The Department of Engineering Science
The University of Auckland

Chapter 5

Loops

Learning Outcomes

- Explain what a for loop is
- Use for loops in programs
- Manipulate 1D arrays using a for loop
- Explain what a while loop is
- Use while loops in a program
- Describe loops using flowcharts and pseudocode

Loops

- Often in your programs you will want to "loop"
  - repeat some commands multiple times
- May know how many times you want to loop
  - use a `for` loop
- May be looping until something happens
  - conditional loop
  - use a `while` loop
- If you find yourself typing similar lines more than a couple of times, use a loop

For loops

- We want to write out the squares of all integers from 2 to 7
- We will do this several ways in Matlab and along the way will meet the for loop

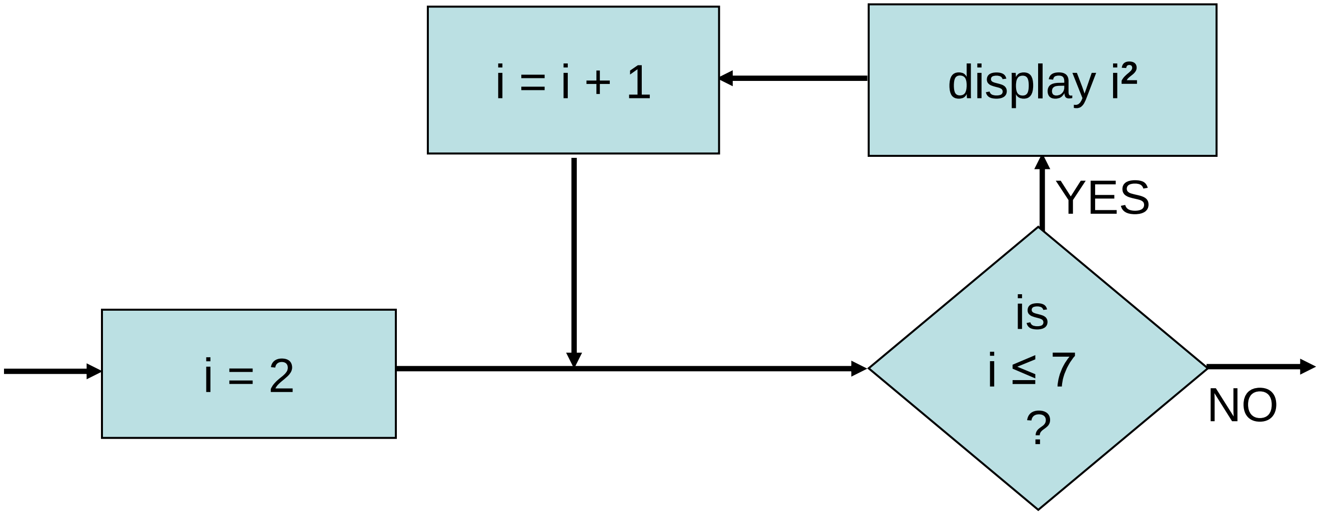
Describing our for loop

- To write out the squares of the integers from 2 to 7

<u>Pseudocode</u>

```
for i = 2 to 7 by 1
        display i²
end
```

<u>Flowchart</u>



Loop variables

- At the heart of a for loop, is the loop variable (often give the name i)
- The first time through, i has a start value
- Each subsequent time it is increased by the step size (usually 1)
- We continue looping until the finish value is reached
- The body of the for loop will often use the loop variable (but it doesn't have to)

