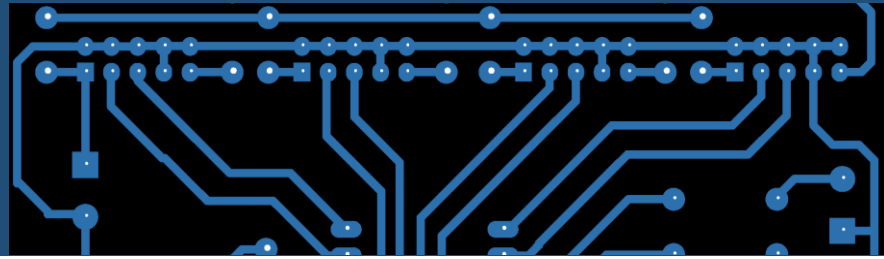
Introduction to MATLAB© Part III



Presented by: Prof. Eric Larkins (Coates A46)

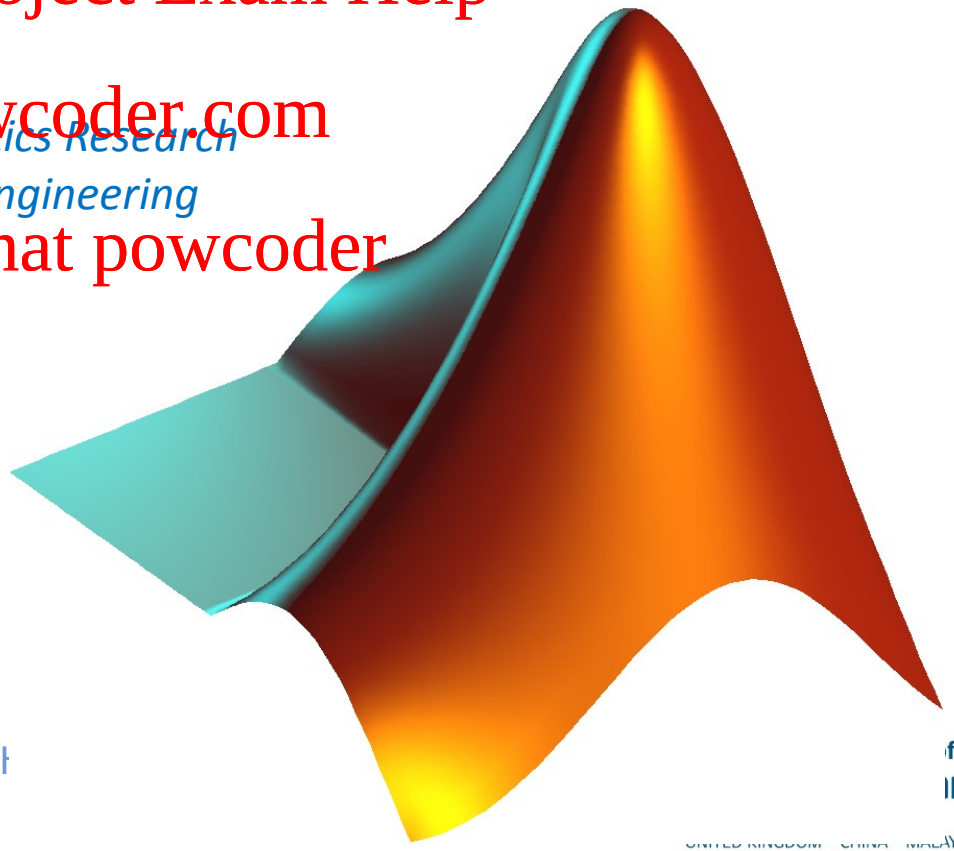
Optics and Photonics Group

Department of Electrical and Electronic Engineering

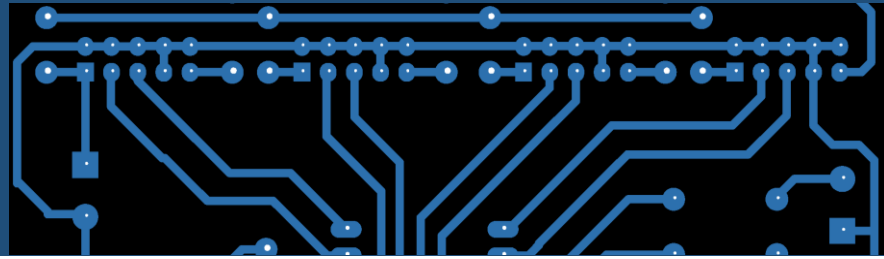
Prepared by: Dr. Steve Greedy

George Green Institute for Electromagnetics Research

Department of Electrical and Electronic Engineering



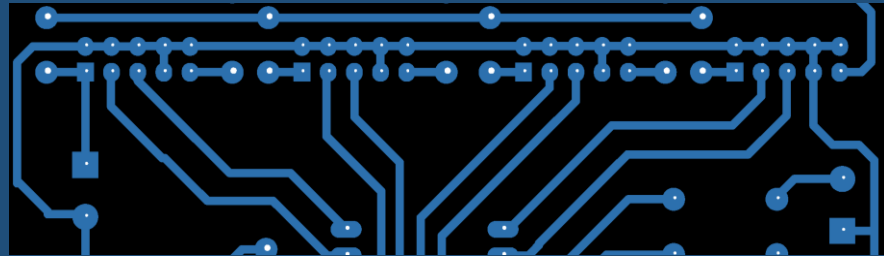
Introduction to MATLAB© Part III



Aims:

