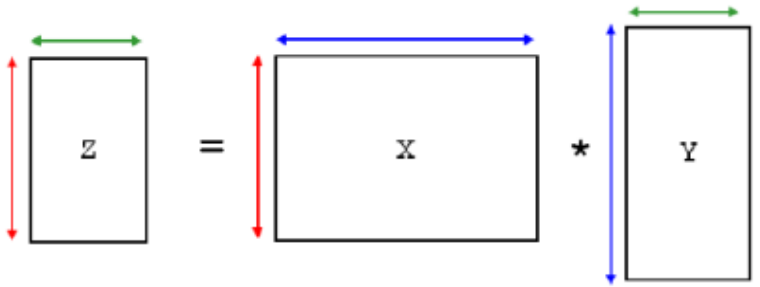
MATLAB® stores data in arrays and performs all numerical computations using array operations.

All of the arithmetic operators: + – * / ^
perform array operations and must follow the rules for arrays.

[0 2 4 6 8 10] * [0 2 4 6 8 10]

(1 x 6) * (1 x 6)

Inner Matrix Dimensions are not the same. Invalid Multiplication



Entry by Entry Operators

MATLAB provides entry by entry operators that allow the user to perform computations on each entry in an array.

The entry by entry operators are: .* .^ ./

Back to Example

```
>> t = 0:2:10
     t =     0     2     4     6     8    10
```

How do we square each entry in the vector, t?

```
>> t.^2

ans =

     0     4    16    36    64   100
```

Your Turn …

Try these commands (one at a time) in MATLAB.

Explain what each command does.

```
>> x = [5  -3  7  -10]
>> y = [1;  3;  -17]
>> t = 0:0.1:2
>> z = linspace(0,2,11)
>> q = [1  -2  3  9]; q^3
>> q = [1  -2  3  9]; q.^3
>> a = [1 2 3]; b = [4 5 6]; a*b
>> a = [1 2 3]; b = [4 5 6]; a.*b
>> a = [1 2 3]; b = [4 5 6 7]; a.*b
```

Introduction to Graphing Using MATLAB

Line Graphs

- Useful for graphing functions
- Useful for displaying data trends over time
- Useful for showing how one variable depends on another

Line Graphs

The MATLAB command for creating a line graph is ***plot***.

General Form:

% A single Function

plot(x-coordinates, y-coordinates, optional formatting)

% Multiple Functions

plot(x-values of f_1, y-values of f_1, formatting for f_1, x-values of f_2, y-values of f_2, formatting for f_2, …)

Format Options (color, linestyle, …)

At the command prompt, type: >> help plot

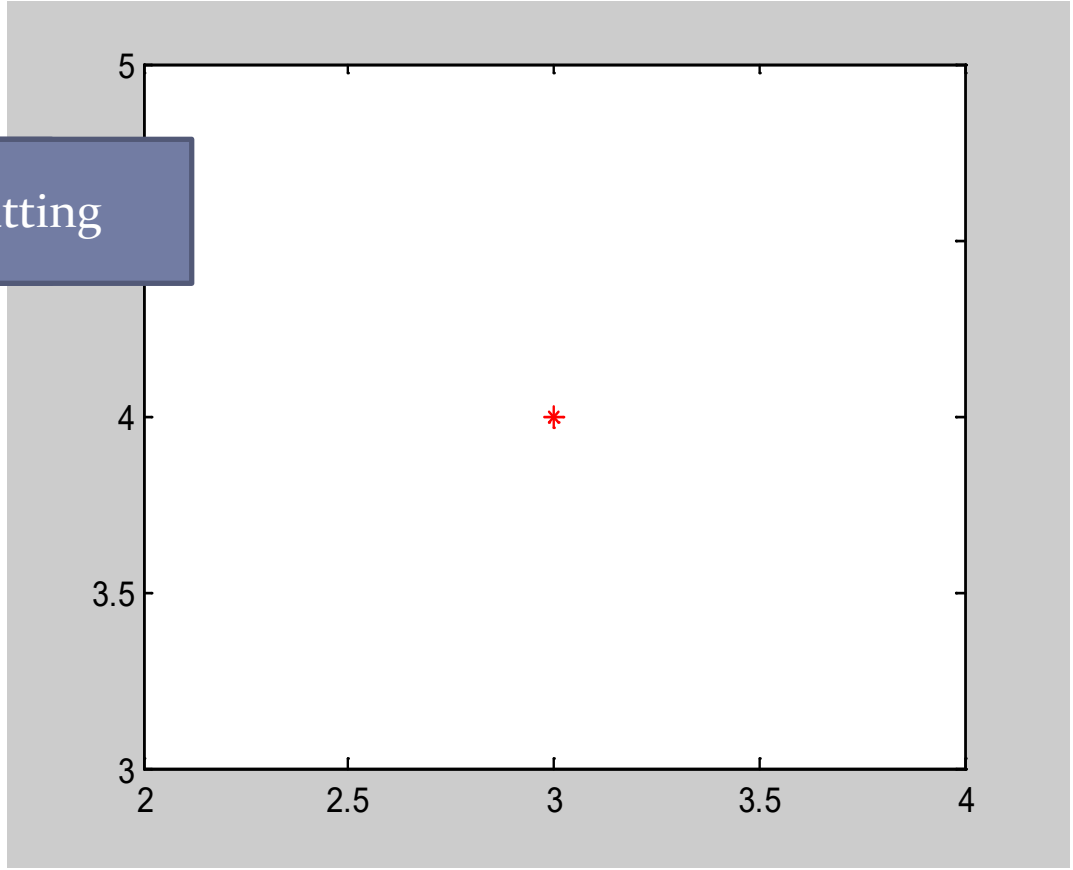
Scroll up to see this table of options:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		