General for Loops

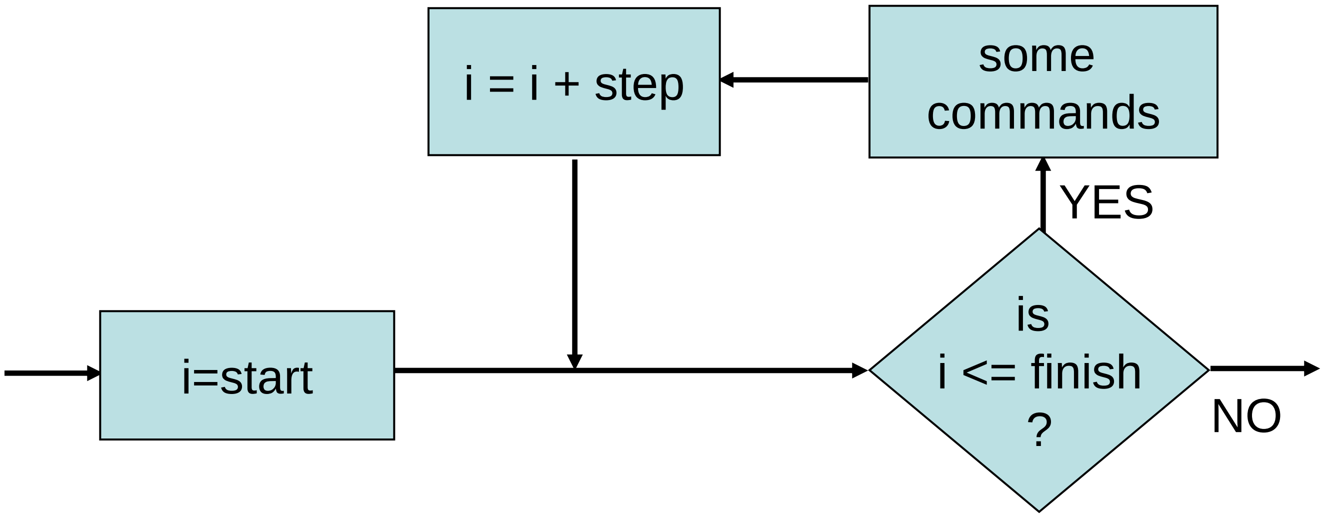
<u>Pseudocode</u>

for i = *start* to *finish* by *step*

some commands

end

<u>Flowchart</u>



Syntax

```
for variable = start:step:finish
```

some commands

```
end
```

- If no step specified assumed to be 1

File: for_loop.m

```
for i=1:5

    disp(i)

end
```

Matlab command prompt

```
>> for_loop
     1
     2
     3
     4
     5
>>
```

Some examples

```
for variable = start:step:finish
```

some commands

```
end
```

– If no step specified assumed to be 1

File: for_loop.m

```
for i=1:5
    disp(i)
end
```

Matlab command prompt

```
>> for_loop
     1
     2
     3
     4
     5
>>
```

`for` loop Example

File: squares.m

```
for i=1:3
    disp('Hi')
end
```

Matlab command prompt

```
>> for_loop_greeting
     Hi
     Hi
     Hi
>>
```

Different step values

File: more_for_loops.m

```
for time=0:0.1:0.5
    disp(time)
end
```

Matlab command prompt

```
>> more_for_loops
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
>>
```

Different step values

File: countdown.m

```
for i=5:-1:1
     disp(i)
end
disp('blastoff!')
```

Matlab command prompt

```
>> countdown
     5
     4
     3
     2
     1
     blastoff!
>>
```

Don't necessarily get finish value

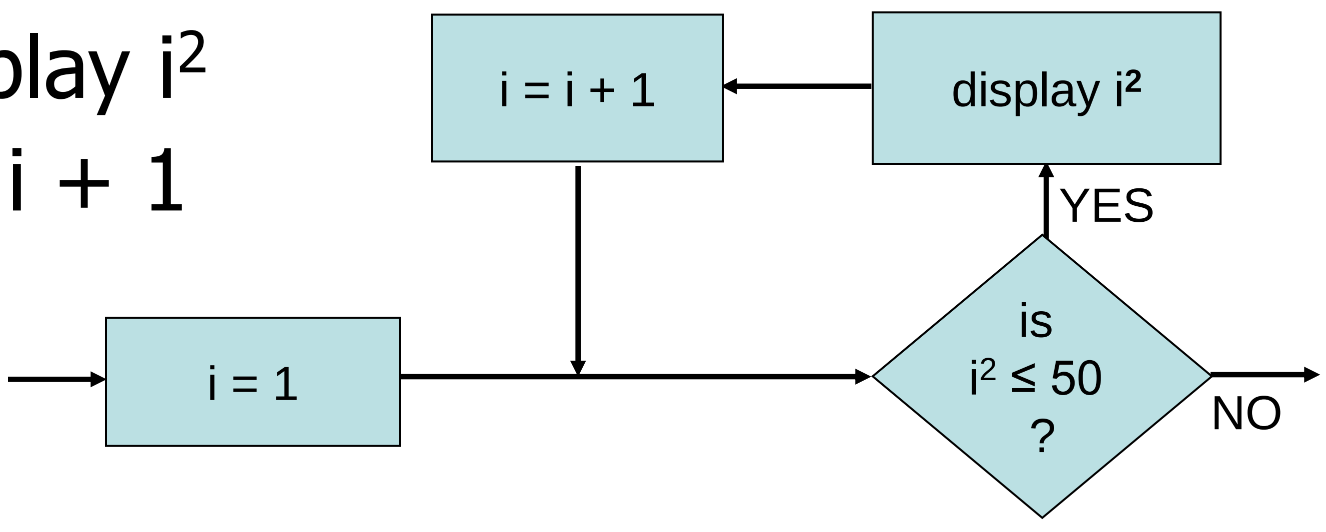
While loops

- Maybe you want to write out squares of integers (starting at 1) until the square exceeds 50