- Programming recap
- The m-files
  - Script files
  - Function files
- Animating your plots
- In class exercise – Simulation of a bouncing ball

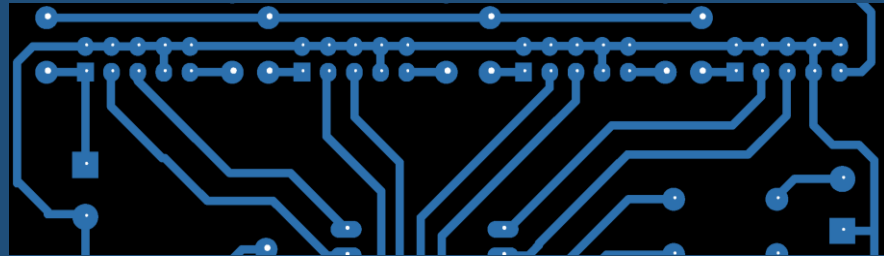
Introduction to MATLAB© Part III



Programming – Quick Recap

- Variables: int, float, double etc.
- Operators: =, +, -, * etc.
- Conditional statements **do** something **while a condition** is true or false
- Loops
  - To repeat a set of statements a **for** loop can be used.
- Designing your program: write program out in psuedo code.
  - Comments to make purpose and structure clear (aids debugging)
- Debugging your program: use the top down approach.
  - Isolate and investigate

Introduction to MATLAB© Part III



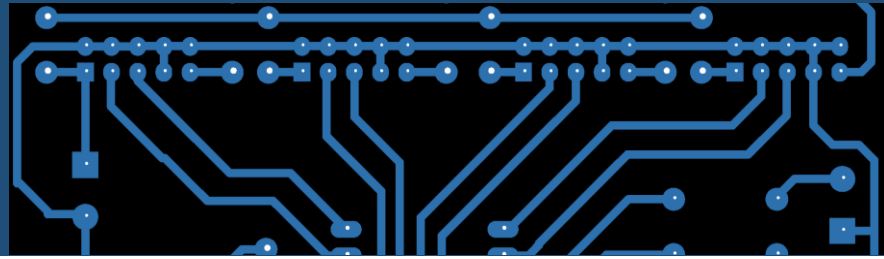
Script Files

- Allow you to save a sequence of commands in a single file
- For example:

```
a = 1;
b = 2;
c = a + b;
```


- Saving the file as **adda2b.m** it can then be run from the command line:

```
>>adda2b
c =
     3
```



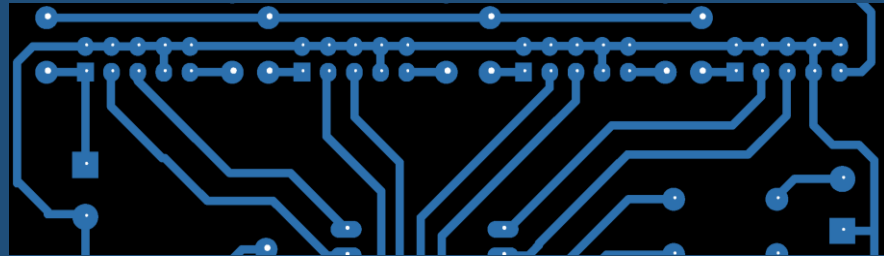
Function Files

- Allow you to pass arguments to a MATLAB program
- If you wanted to change the values assigned to **a** and **b** you would need to edit the script.
- However you could convert the **script** to a **function**
- For example:

```
function c = adda2b(a,b)
c = a + b;
end
```

- Saving the file as **adda2b.m** it can then be run from the command line with the necessary arguements:

```
>>adda2b(1,2)
c =
     3
```