Simple Examples

X Y-C Formatting

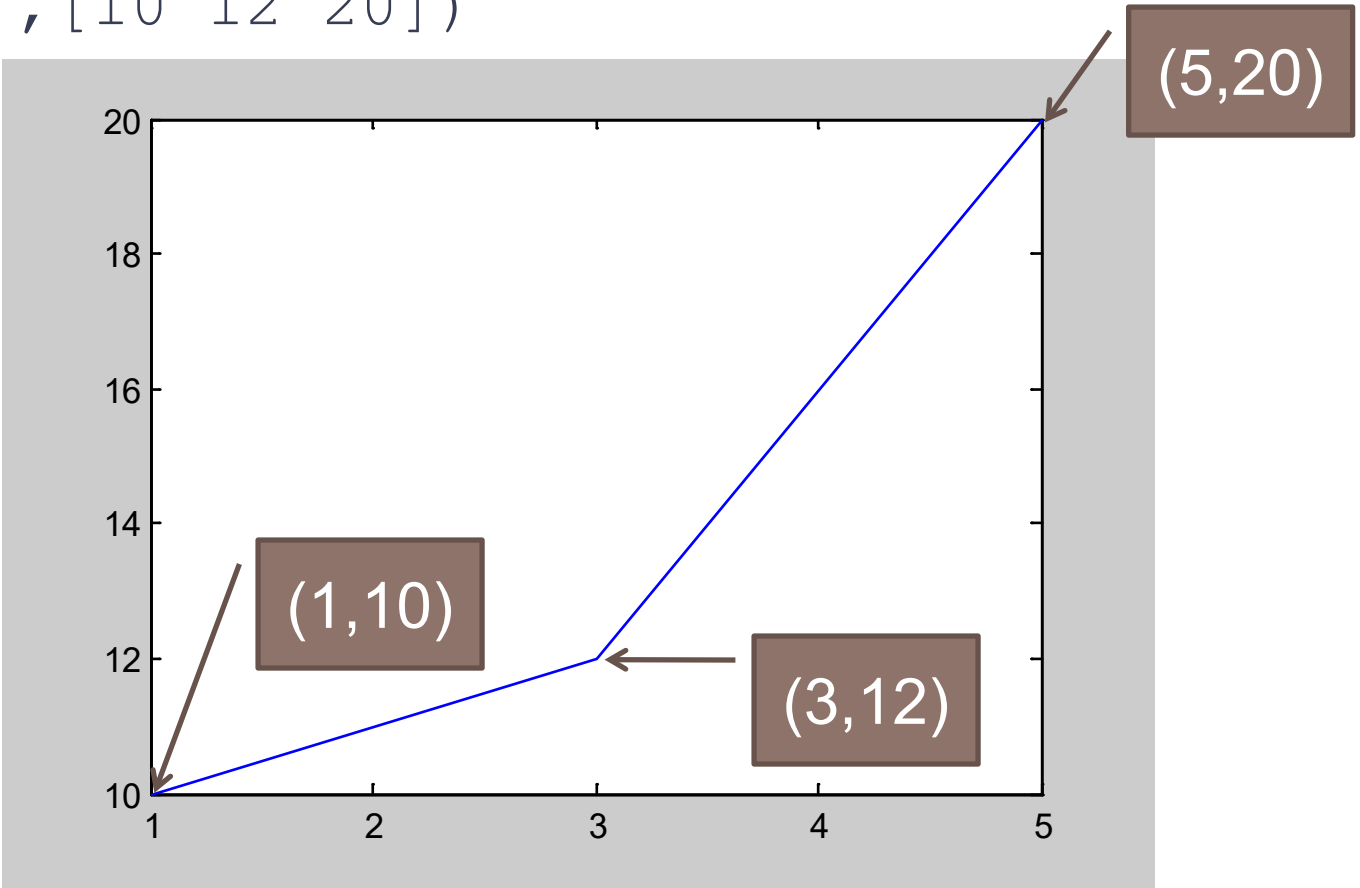
```
>> plot(3,4,'r*')
```



les

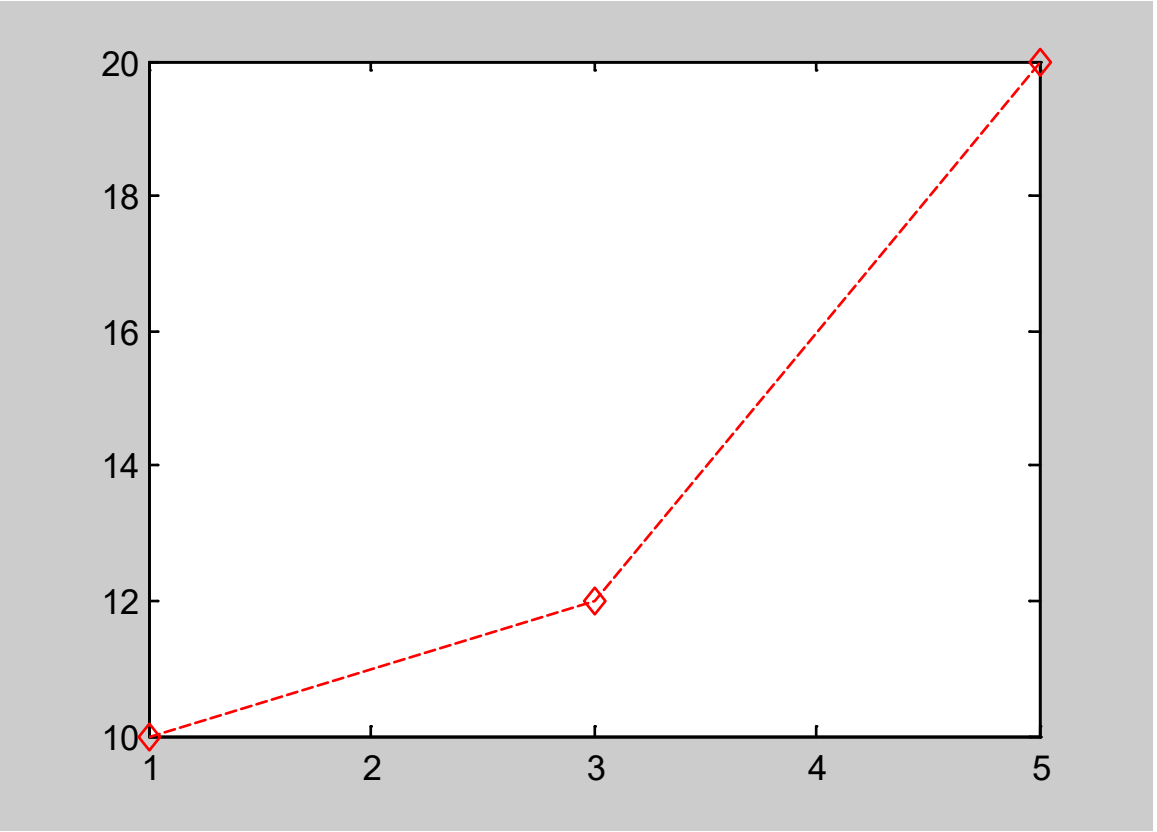
X-Coordinates Y-Coordinates

```
>> plot([1 3 5],[10 12 20])
```



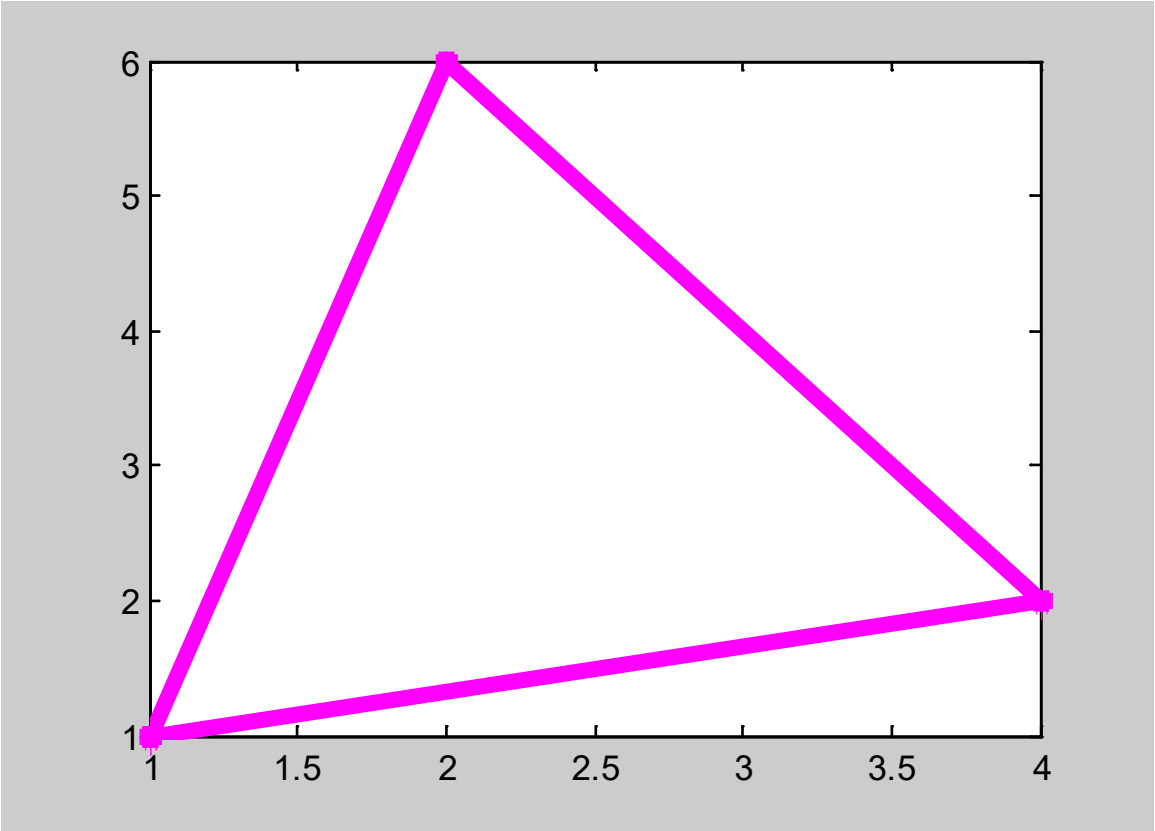
Simple Examples

```
>> plot([1 3 5],[10 12 20],'rd--')
```



Simple Examples

```
>> plot([1 2 4 1],[1 6 2 1],'m*-','LineWidth',5)
```



Graphing Functions

Graph the polynomial function $y = t^3 - 6t^2 + 3t + 10$

Must generate a set of t-values to go on the x-axis then calculate the corresponding y-values. A few options:

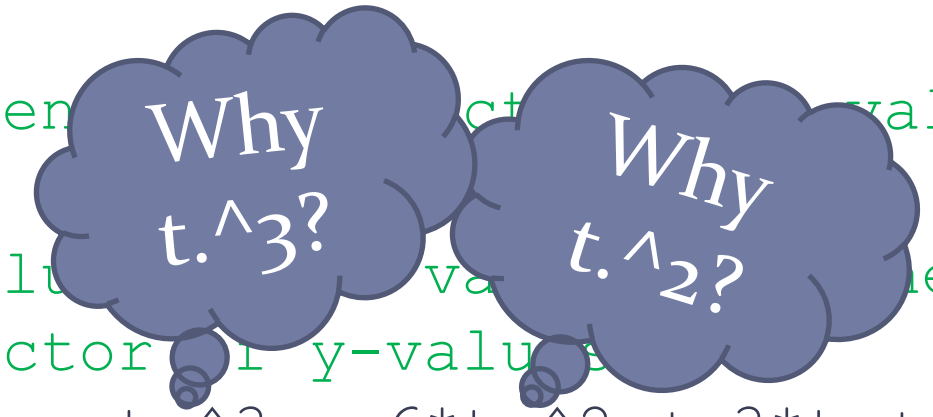
```
>> t = [-2 -1.5 -1 -0.5 0 0.5 1 1.5 2]
% This works OK if you have only a few t-values
```


```
>> t = -2:0.01:7;  % Generates a vector of 901
t-values from -2 to +7 spaced apart by 0.01
```

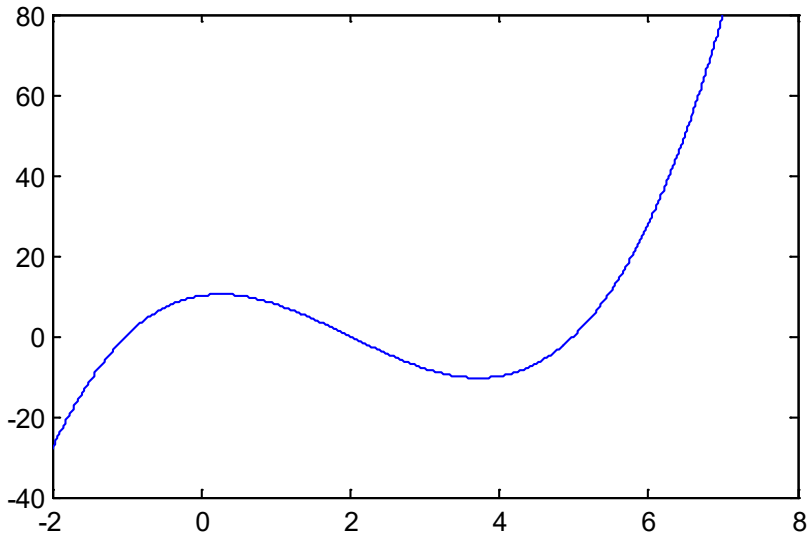

```
>> t = linspace(-2,7,500); % Generates a vector
of 500 t-values evenly spaced from -2 to +7
```