Pseudocode

```
i = 1
while  i² <= 50
       display i²
       i = i + 1
end
```

Flowchart



MATLAB `while` loop Example

initialise

`while` *condition*

   some commands

   update

`end`

File: while_loop.m

```
i = 1;
while i^2 <= 50
    disp(i^2)
    i = i + 1;
end
```

Matlab command prompt

```
>> while_loop
     1
     4
     9
    16
    25
    36
    49
```

While Loops

<u>Pseudocode</u>

initialise

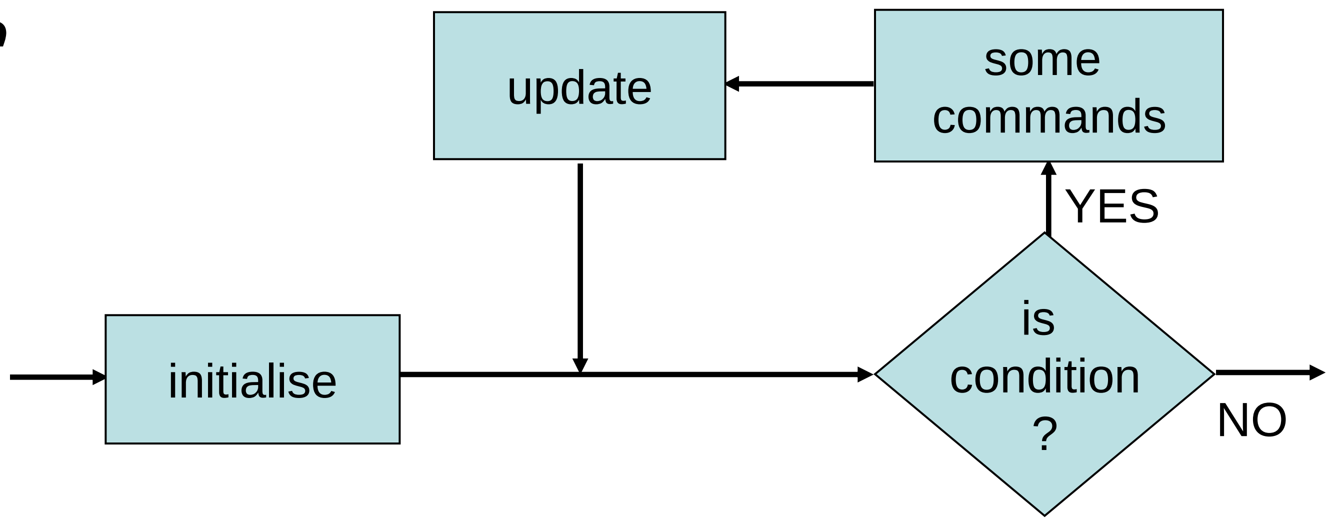
while *condition*

some commands

update

end

<u>Flowchart</u>



Infinite Loops

- "Infinite loop" = piece of code that will execute again and again … without ever ending
- Possible reasons for infinite loops:
  - getting the conditional statement wrong
  - forgetting the update step
- If you are in an infinite loop then *ctrl-c* stops MATLAB executing your program

Infinite Loops

File: infinite_loop.m

```
i = 1;
while i >= 0
    disp(i)
    i = i + 1;
end
```

Matlab command prompt

```
>> infinite_loop
         1
         2
         3
         4
         …
      6824
      6825
>>
```

CTRL-C!

Infinite Loops

File: infinite_loop.m

```
i=1;
while i<=10
    disp(i)
end
```

Matlab command prompt

```
>> infinite_loop
     1
     1
     1
     1
     …
     1
     1
>>
```

CTRL-C!