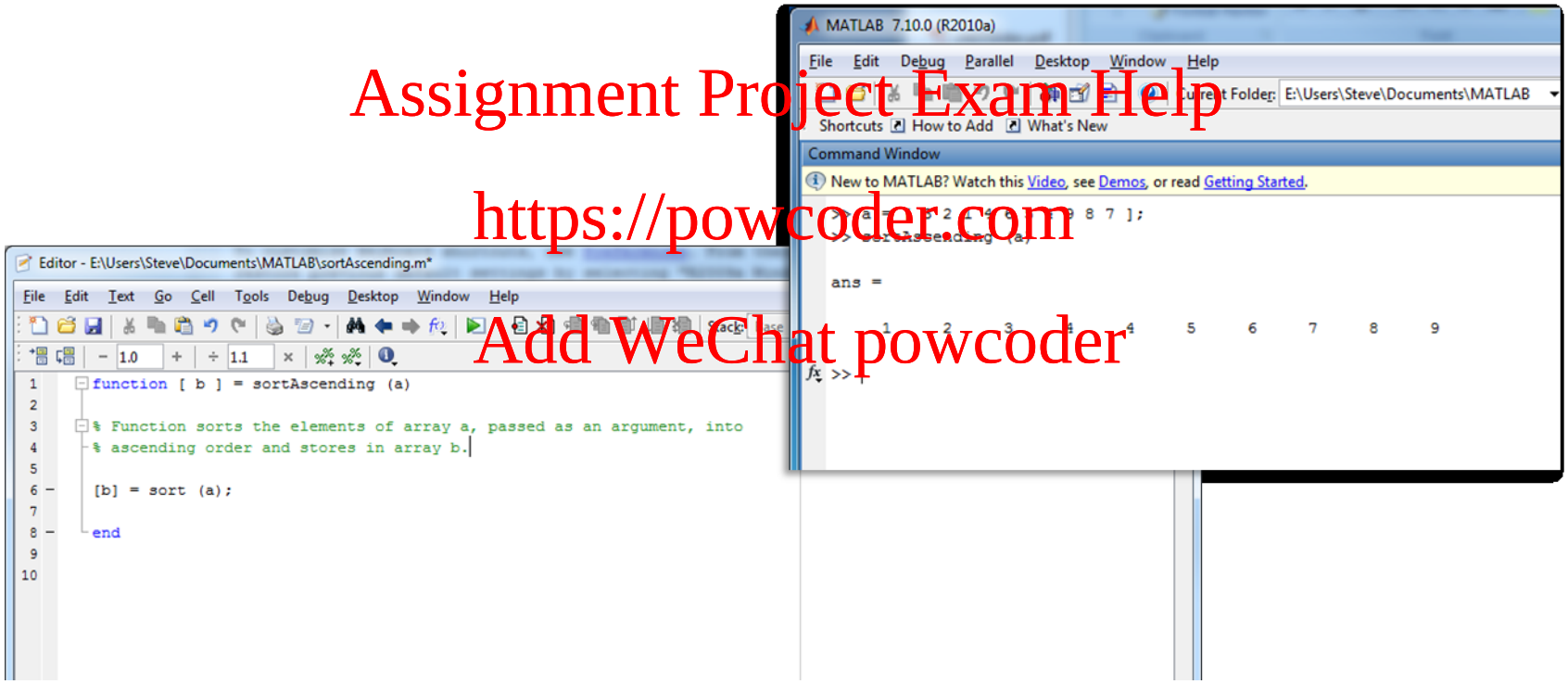


Function Files

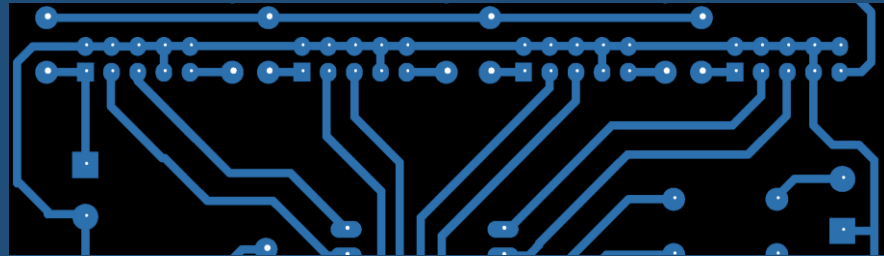
- Functions have input and output parameters
- Variables are local to the function in which they are declared.
  - You can re-use variable names in different function files
  - Compare with scripts where all variables exist within workspace

Introduction to MATLAB© Part III

- Example of a function file:



Introduction to MATLAB© Part III



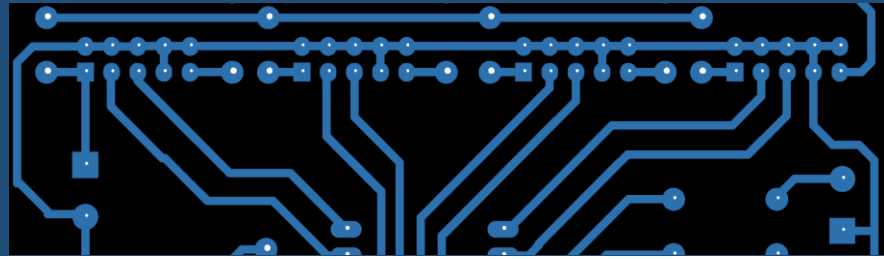
- **Analysis of the function file:**

```
function [ b ] = sortAscending (a)

% Function sorts the elements of arra
% argument, into ascending order an

[b] = sort (a);

end
```

Function definition. Starts with the function keyword.