Graphing Functions

Graph the polynomial function $y = t^3 - 6t^2 + 3t + 10$



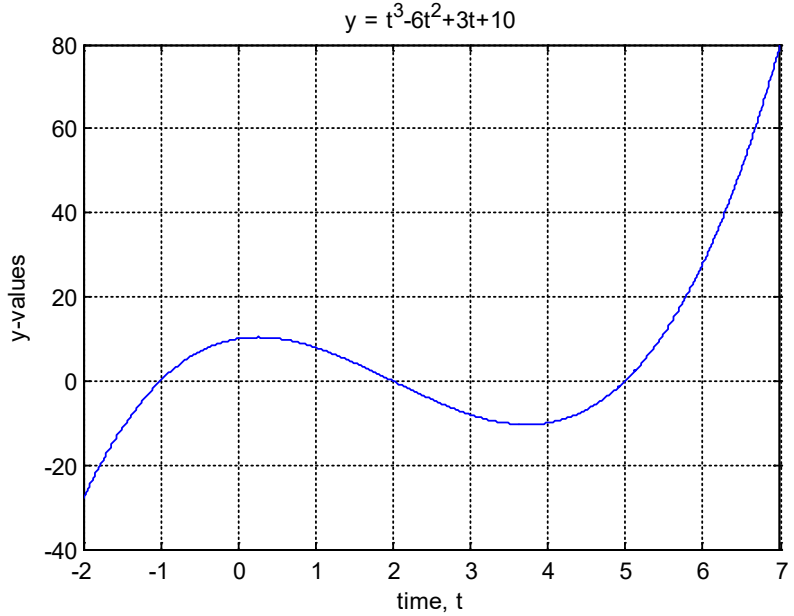

```
%  Gen          ct      values for x-axis
>> t
%  Plu          va          e equation to get
a vector of y-valu
>> y = t.^3 - 6*t.^2 + 3*t + 10;

>> plot(t,y)
```



Graphing Functions

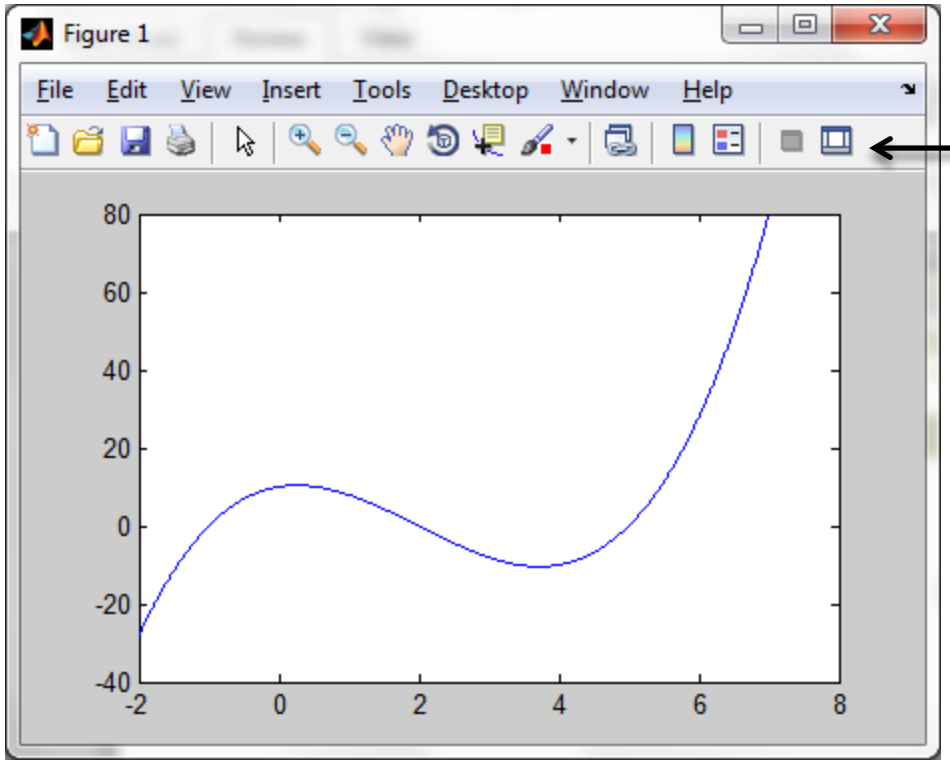
Plots should be labeled and titled. This can be done using MATLAB commands or by using plot tools. Commands:

```
>> plot(t,y)
>> xlabel('time, t'); ylabel('y-values');
>> title('y = t^3-6t^2+3t+10'); grid
```



Plot Tools

Plot Tools is another nice option for editing graphs.



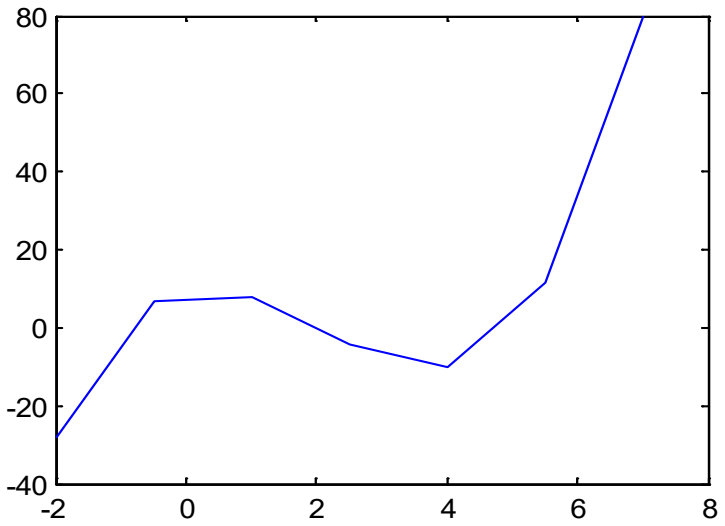
Click this icon to open plot tools

Common Errors

Choosing the x-axis values poorly.

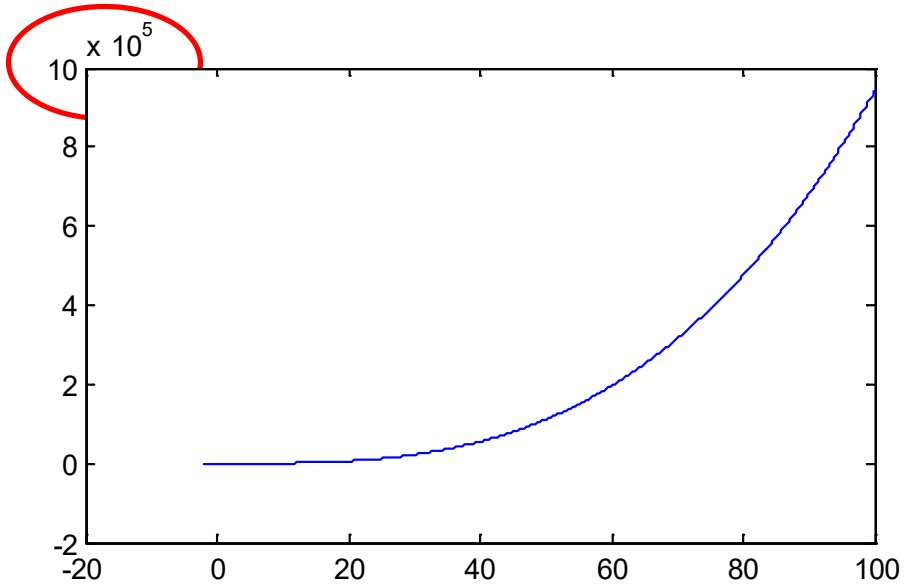
Increment too large:

```
>> t = -2:1.5:7;
>> y = t.^3-6*t.^2+3*t+10;
>> plot(t,y)
```



Poor choice for range of x-axis:

```
>> t = -2:0.01:100;
>> y = t.^3-6*t.^2+3*t+10;
>> plot(t,y)
```



Solving Equations Graphically

Suppose a capacitor is charging in an RC circuit and the voltage across the capacitor is given by:

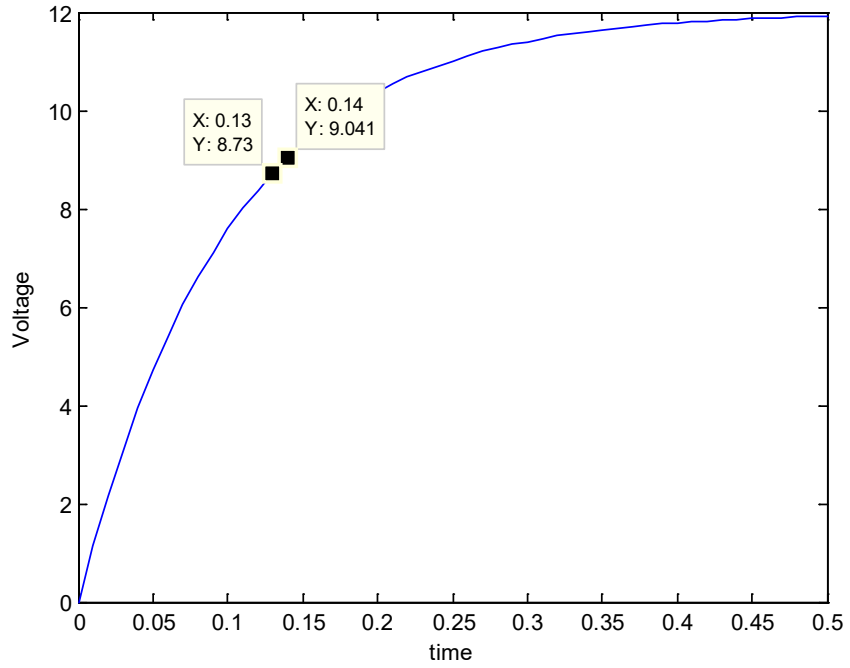
$$V_c = 12(1 - e^{-10t})$$

Vc is in volts and t is in seconds. Plot the voltage across the capacitor versus time then determine the time at which the capacitor voltage reaches 9 volts.

```
>> t = 0:0.01:0.5;
>> y = 12*(1-exp(-10*t));
>> plot(t,y);xlabel('time');ylabel('Voltage');
```

Solving Equations Graphically

In the Figure Window, Click on Tools then select Data Cursor.
Click on graph – move data cursor if needed using arrow keys.
To add additional datatips, right click on an existing datatip and select add new datatip.



The capacitor reaches 9 volts between t = 0.13 seconds and t = 0.14 seconds.