Booleans and `while` loops

- Use a <u>boolean</u> to control `while` loop

```
stillLooping = true;
while stillLooping
  some commands
  if some conditions
    stillLooping = false;
  end
end
```

Recommended Reading

Chapter 5 Loops	Introduction to Matlab 7 for Engineers (2nd ed)		A Concise Introduction to Matlab (1st ed)	
Topic	**Section**	**Pages**	**Section**	**Pages**
Loops	1.6	48-51		
For loops	4.5	211-213	4.4	170-174
While loops	4.5	221-225	4.4	178-180

The Department of Engineering Science
The University of Auckland

Chapter 6

2D and 3D Arrays

Learning outcomes

- Explain what a 2D array is
- Create and manipulate 2D arrays
- Draw plots of 2D arrays
- Perform calculations with 2D arrays
- Manipulate 2D arrays using for loops
- Manipulate images via 3D arrays

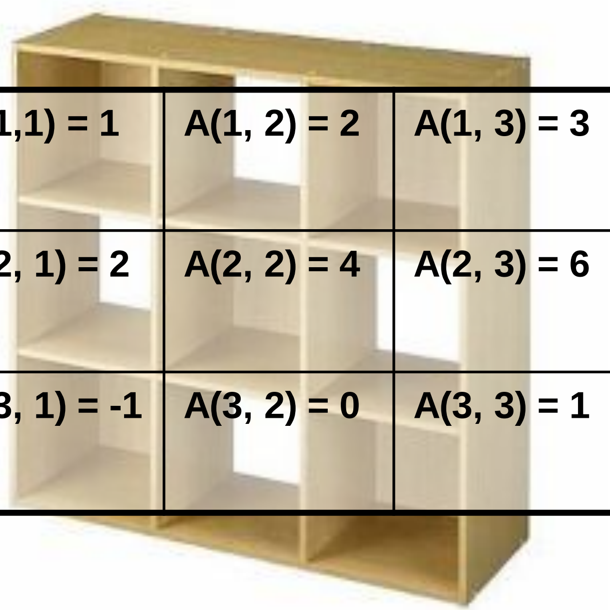
2D Arrays

- Variables so far have been scalars (single value) and 1D arrays (lists of values)
- Some types of data are suited to being stored in 2D arrays
  - data which corresponds to an underlying physical "grid"
  - data from a table
  - data representing the elements of a matrix

2D Arrays versus 1D Arrays

- If a 1D array is like a *filing cabinet*, a 2D array is like a set of *cubby holes*

```
>> A = [1, 2, 3;
        2, 4, 6;
       -1, 0, 1]

A =

     1     2     3
     2     4     6
    -1     0     1
```



A

A(1,1) = 1	**A(1, 2) = 2**	**A(1, 3) = 3**
A(2, 1) = 2	**A(2, 2) = 4**	**A(2, 3) = 6**
A(3, 1) = -1	**A(3, 2) = 0**	**A(3, 3) = 1**

Creating 2D arrays

- Create a table of values
  - enclosing numbers within [ ]
  - separating columns by , or a space
  - separating rows by ;

```
>> QuarterlyProd = [42, 52, 48, 47;
          41, 48, 50, 42;
          51, 38, 40, 41]
QuarterlyProd =
          42   52   48   47
          41   48   50   42
          51   38   40   41
>>
```

Accessing Array Elements

- You can access 2D array elements by specifying the row and column using (,)

```
>> QuarterlyProd = [42, 52, 48, 47;
         41, 48, 50, 42;
         51, 38, 40, 41]

QuarterlyProd =

    42    52    48    47
    41    48    50    42
    51    38    40    41

>> QuarterlyProd(2,3)

ans =

    50

>> QuarterlyProd(2,3) = 35

QuarterlyProd =

    42    52    48    47
    41    48    35    42
    51    38    40    41
```

Extending Arrays

- You can add extra elements by
  - creating them directly (,)
    - MATLAB fills in the gaps with 0

```
>> QuarterlyProd = [42, 52, 48, 47;
         41, 48, 50, 42;
         51, 38, 40, 41]

QuarterlyProd =

    42    52    48    47
    41    48    50    42
    51    38    40    41

>> QuarterlyProd(4, 1) = 45

QuarterlyProd =

    42    52    48    47
    41    48    50    42
    51    38    40    41
    45     0     0     0
```

Extending Arrays

- You can concatenate elements to 2D arrays
  - Need to make sure dimensions of new elements are correct

```
>> A = [8, 9; 1 2]

A =

     8     9
     1     2

>> B = [4 5]

B =

     4     5
```

```
>> C = [3; 5]

C =

     3
     5

>> D = [A; B]

D =

     8     9
     1     2
     4     5
```

```
>> E = [A, C]

E =

     8     9     3
     1     2     5

>> F = [A, C; B, 12]

F =

     8     9     3
     1     2     5
     4     5    12
```