The first parameter defines the output arguments, in this case an array **b**. Then the function name, **sortAscending**, followed by the input arguments, an array **a**.

Introduction to MATLAB© Part III



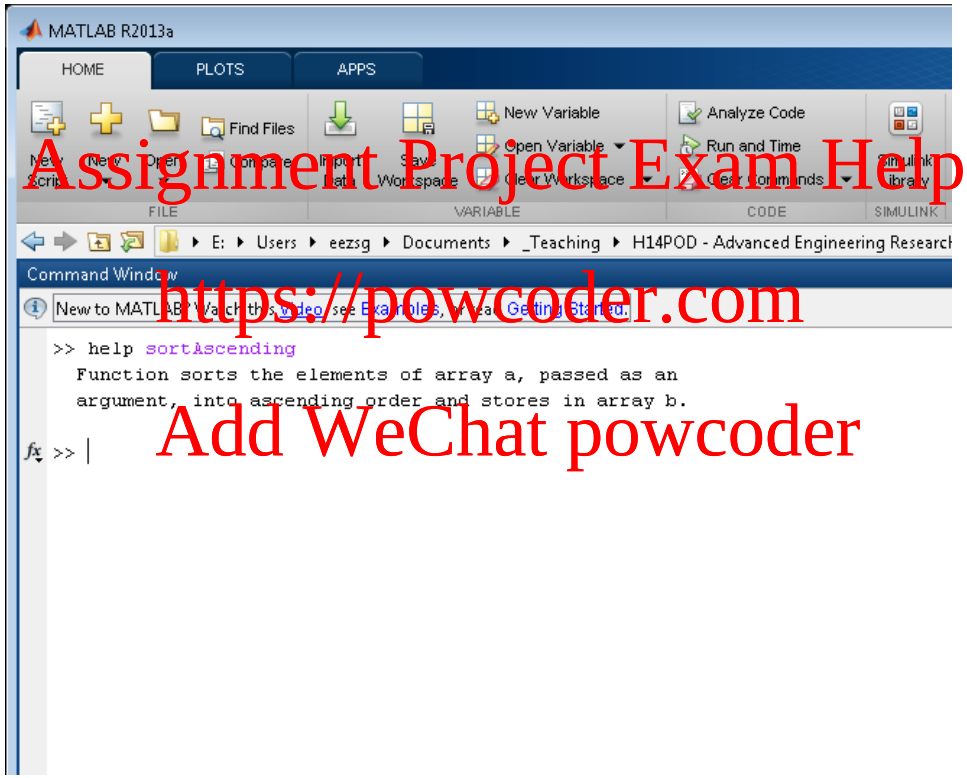
- **Analysis of the function file:**

```
function [ b ] = sortAscending (a)
% Function sorts the elements of array a, passed as an
% argument, into ascending order and stores in array b.

[b] = sort (a);

end
```

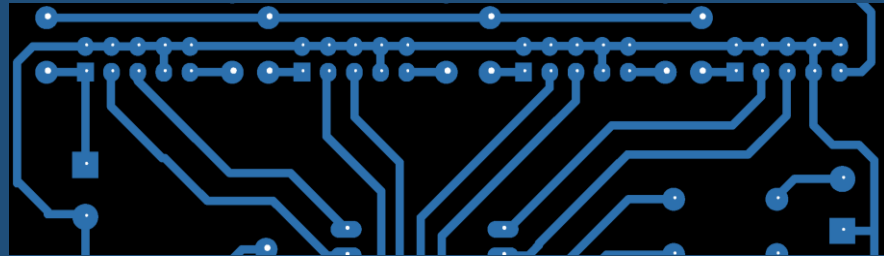

Comments, for a function it is useful to describe the input and output arguments and their type.