Note: Our precision is limited by the increment chosen for t which was 0.01 seconds in this example.

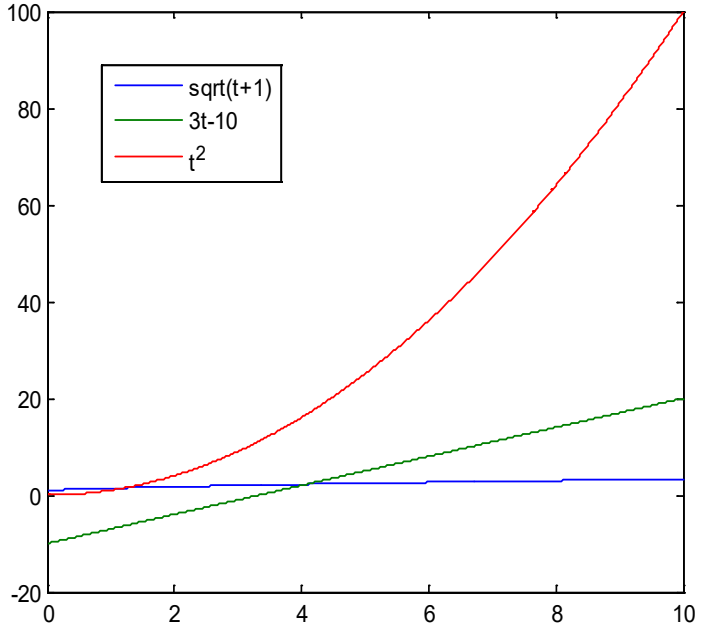
Multiple Plots on a Single Graph

Plot each of the following functions on the same graph:

$$f_1 = \sqrt{t+1} \qquad f_2 = 3 * t - 10 \qquad f_3 = t^2$$

```
>> t = 0:0.01:10;
>> f1 = sqrt(t+1); f2 = 3*t-10; f3 = t.^2;
>> plot(t,f1,t,f2,t,f3);
>> legend('sqrt(t+1)','3t-10','t^2')
```

Note: These functions don't look so great on the same plot. The function t^2 increases so much faster than the square root function it causes the square root function to look pretty flat.

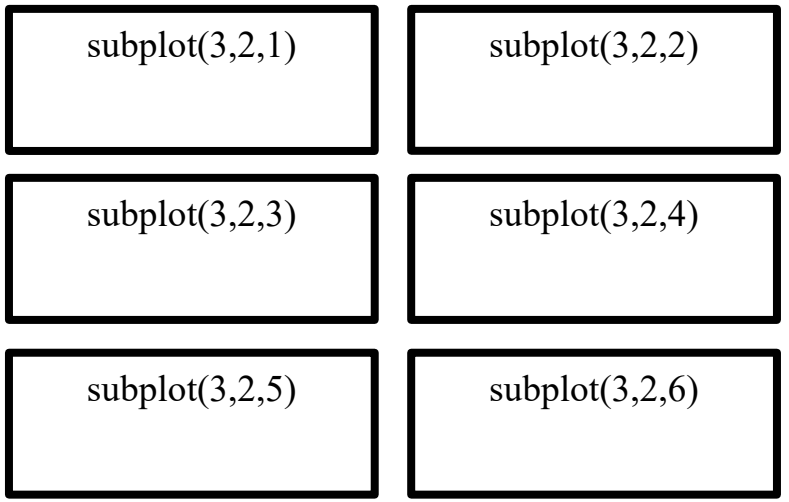


Subplot Command

subplot`(m,n,k)`

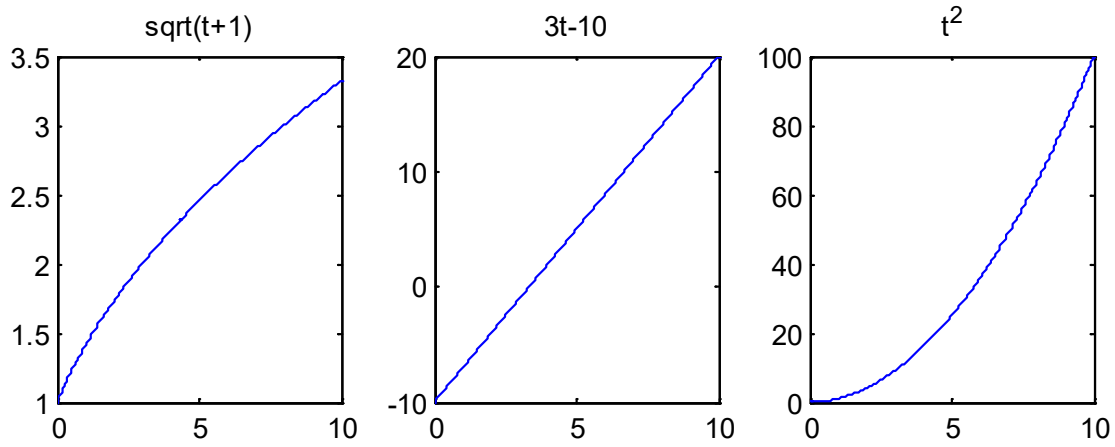
The subplot command splits the figure window into several sub-windows. The first two entries in subplot show how the window is to be split up by specifying number of rows and number of columns. The third entry points to a particular sub-window.

Subplot(3,2,4) would divide the plot window into 3 rows and 2 columns allowing for 6 smaller plot windows and would point to the 4th sub-window as shown in the diagram.



Multiple Plots Using Subplot

Repeat the previous example but put each plot in a separate sub-window of the figure using subplot.




```
>> t = 0:0.01:10;
>> f1 = sqrt(t+1); f2 = 3*t-10; f3 = t.^2;
>> subplot(1,3,1);
>> plot(t,f1);title('sqrt(t+1)')
>> subplot(1,3,2);plot(t,f2);title('3t-10')
>> subplot(1,3,3);plot(t,f3);title('t^2')
```

Some Useful Commands for Plotting

```
plot(x-coordinates, y-coordinates, formatting)
title('Insert Desired Title for Plot')
xlabel('Insert label for x-axis')
ylabel('Insert label for y-axis')
legend('Plot1 Label','Plot2 Label', …)
grid   % Adds a grid
close  % Closes the current figure window
figure % Creates a new figure window
subplot(m,n,k) %Subdivides a figure window into m by n
subwindows & points to the kth subwindow
axis([xmin xmax ymin ymax]) %Set axis scale
hold on  %Holds current plot on & allows add-ons
hold off  % Turns off the hold
```

Your Turn ...

Try these commands (one at a time) in MATLAB. Explain what each command does.

```
>> t = -4:0.001:4;
>> y1 = t.^2;
>> plot(t,y1); xlabel('t')
>> close

>> y2 = (t-1).^2;
>> plot(t,y1,t,y2); legend('t^2','(t-1)^2');
>> close

>> subplot(2,1,1);plot(t,y1);title('t^2');
>> subplot(2,1,2); plot(t,y2);title('(t-1)^2');
```

Test Your Understanding

ENMC3111
Computer Aided Engineering

Lecture #3

Ihab Abu Ajamieh, PhD.

Input and Output Statements

Overview

- Input
  - Input statements
  - Menu statements
- Output
  - Display statements
  - fprintf statements
  - Message boxes

Entering Inputs into a Program

There are a variety of ways to enter input variables into a MATLAB program. Three methods will be introduced initially:

1. Use ***hard-coded variables*** in the MATLAB program.
2. Use ***input statements*** within the program to prompt the user in the MATLAB command window to enter input variables.
3. Use ***menu statements*** within the program to prompt the user to click on a button to select an input.

Using Hard-Coded Variables

- So far, whenever we need a value in our program, we have simply created a variable:

```
x = 5;
pizza = 'pepperoni';
```

- Why not use hard-coded variables all the time?
  - Does not allow the user to choose different values
  - Program is only useful for one specific case
  - You might not know the needed values beforehand

Example #1

- Assume we write a script (circle_properties.m) with the following commands:

```
radius = 5;
circumference = 2*pi*radius;
area = pi*radius^2;
```

- What if we want to check a new circle with radius 10?

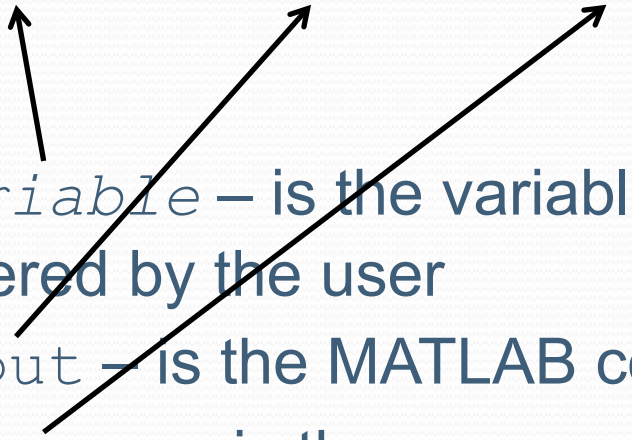
```
radius = 10;
circumference = 2*pi*radius;
area = pi*radius^2;
```

- What if you don't know the value (now)?