2D Array Functions

- Standard mathematical functions can be applied to 2D arrays too

```
>> x = [1, 2, 3; 4, 5, 6];
>> y = sin(x)
y =
     0.8415     0.9093     0.1411
    -0.7568    -0.9589    -0.2794

      sin(1)     sin(2)     sin(3)
      sin(4)     sin(5)     sin(6)
```

Special Array Functions

```
>> [m, n] = size(A)
```

– `m` = number of rows, `n` = number of columns

- transpose operator '
  – swaps the rows and columns in an array

```
>> A = [1 2 3;
        4 5 6];
>> B = A'

B =

     1     4
     2     5
     3     6
```

Automatic 2D Arrays

- Ways to create 2D arrays automatically

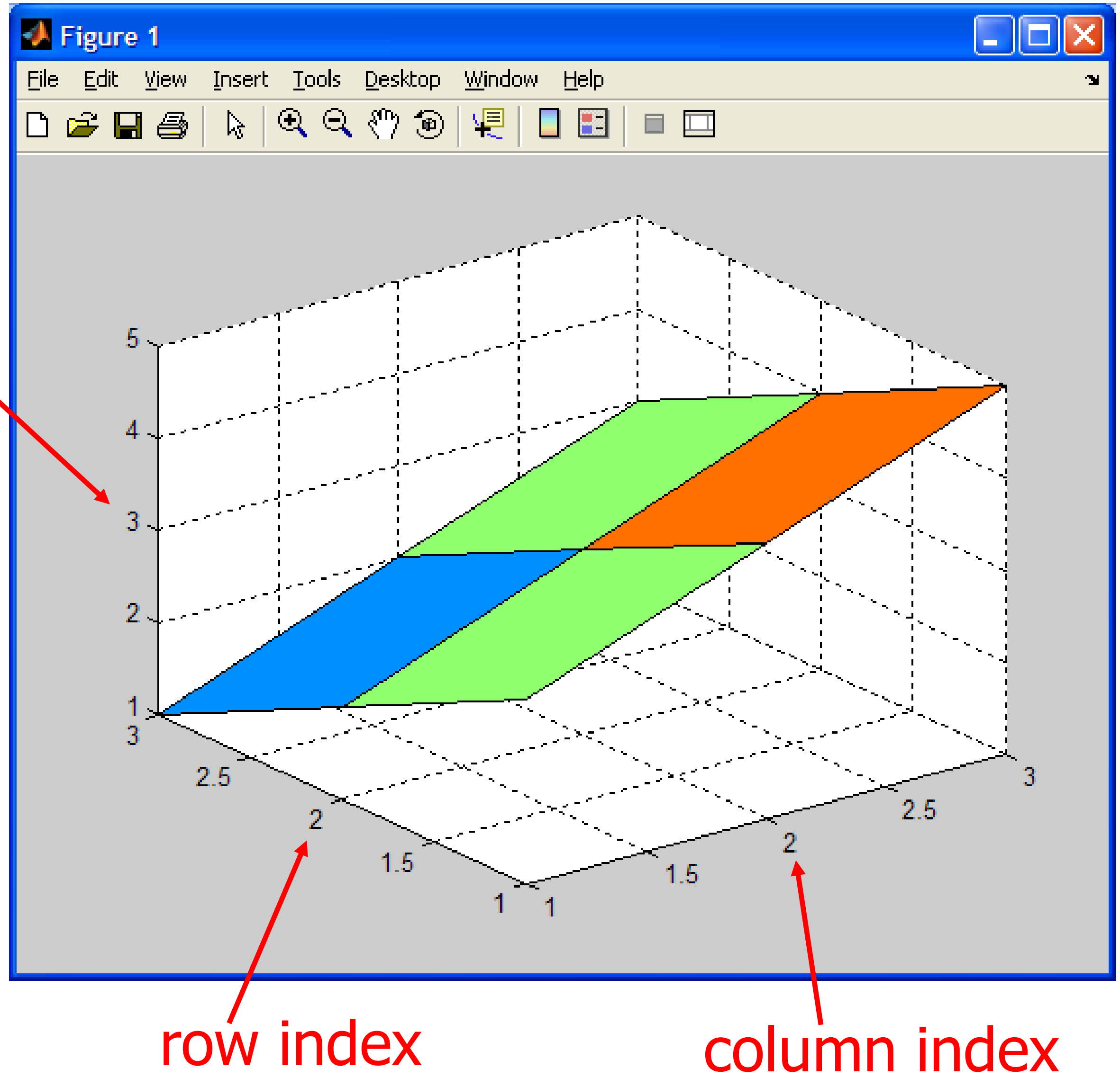
```
>> eye(3)
ans =

     1     0     0
     0     1     0
     0     0     1
```

```
>> zeros(2, 4)
ans =

     0     0     0     0
     0     0     0     0
>> ones(3, 2)
ans =

     1     1
     1     1
     1     1
```