>>help sortAscending
will display these comments

Introduction to MATLAB© Part III



Introduction to MATLAB© Part III

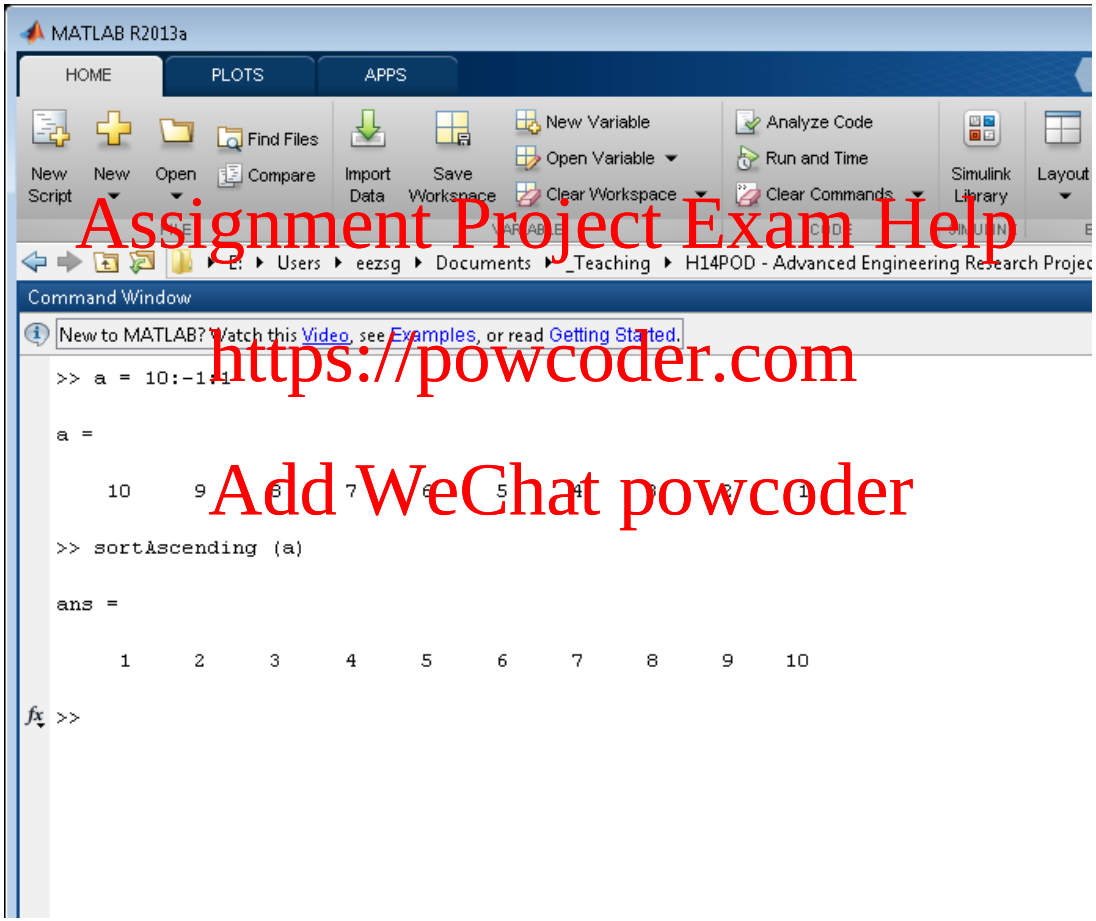


- **Analysis of the function file:**

```
function [ b ] = sortAscending (a)

% Function sorts the elements of array a, passed as an
% argument, into ascending order and stores in array b.

[b] = sort (a);

end
```


Call the MATLAB **sort** function to sort the elements of **a** into ascending order and store in a vector **b**

Introduction to MATLAB© Part III



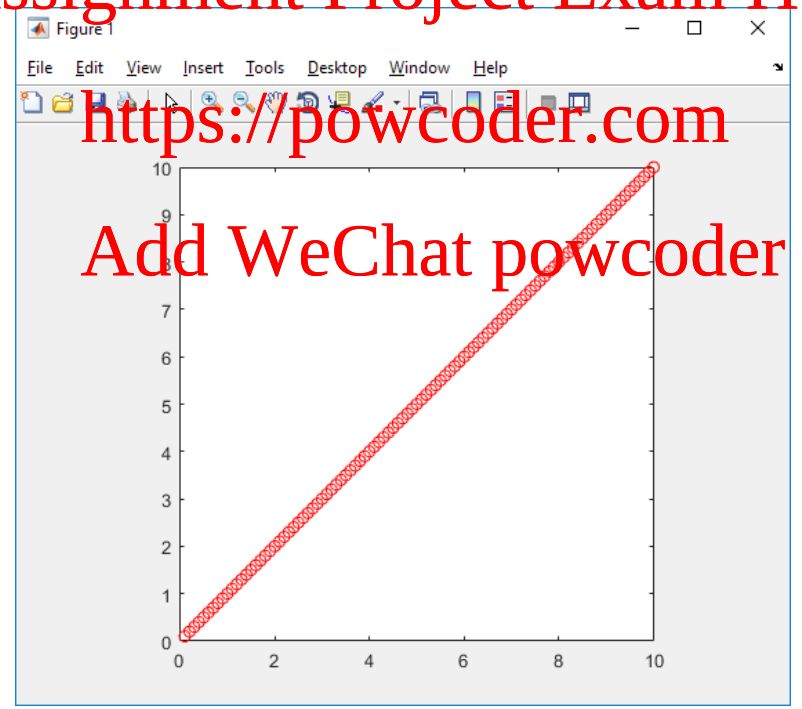
Animating Your Data

Animating Your Data

- As in stop motion video creation animations are created from a series of still frames