Using Input Statements for Numbers

- One way to allow the user to choose a value is to use an input statement:

```
variable = input('message');
```



where

- *variable* – is the variable in MATLAB where you wish to store the value entered by the user
- input – is the MATLAB command used
- *message* – is the message you want to display to the user

Example #2

- Assume we write a script (circle_properties.m) with the following commands:

```
radius = input('Please enter the radius: ');
circumference = 2*pi*radius;
area = pi*radius^2;
```

- When we run this script:

```
Please enter the radius: 5
```

  - Value of `5` is stored into variable `radius`
  - `circumference` and `area` are computed as normal

Using Input Statements for Strings

- What happens if you want to allow the user to enter a letter or a word instead of a number?

```
>> letter = input('Please enter a letter: ');
Please enter a letter: a
Error using input
Undefined function or variable 'a'.
```

- Add the second argument to the input statement to specify that the value will be a string:

```
variable = input('message', 's');
```

Using Menu Statements

- A menu statement can be used to allow the user to select from a set of predefined choices
- When you use a menu statement, MATLAB will create a new window with a set of buttons
- A menu statement takes the following form:

```
variable = menu('message', 'opt1', 'opt2',…);
```

where:

- *variable* – is the variable in MATLAB where you wish to store the button pressed by the user
- menu – is the MATLAB command used
- *message* – is the message you want to display to the user
- 'opt1','opt2',… – define the number of buttons and the text displayed on them

Using Menu Statements

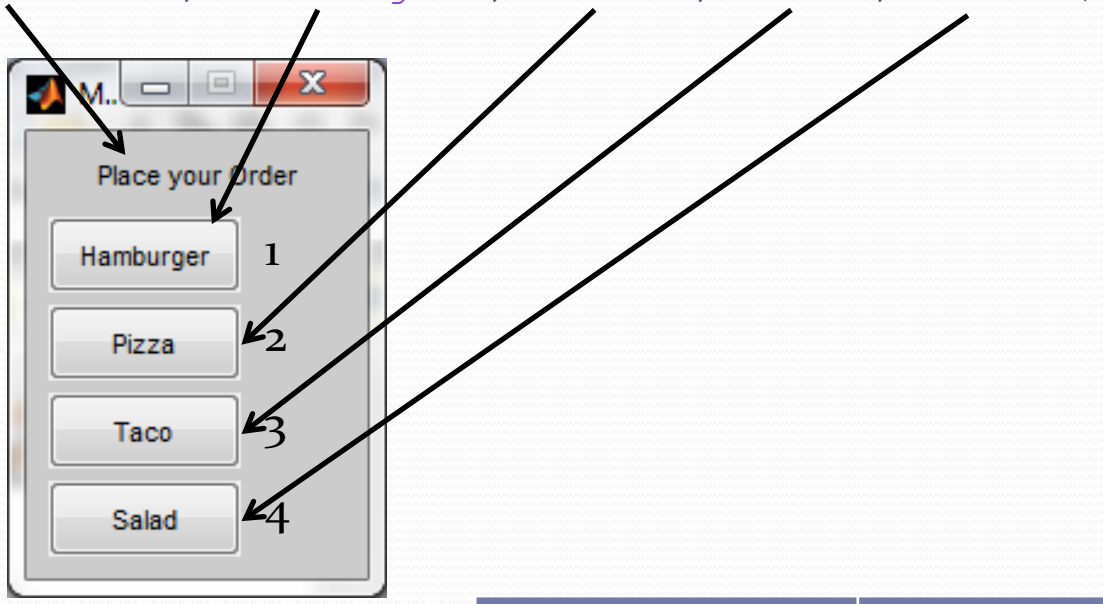
- What value is stored in the variable when the user presses one of the buttons?

```
variable = menu('message', 'opt1', 'opt2',…);
           (1)             (2)
```

  - The order you list the options in your message statement determines the order the buttons are displayed
  - The first button is assigned a value of 1, the second a value of 2,…
  - When the user presses a button the value of the button is stored

Example #3

```
>> lunch = menu('Place your Order','Hamburger','Pizza','Taco','Salad')
```



- When a button is pressed, the value of the button is stored into the variable `lunch`

Button	Value of `lunch`
Hamburger	1
Pizza	2
Taco	3
Salad	4

Displaying the Outputs of a Program

There are a variety of ways to display and store the output variables of a MATLAB program. **Four methods** will be introduced initially:

1. Do not use ***suppression*** (;) for a given statement in the MATLAB program.
2. Use ***display statements*** (disp) within the program that will display the desired outputs in the MATLAB command window.
3. Use ***fprintf statements*** within the program that will display the desired outputs in the MATLAB command window and/or write the resulting outputs to a file.
4. Use ***message boxes*** within the program that will open a new window to display a message to the user.

Unsuppressing Statements

- The easiest way to display a value in MATLAB is to not suppress it
  - When you place a `;` at the end of a line in MATLAB, you are telling MATLAB that you do not want to see the result
  - If you leave the `;` off, MATLAB will show you the result

```
>> x = 10+5;
>>

>> x = 10+5

x =

 15

>>
```

- This works in a script as well.
  - If you leave off a `;` in a script, the result will be displayed in the command window

Using Display Statements

- Often, we want to add some additional information to the value we display:
  - Message describing a value
  - Units
- You can display anything you want to the command prompt using the `disp` command:

```
disp(constant or variable);
```

where

- `disp` – is the MATLAB command used
- *`constant or variable`* – is the thing you want to display

Example #4

- Going back to our circle example, we can now add a couple of lines to display the results:

```
radius = input('Please enter the radius: ');
circumference = 2*pi*radius;
area = pi*radius^2;
disp('Circumference:'); disp(circumference);
disp('Area:'); disp(area);
```

- Running the script now, we would see:

```
Please enter the radius: 5
Circumference:
   31.4159

Area:
   78.5398

>>
```

Using fprintf Statements

- If we want additional formatting, we can use the `fprintf` statement
  - Mix strings with values
  - Specify the number of decimal places used
- `fprintf` statement takes the following form:

```
fprintf('Formatted Message', val1, val2,…);
```

where

- `fprintf` – is the MATLAB command used
- *`Formatted Message`* – is the message you want to display with formatting symbols
- *`val1, val2,…`* – are the values or strings that you want to insert into your formatted message

Using fprintf Statements

Insertion Formats

- `%s` – inserts a string
- `%i` – inserts an integer number
- `%f` – inserts a floating point (decimal) number

Escape Characters

- `\n` – moves to the next line
- `\t` – inserts a tab
- `\\` – displays the `\` character
- `%%` – displays the `%` character
- `''` – displays the `'` character

- When using `%f`, there are some additional settings you can use

  `%W.Pf`

  where W is the number of characters used to display the value and P is the number of decimal places displayed

Examples #5

```
>> day = 'Wednesday'; weather = 'sunny'; temp = 75;

>> fprintf('Today is %s \n',day)
Today is Wednesday

>> fprintf('The weather today is %s and the temperature
is %i \n', weather,temp)
The weather today is sunny and the temperature is 75

>> fprintf('The weather today is %s and the temperature
is %0.3f \n',weather,temp)
The weather today is sunny and the temperature is 75.000

>> fprintf('The weather today is %s and the temperature
is %0.3f \n',temp,weather)
The weather today is K and the temperature is 115.000
```

Using Message Boxes

- If we want to output a message outside of the command window, we can use a message box
  - A message box creates a new window (similar to a menu) which displays a message
- Message boxes take the following form:

```
msgbox('Message');
```

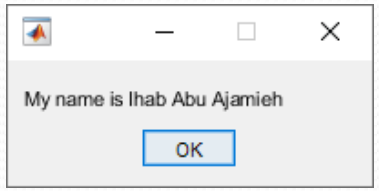
where

- `msgbox` – is the MATLAB command used
- *`Message`* – is the message you want to display to the user

Examples #6

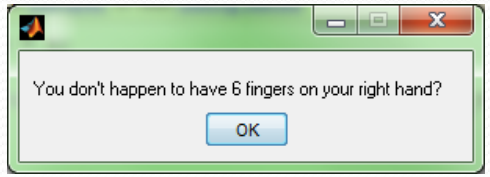
- Simple message box:

```
>> msgbox('My name is Ihab Abu Ajamieh');
```



- Constructing strings for message boxes

```
number_of_fingers = 6;
hand = 'right';
message = ['You don''t happen to have ' num2str(number_of_fingers) fingers on your ' hand ' hand?'];

msgbox(message);
```



Test Your Understanding

ENMC3111
Computer Aided Engineering

Lecture #4

Ihab Abu Ajamieh, PhD.

Conditional Statements

Conditions and
if, if-else, and if-elseif-else

Outline

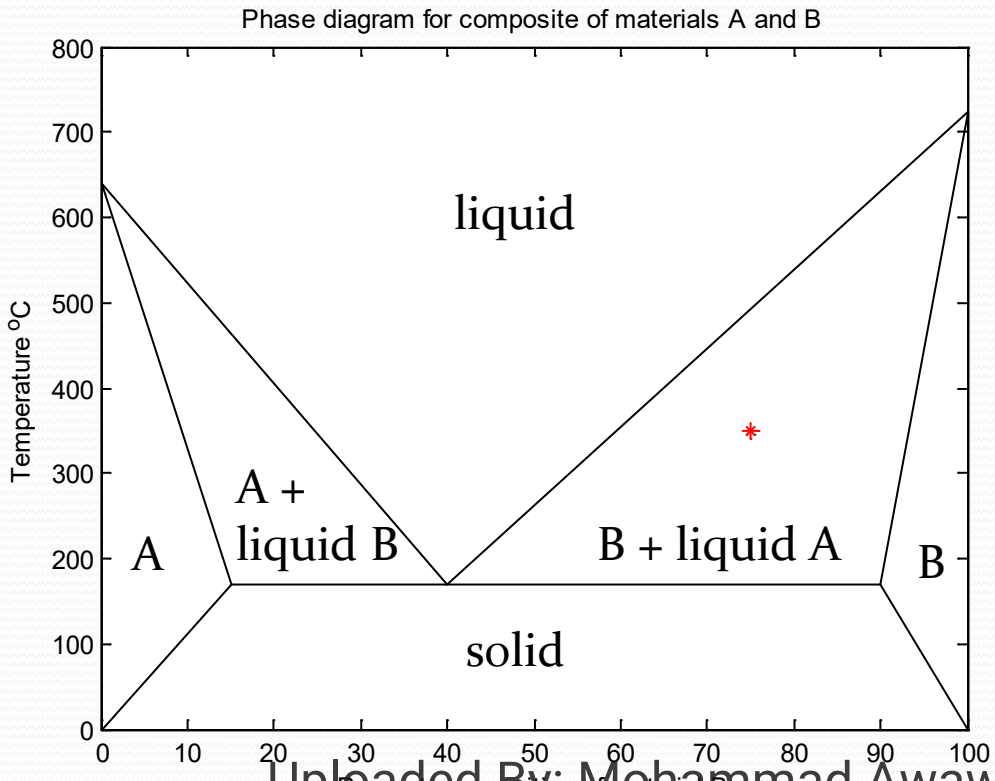
- Motivation
- Conditional Operators
  - Boolean Logic
  - Relational Logic
- Conditions
- Conditional Structures
  - if structures
  - if-else structures
  - if-elseif-else structures

What can you do?