  - `meshgrid` (more later)

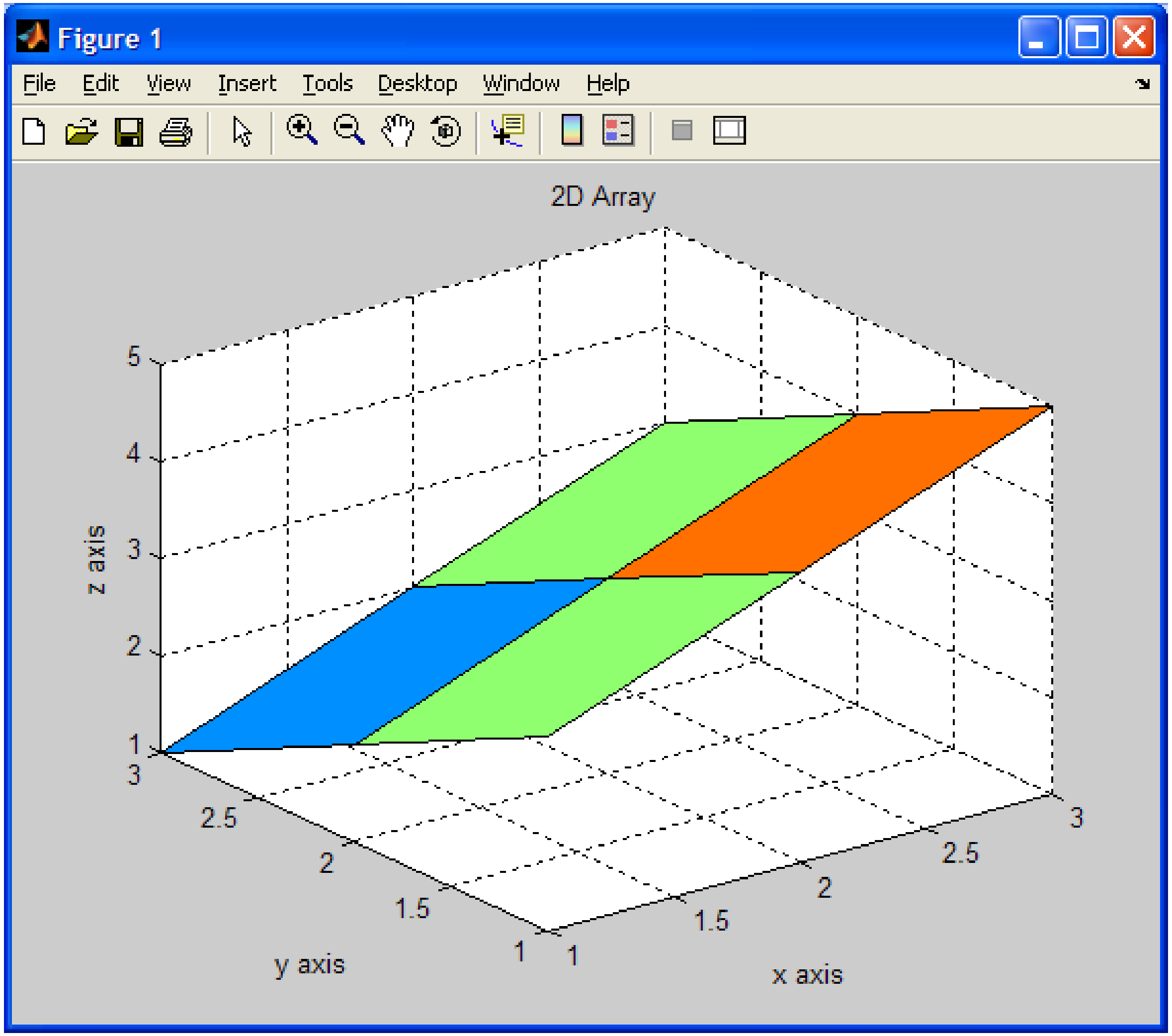
Drawing 2D Arrays

```
>> M = [3 4 5;
2 3 4;
1 2 3]

M =

     3     4     5
     2     3     4
     1     2     3

>> surf(M)
```