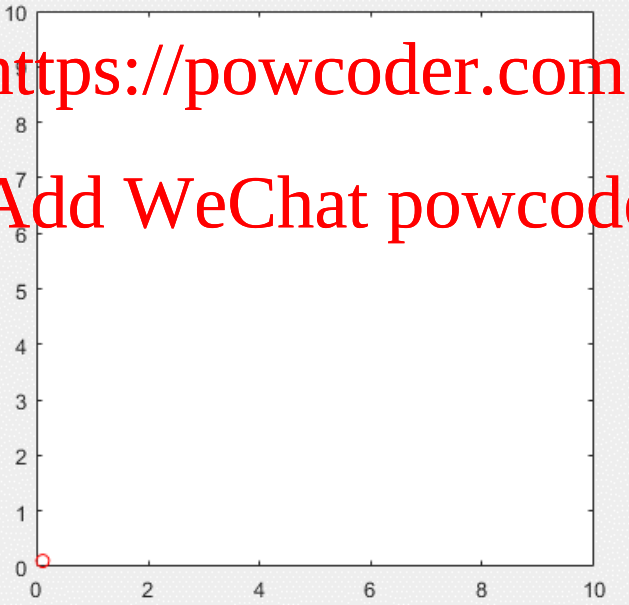
Animating Your Data

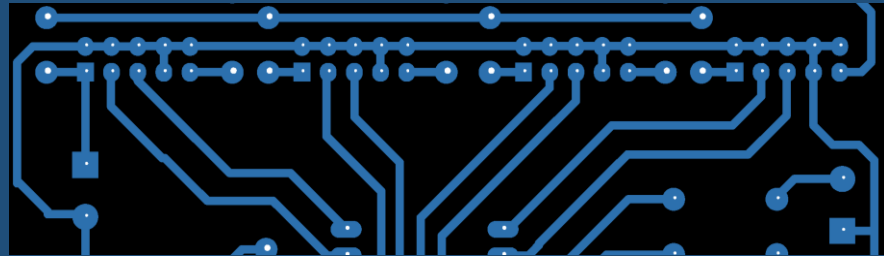
- As in stop motion video creation animations are created from a series of still frames



Animating Your Data

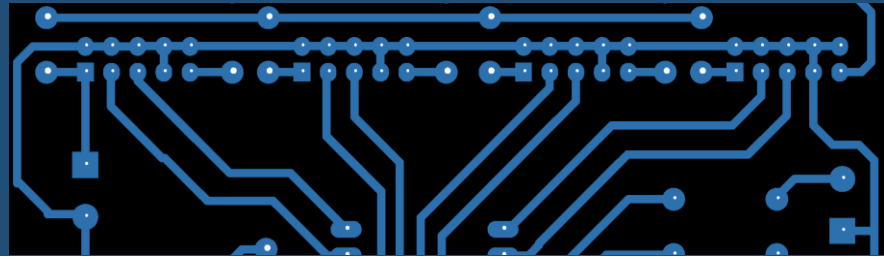
- As in stop motion video creation animations are created from a series of still frames





Animating Your Data

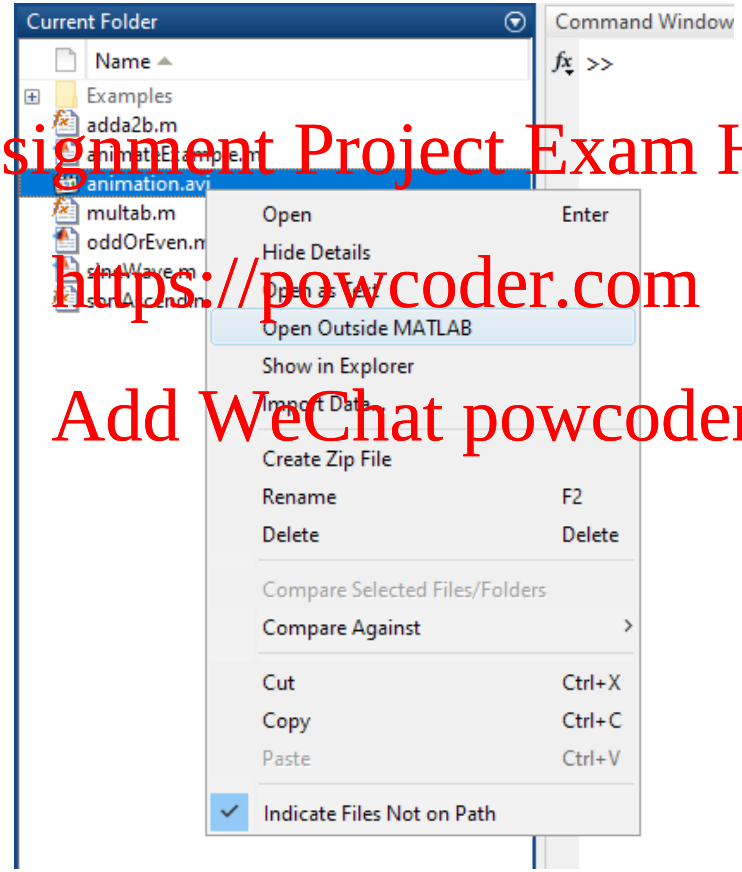
```
% Clear command window, close any open files and clear workspace
clear all;close all;clc

for frame=1:100          % Number of frames in the video
    x = frame/10;        % x, which is a function of the frame number
    y = frame/10;        % y, which is a function of the frame number
    plot (x,y, 'ro');    % plot a red 'o' at x,y
    axis equal;          % set axis x and Y to be equal
    axis([0,10,0,10]);   % set axis limits
    M(frame) = getframe(gcf);    % save each plot frame in an array
end
```