- Based on what we've learned in the class so far, what can you do with MATLAB?
  - Create scripts
  - Input data
  - Output data
  - Perform mathematical computations/modeling
  - Create graphs and figures

What else do you need?

- Can you solve the following problem, given what you know in MATLAB so far?
  - Produce a phase diagram for a composite material, place a point on the diagram showing where a given material would exist, and ***return a message to the user about the state***



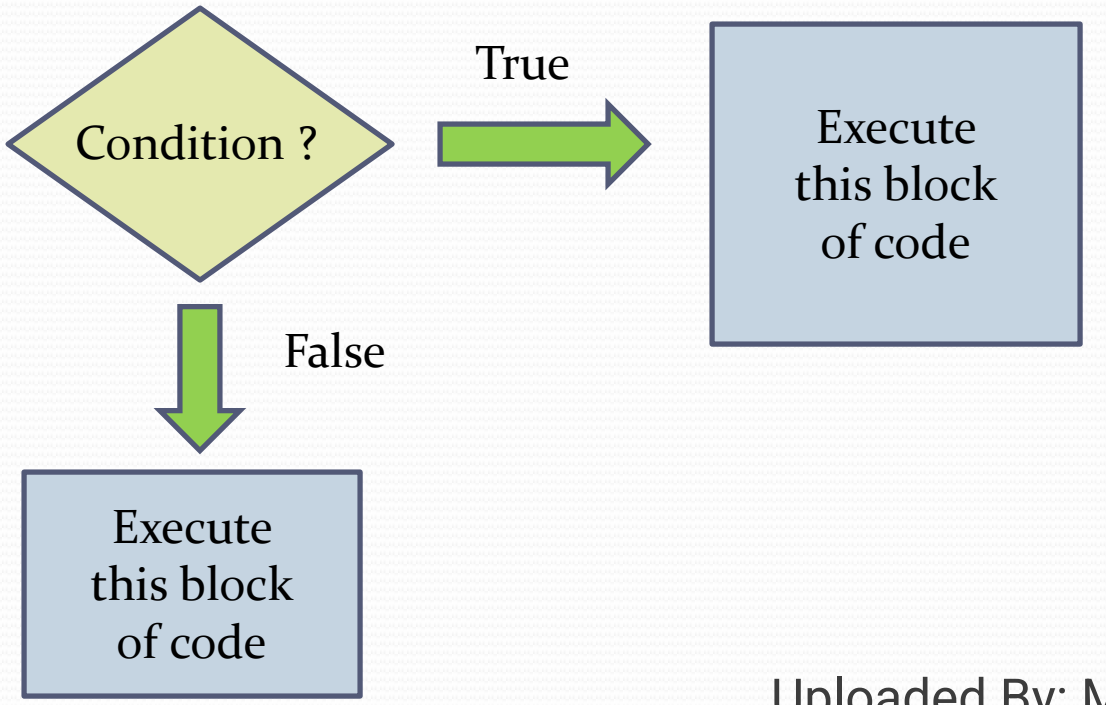
What else do you need?

- In order to solve the last part of the previous problem, we need a way to allow the computer to make decisions

- Conditional statements allow us to ask yes or no questions and decide between two courses of action:
  - Does this data point fall within a given range?
  - Does the user want to convert a temperature from Celsius to Fahrenheit?
  - Is there an error in the type of data the user entered?

Conditional Statements

- Conditional statements control when a section of program code executes and when it does not execute.
- All programming languages use conditional statements but syntax will vary among languages.



Relational and Logic Operators

Relational Operator	Description	Logical Operator	Description
==	Equal	&&	Scalar Logical AND
~=	Not equal	||	Scalar Logical OR
<	Less than	&	Element by Element AND
>	Greater than	|	Element by Element OR
<=	Less than or equal to	~	Logical NOT
>=	Greater than or equal to	xor	Logical Exclusive OR

Compare numbers

Compare true/false values

Logical Operators

Truth Tables for AND, OR, and NOT (~)

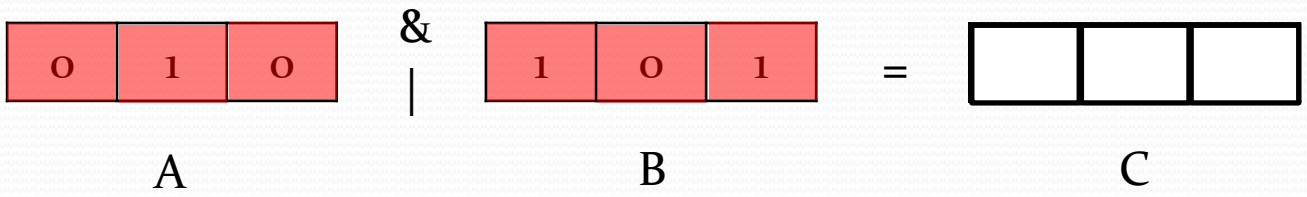
A	B	A && B	A || B	~A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

A = expression1 B = expression2

1 = TRUE 0 = FALSE

Element-wise Operators

- The single & and | are also called element-wise operators because they act like the . in front of an operator: ANDing or ORing the elements of a vector or array:



Relational and Logical Operators

- It is possible to string several relational and logical operators together. It is a good idea to use parenthesis and to understand the order of operation (just like with arithmetic operators).

Operator(s)	Operation	Priority
>, <, >=, <=, ==, ~=	Relational operators	Highest
~	NOT	
&	Elementwise AND	
|	Elementwise OR	
&&	Short-circuit AND	
||	Short-circuit OR	Lowest

*** All of these operators have lower priority than mathematical operators**

Logical Operator Examples

- If `a = true`, `b = true`, `c = false`, what do the following expressions result in:

`a && b`	`1 (True)`
`a && c`	`0 (False)`
`a || b`	`1 (True)`
`c || b`	`1 (True)`
`~b`	`0 (False)`
`a && ~c`	`1 (True)`
`(a || b) && ~c`	`1 (True)`
`~(a && b) || (a && c)`	`0 (False)`

Relational Operator Examples

- If `a = 2`, `b = 3`, `c = 5`, what do the following conditional expressions result in:

`a > b`	`0 (False)`
`a ~= c`	`1 (True)`
`a <= b`	`1 (True)`
`c == a + b`	`1 (True)`
`(c > a) && (c > b)`	`1 (True)`
`(c > a) || (a > b)`	`1 (True)`
`(a <= b) && ~(c == 5)`	`0 (False)`
`~((a < b) || (b < c))`	`0 (False)`

Cautions: the == operator

It is very important to understand the difference between = (an assignment operator) and == (a relational operator)

```
>> x = 10
% Creates a variable, x, and assigns 10 to that variable.

>> x == 10
% Checks to see if the variable x is equivalent to 10
```

The statement x == 10 will produce an error if x is not defined in the workspace or program; a 1 (TRUE) if x is indeed defined and equal to 10; and a 0 (FALSE) if x is defined but not equal to 10.

Cautions: complex conditions and relational operators

$$15 < A < 25$$

We interpret this expression to be true only if A is between 15 and 25.

In MATLAB, the expression: 1 **OR** 0 < 25 is always true for any A! This can create serious problems in your code!

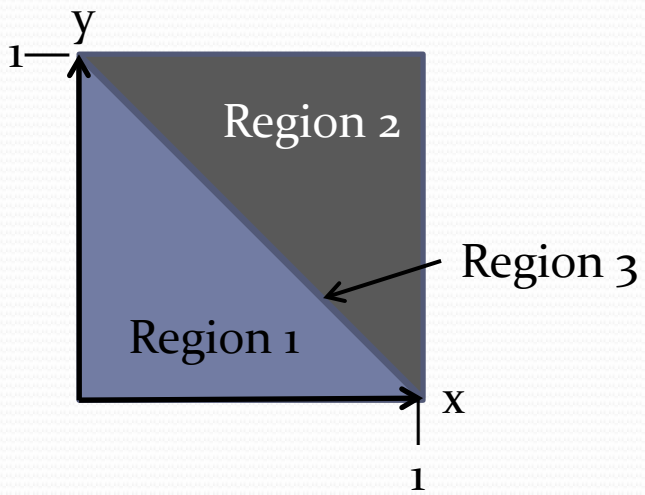
In MATLAB, we must split the expression apart with a logical operator as follows:

$$15 < A \;\&\&\; A < 25$$

Then the expression will be true only if A lies between 15 and 25.