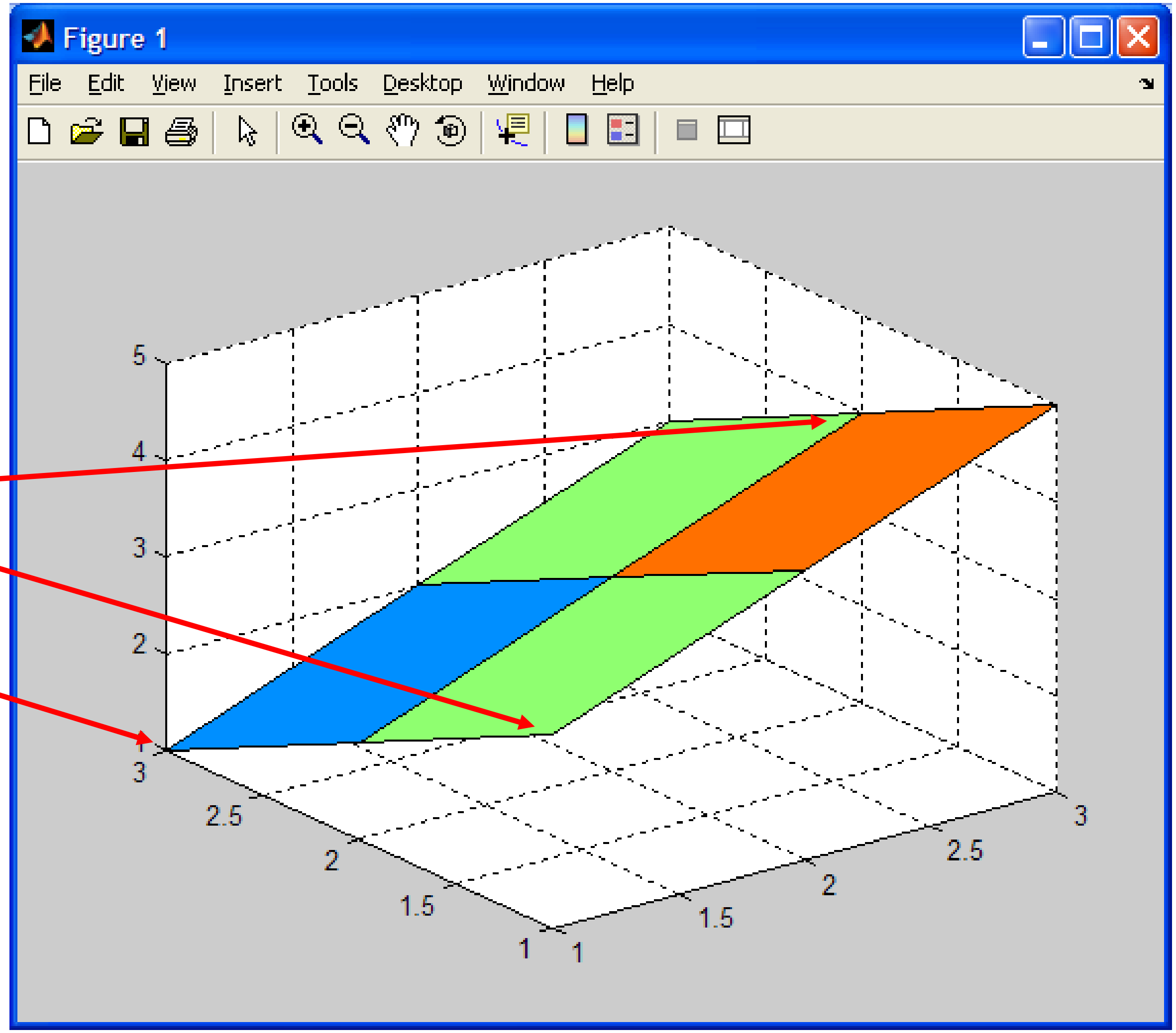
Adding Labels

```
>> M = [3 4 5;
2 3 4;
1 2 3]

M =

     3     4     5
     2     3     4
     1     2     3

>> surf(M)
>> xlabel('x axis')
>> ylabel('y axis')
>> zlabel('z axis')
>> title('2D Array')
```