Animating Your Data

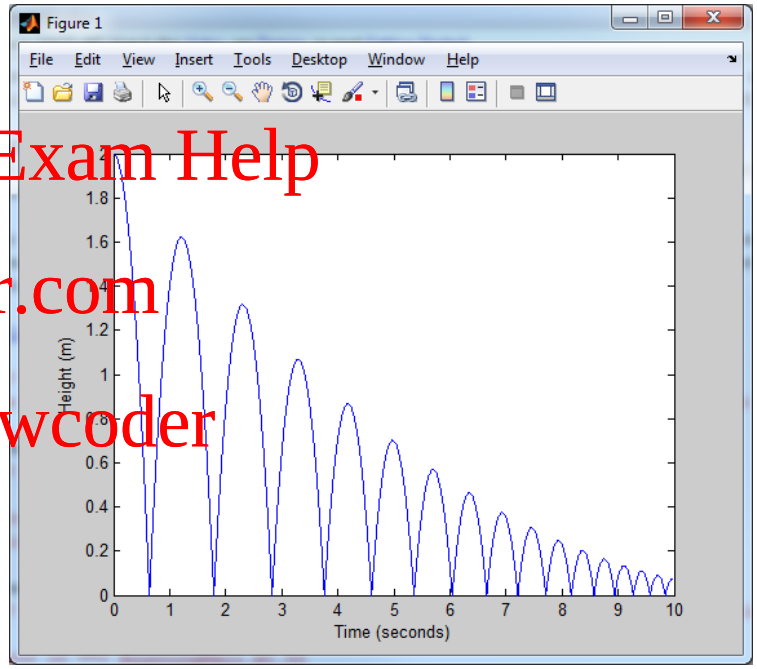
```
% Create a video object called animation
v = VideoWriter('animation.avi','Uncompressed AVI');

% Open the file for writing to
open(v);

% Write the images stored in M to the video file.
writeVideo(v,M)

% Close the file
close(v)
```

Animating Your Data

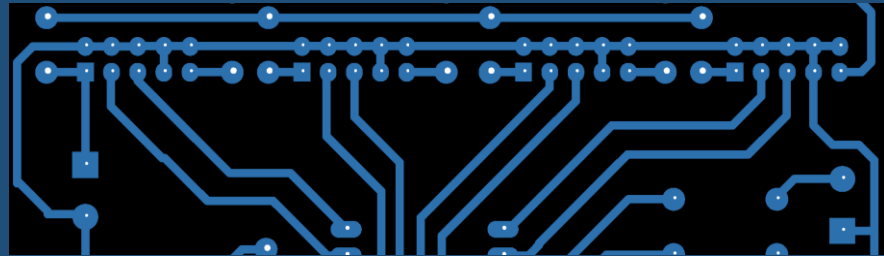


Introduction to MATLAB© Part III

- **In class exercise**
  - To simulate a bouncing ball
  - Program to prompt for initial drop height and coefficient of restitution (COR)
  - Program to output a plot showing the bounce pattern of the ball