Building Conditions

- What is the condition to test if a point (x,y) is in:
  - Region 1 `y < 1 - x`
  - Region 2 `y > 1 - x`
  - Region 3 `y == 1 - x`
  - Not in Region 2 `~(y > 1 - x)` OR `(y < 1 - x) || (y == 1 - x)`



if … Structures

```
if   expression == true

     MATLAB commands

end

MATLAB commands
```

The expression is the condition being checked. Examples:

```
if a == 5
if  b ~= 0
if  a == 5 || b ~= 0
if  4 < a && a <= 12
```

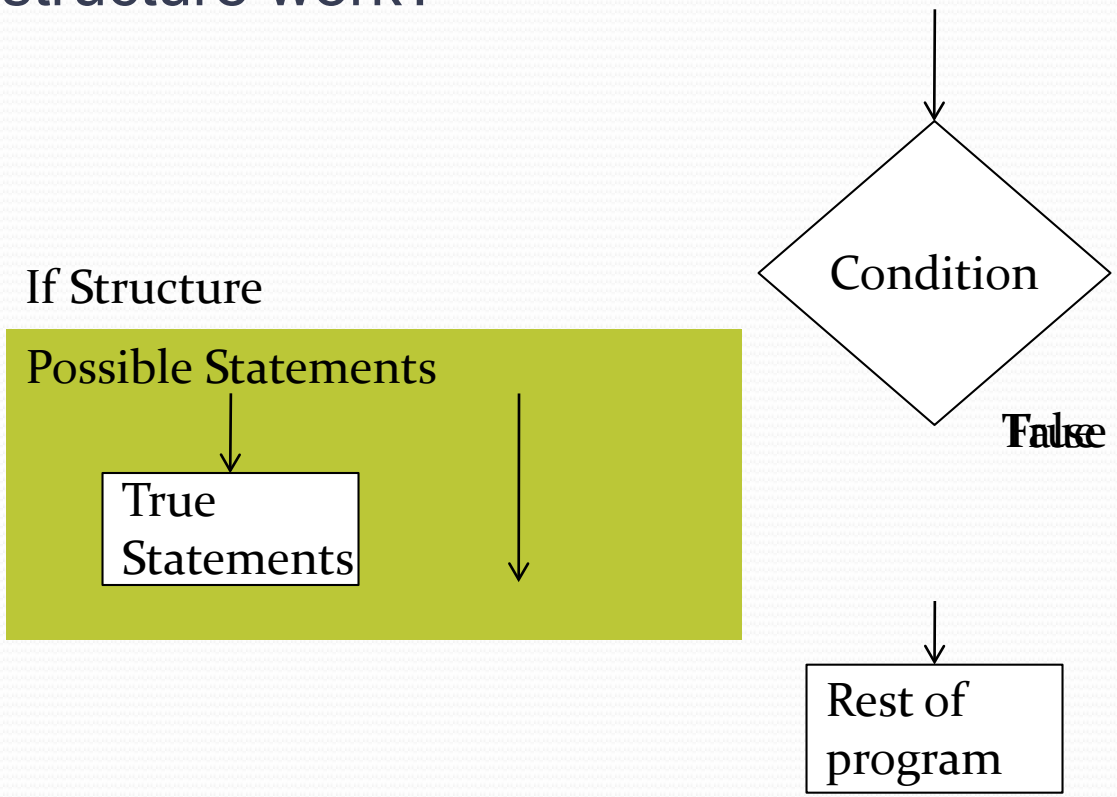
If the expression is true, then the MATLAB commands following the if statement will run.

Specifies where the end of the MATLAB commands in the if statement are

If the expression is false, then the program will skip to the MATLAB commands following the end statement

if … Structures

- How does the if structure work?



if … else … Structures

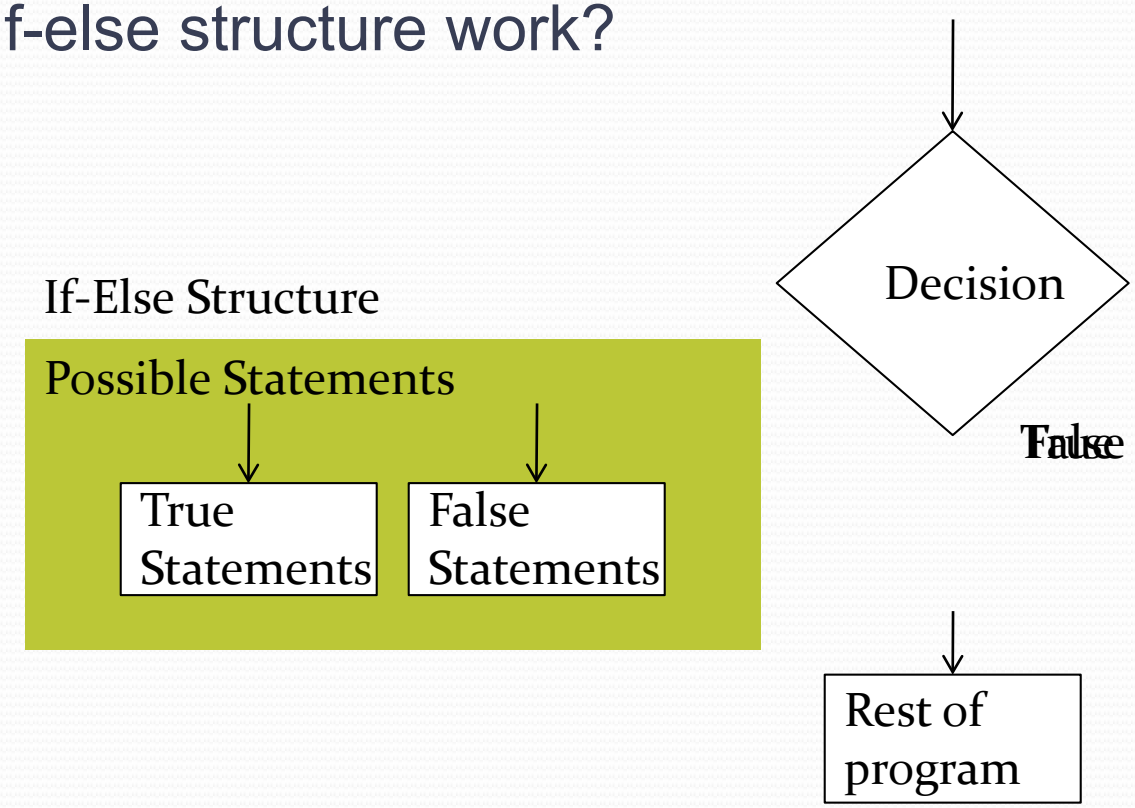
```
if    expression == true

      MATLAB commands

else "expression == false"
      MATLAB commands

end

MATLAB commands
```

If the expression is true, then the MATLAB commands following the if statement will run.

If the expression is false, the MATLAB commands following the else statement will run

if … else … Structures

- How does the if-else structure work?



You can only execute either what is in the true case or what is in the false case during a single run of a program

if … elseif … else … Structures

```
if   expression1
     MATLAB commands

elseif expression2
     MATLAB commands


else   No condition!!!!
     MATLAB commands
end
MATLAB commands
```

If expression1 is true, then the MATLAB commands following the if statement will run. Program then jumps to end.

If the expression1 is false but expression2 is true, the MATLAB commands following the elseif statement will run. Program then jumps to end.

If neither of the expressions are true, then the MATLAB commands following the else statement will run. Note: you don't have to include an else statement here if it isn't needed for your code.

Comments on if … statements

- Multiple **elseif** statements can be included within an **if** statement
- **if** statements can be nested within other **if** statements
- Don't include a condition after an ***else*** statement. This makes no sense: ***else*** 0 <= a && a < 10
- An else executes automatically if everything above it is false.

```
answer = menu('What day is today?', 'a) Mon', 'b) Tues', 'c) Wed',
              'd) Thur', 'e) Fri');
if answer == 1
    disp('Today is not Monday! ');
elseif answer == 2
    disp('Today is not Tuesday! ');
elseif answer == 3
    disp('That is correct! ');
elseif answer == 4
    disp('Today is not Thursday! ');
elseif answer == 5
    disp('Today is not Friday! ');
else
    disp('Are you sure you entered a day? ');
end
```

```
temp = input('What is the outside temperature? ');
precip = menu('Is it precipitating outside? ', 'Yes', 'No');
if temp >= 50
    if precip == 2
        disp('Today is a nice day, go outside! ');
    else
        disp('It is an ok day, but you're going to get wet! ');
    end
else
    disp('Stay inside! ');
end
```

MATLAB®

Introduction to Loops

For Loops

Motivation for Using Loops

- So far, everything we've done in MATLAB, you could probably do by hand:
  - Mathematical operations
  - Graphing
  - Logical operations
  - Making decisions based on data

- What if I were to ask you to perform a computation
  - 10 times?
  - 100 times?
  - 1,000,000 times?
  - Until you find the number I want you to get?

What is a Loop?

A loop is a block of code that repeats itself

A loop can be programmed to repeat a fixed number of times or it can be programmed to repeat until some condition is no longer met

Counter Based Loop	Conditional Based Loop
For Loop	While Loop

Loop Type Comparison

- Counter Based
  - Runs a predetermined number of times
  - Uses a counter variable
  - Automatically updates the loop variable
- Conditional Based
  - Runs until some condition is met
  - Uses any variable(s)
  - Loop variable(s) must be updated manually

Both are based on a condition, it is simply that the **endpoint** of the counter based loop is known ahead of time (unless you make a mistake!), whereas the conditional based loop is not

Which loop type would you use?

- You want to calculate the p… d on the initial velocity, angle, and time since it was launched, stopping when the object comes to rest. **Conditional Loop**
- You want to calculate the pos… ased on the initial velocity, angle, and time since it was launched, for the first 5 seconds of its trajectory. **Counter Loop**
- You need to create a set of 10… mulate measurements that your new sorting machine might e… g algorithm you've developed. **Counter Loop**
- You want to collect perform… ,000,000 revolutions, unless the measurements indicate… h case you need to turn off the engine. **Conditional Loop**
- You need to ask the user of … r, but your program cannot continue unless the value is… **Conditional Loop**

General Items About Loops

- Loop condition is only checked once during each iteration
- Conditional test is based on the ***loop control variable(s)***:
  - Initialized prior entering the loop
  - Updated within the loop
  - When the loop control variable(s) take on value(s) that cause the condition to fail, the loop will exit and continue on with the rest of the program
- The code that is repeated by the loop is called the ***loop body***
- Each time the loop repeats is called an ***iteration***

FOR loops

FOR loops are counter based loops

Sample Construction:

```
for control_variable = begin:step:end
     MATLAB statements
end
```

- The control variable will count from the beginning value to the end value based on the step size specified
- It will repeat the statements included inside the loop (between the **for** and **end**) each time it counts
- Once the control variable is outside of the specified range, the loop will stop repeating

FOR loops

FOR loops are counter based loops

Sample Construction:

```
for k = 1:1:10
        MATLAB statements
end
```

- k is set to 1
- All the MATLAB statements are executed
- k is incremented to 2
- All the MATLAB statements are executed a second time

⋮

- k is incremented to 10
- All the MATLAB statements are executed a 10^{th} time
- k is incremented to 11
- The value of k no longer meets the requirements and the loop terminates

FOR loops

Another Sample Construction:

```
for k = 18:-2:0
        MATLAB statements
end
```

How many times will this loop execute?