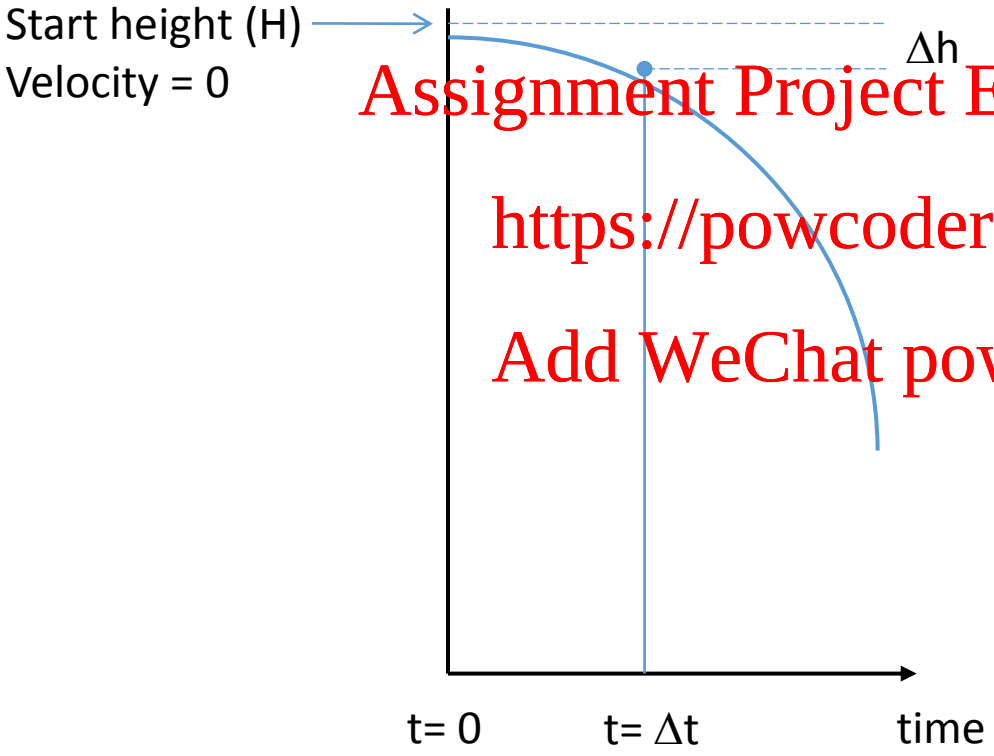
Introduction to MATLAB© Part III



Hints!

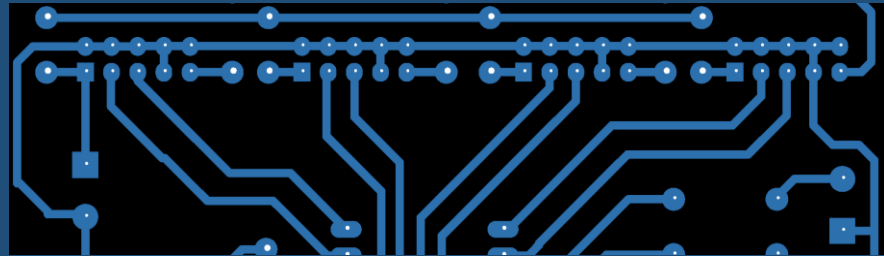
- Track the velocity of the ball.
- Use the velocity to update the height.
- Check to see if ball has hit floor, if so negate velocity and apply COR to determine return velocity.

Hints!



- Set initial conditions.
- Calculate velocity of ball after time Δt.
- Calculate distance moved after time Δt using velocity

Introduction to MATLAB© Part III



Hints

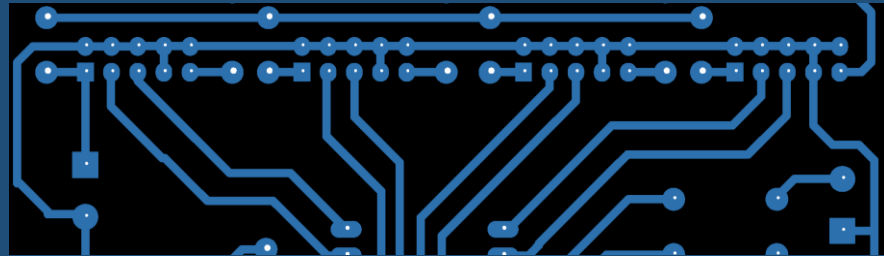
- Velocity after time t, knowing initial velocity and acceleration:

$$v = u + a*t$$

- Distance moved after time t, knowing initial velocity and velocity after time t:

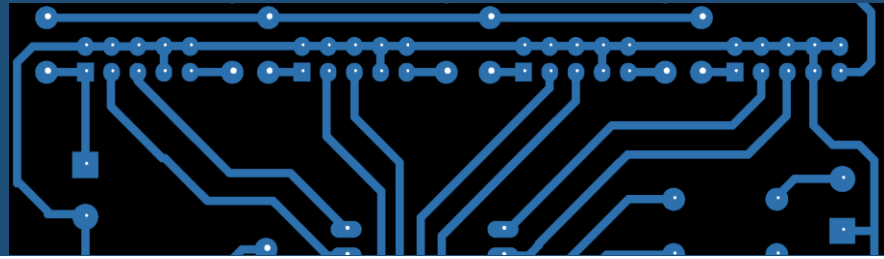
$$s = ½ (u + v) * t$$

$$a = g = 9.8m/s^2$$



Pseudo Code

- Define constants:
  - g, total time, timestep
- Input start height and COR
- Set initial conditions
- Loop over total time
  - Calculate velocity at t = t + timestep
  - Calculate distance moved in time = timestep
  - Update initial velocity
  - Update height
  - Has ball reached floor
    - Yes: negate velocity



Any Questions?