Answer: 10 times
(k = 18, 16, 14, 12, 10, 8, 6, 4, 2, 0)

Note: It make absolutely no sense to set up the loop this way unless your program makes some clever use of k that requires it to decrement by 2

Example 1: Simple FOR Loop

```
sum=1;
product = 2;
for k=1:3
    sum = sum + 0.5^k;
    product = product*2;
end
```

k	sum	product

Example

```
for k = 1:1:5
   fprintf('%i \n', k);
end
```

The loop control variable, k, is

- initialized to: 1
- tested against the stop value: 5
- changed by each iteration

Output:

```
1
2
3
4
5
```

Example

```
for k = 5:-1:1
   fprintf('%i', k);
end
```

Output:

The loop control variable, k, is

- initialized to: 5
- tested against the stop value: 1
- changed by each iteration -1

Example

```
n = 10;
s = 2;
for k = 1:s:n
   fprintf('%i \n', k);
end
```

The loop control variable, k, is

- initialized to: 1
- tested against the stop value: 10
- changed by 2 each iteration

Output:

```
1
3
5
7
9
```

Example

```
for k = 1:0.5:5
   fprintf('%0.1f \n', k);
end
```

The loop control variable, k, is

- initialized to: 1
- tested against the stop value: 5
- changed by each iteration 0.5

Output:

```
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
```

Comments on FOR loop

- MATLAB automatically updates the loop control variable (index for the loop – k in our examples)
- Don't try to change the loop control variable inside the loop; you wouldn't want to put the line k = 7 in the loop – it won't work
- You can use the loop control variable to determine other variable values
- Don't use i and j as index variables in MATLAB because i and j are sqrt(-1)
  - Many other languages commonly use i and j as index variables, but don't use them here!

Test Your Understanding

ENMC3111
Computer Aided Engineering

Lecture #5

Ihab Abu Ajamieh, PhD.

MATLAB®

While Loops

Why While Loops?

- Consider the following situation:
  - Someone has been stealing your cookies! To figure out who, you are writing a script to act as a security system for your room. You need to take a measurement from a motion sensor every 10 seconds and take a picture if the motion sensor detects someone near the cookie jar.

- Can you write this program using a For loop?
  - No, you don't know if/when someone will try to steal your cookies!

While Loops

WHILE loops repeat until some condition is no longer met

Construction:

```
while  conditional expression
       MATLAB statements
end
```

- As long as the expression is true, the MATLAB statements in the while loop will continue to execute
- If the expression is initially false, the while loop will never execute
- If the expression becomes false, the loop will terminate

While Loops

Example1:

```
while  a >= 0
       MATLAB statements
end
```

- (a) has to be defined in the program before you hit the while loop, otherwise the program won't run.
- If (a) is negative, the while loop will not execute at all.
- If (a) is positive, the while loop will continue to execute as long as it stays positive

What happens if (a) is positive and none of the statements in the loop ever change (a)?

Infinite Loop!

Example 2: Simple WHILE Loop

```
sum = 50;
while sum < 100
    sum = sum + 8;
end
disp(sum)
```

```
Final Value for sum?
        106
# Times thru loop?
         7
```

sum	Loop?

Common Misconception about WHILE Loops

Many first-time programmers think that the condition for the while loop is checked continually as the loop executes and as soon as the condition is false, the loop immediately terminates – **this isn't true**

- If the while condition is true, the loop will completely execute to the **end** statement, even if the condition becomes false somewhere in the middle of the loop
- Once the loop has reached the **end** statement, the condition will be re-evaluated to determine whether or not the program should go through the loop again

Examples3

- What is the output of the following code?

```
x = 1;
while x < 10
    fprintf('%d \n', x);
    x = x+1;
end
```

Output:

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

Examples4

- What is the output of the following code?

```
x = 1;
exitFlag = true;
while exitFlag
    fprintf('%d \n', x);
    if x > 10
        exitFlag = false;
    end
    x = x+1;
end
```

Output:

```
1
2
3
4
5
6
7
8
9
10
11
```

Examples5

- What is the output of the following code?

```
x = 1;
while x < 10
    if (x < 4 || x > 7)
        fprintf('%d', x);
    end
    fprintf('\n');
    x = x+1;
end
```

Output:

```
1
2
3
8
9
```

Examples6

- What is the output of the following code?

```
x = 1;
while (x > 1)
    fprintf('%d \n', x);
    x = x-1;
end
```

Output:

```
Nothing!!!
```

Examples7

- Using a while loop, you can add error checking to your program to ensure the user enters values correctly
  - Require the user to enter a number between 1 and 10 before continuing:

```
x = 0;
while (x < 1) || (x > 10)
  x = input('Please enter a number: ');
  if (x < 1) || (x > 10)
      fprintf('Number must be between 1 and 10.\n');
  end
end
```

MATLAB®

Nested Loops and the Break Statement

What are Nested Loops?

- Nested loops are:
  - Loops which run inside another loop

- When would you use nested loops?
  - Performing the same set of operations on different data sets
  - Performing a set of computations with multiple independent variables
  - Working with 2-dimensional arrays

Nested For Loops

Nested for loops consist of one (or more) for loops inside of another:

```
for m = start:inc:end
      MATLAB Statements;
      for n = start2:inc2:end2
         MATLAB Statements;
      end
      MATLAB Statements;
end
```

- When nesting for loops:
  - Loop control variables must have different names
  - Inner loop will execute completely before the next iteration of the outer loop

Example1: Multiplication Tables

```
for m = 1:1:5
    for n = 1:1:5
        fprintf('%i x %i = %i\t',m,n,m*n);
    end
    fprintf('\n');
end
```

Nested While Loops

Nested while loops consist of one (or more) while loops inside of another:

```
while condition1
      MATLAB Statements;
      while condition2
         MATLAB Statements;
      end
      MATLAB Statements;
end
```

- When nesting while loops:
  - Loop control variables can be the same or different
  - Conditions can be the same or different

Example2: Guessing Game

```
repeat = 1;
while repeat == 1
    my_numb = randi(10,1);
    guess = 0;
    while guess ~= my_numb
        guess = input('Guess my number: ');
        if guess ~= my_numb
            fprintf('Try again.\n');
        end
    end
    fprintf('You found my number!\n');
    repeat = menu('Play again?','Yes','No');
end
```

Mix and Match

- It is possible to nest for loops inside of while loops and vice versa, depending on your application
  - If you do this, you must pay attention to which loop will execute at which time and keep track of your loop control variables

```
while more lines in file
    get next line from file
    for n = 1:number of characters in line
        count the number of a's
    end
    display the number of a's
end
```

Contingently Nested Loops

- It is also possible to have nested loops which depend on the outer loop to determine the number of times the loop will iterate

Example 3:

```
for m = 1:5
    for n = 1:m
        fprintf('*');
    end
    fprintf('\n');
end
```

Output:

```
*
**
***
****
*****
```

Break Statement

Another option for exiting a for loop or a while loop is to use a break statement.

break forces MATLAB to terminate the current loop,

Example 3:

```
sum = 1;
for k = 1:20
      sum = sum + 1/2^k;
      if (2 - sum) < 1e-5
            disp(k);
      break
      end
end
```

When k = 17, (2-sum) < 0.00001
so loop terminates.

Break Statement

Note about the Break statement:

- If there is a loop inside a loop (nested loop), the break statement will only affect the loop in which the break statement occurs

Example 4 & 5

```
for x = 1:5
    fprintf('%d\n',x);
    for y = 1:5
        if (x*y > 9)
            break;
        else
            fprintf('%d',y);
        end
    end
    fprintf('\n');
end
```

```
for x = 1:5
    fprintf('%d\n',x);
    for y = 1:5
        fprintf('%d',y);
    end
    fprintf('\n');
    if (x*y > 9)
        break;
    end
end
```

Test Your Understanding

ENMC3111
Computer Aided Engineering

Lecture #6

Ihab Abu Ajamieh, PhD.

MATLAB®

Solving Linear Equations

Solving Linear Equations

- Suppose for example, you have to solve the following linear equations for ‘x’ and ‘y’:

$$x + 2y = 6$$
$$x - y = 0$$

- There are two methods to solve the above-mentioned linear simultaneous equations:
  1. Use matrix algebra.
  2. Use the MATLAB command ‘`solve`’.

Solving Linear Equations

Matrix algebra:

- Representing the two equations in the matrix form, we get:

$$\begin{aligned} x + 2y &= 6 \\ x - y &= 0 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1 & 2 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 6 \\ 0 \end{Bmatrix} \quad \Rightarrow \quad AX = B$$

- Where:

  A is known as the coefficient matrix,

  X is called the variable matrix and

  B is the constant matrix.

Solving Linear Equations

Matrix algebra:

- To solve for X, we find the inverse of the matrix A, and then multiply the inverse by the matrix B, i.e.

$$X = A^{-1}B$$

```
% To solve two simultaneous linear equations.
A = [1 2;1 -1];
B = [6;0];
X = (inv (A))*B
```

```
X =
     2
     2
>>
```

Solving Linear Equations

Solve Command:

- The ‘solve’ command is a predefined function in MATLAB

```
syms x y
p = x+2*y == 6;
q = x - y == 0;
[x, y] = solve(p, q)

x =

2


y =

2

>>
```

Solving Linear Equations

Subs Command:

- The ‘solve’ command is a predefined function in MATLAB

```
syms x y
p = x+2*y == 6;
q = x - y == 0;
[x, y] = solve(p, q)

x =

2


y =

2

>>
```

Test Your Understanding