2D Arrays as Surfaces

```
>> M = [3 4 5;
2 3 4;
1 2 3]

M =

     3     4     5
     2     3     4
     1     2     3

>> surf(M)
```

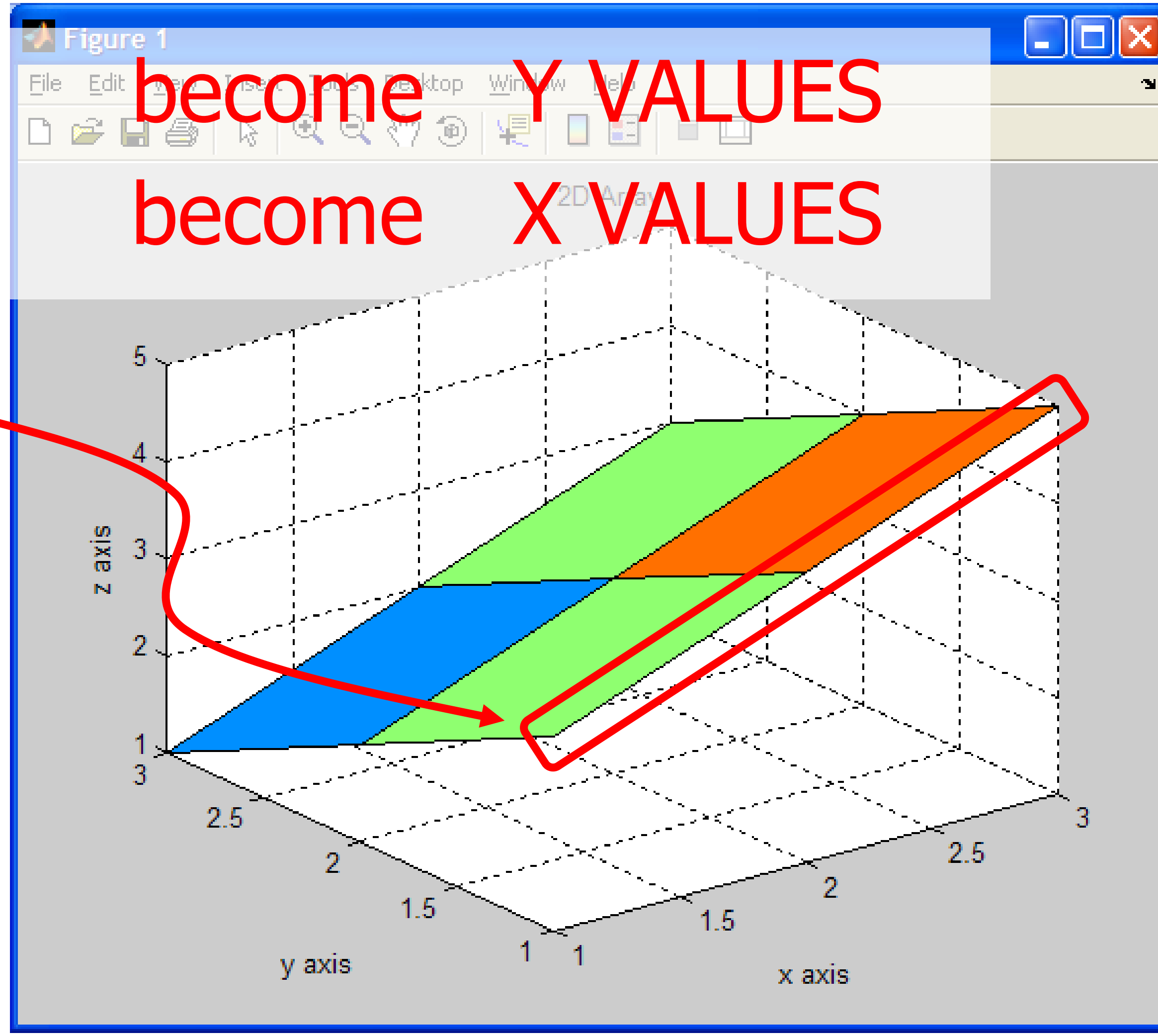


Matrices as Surfaces

```
>> M = [3 4 5;
2 3 4;
1 2 3]

M =

     3     4     5
     2     3     4
     1     2     3

>> surf(M)
>> xlabel('x axis')
>> ylabel('y axis')
>> zlabel('z axis')
>> title('2D Array')
```

ROWS become Y VALUES

COLUMNS become X VALUES



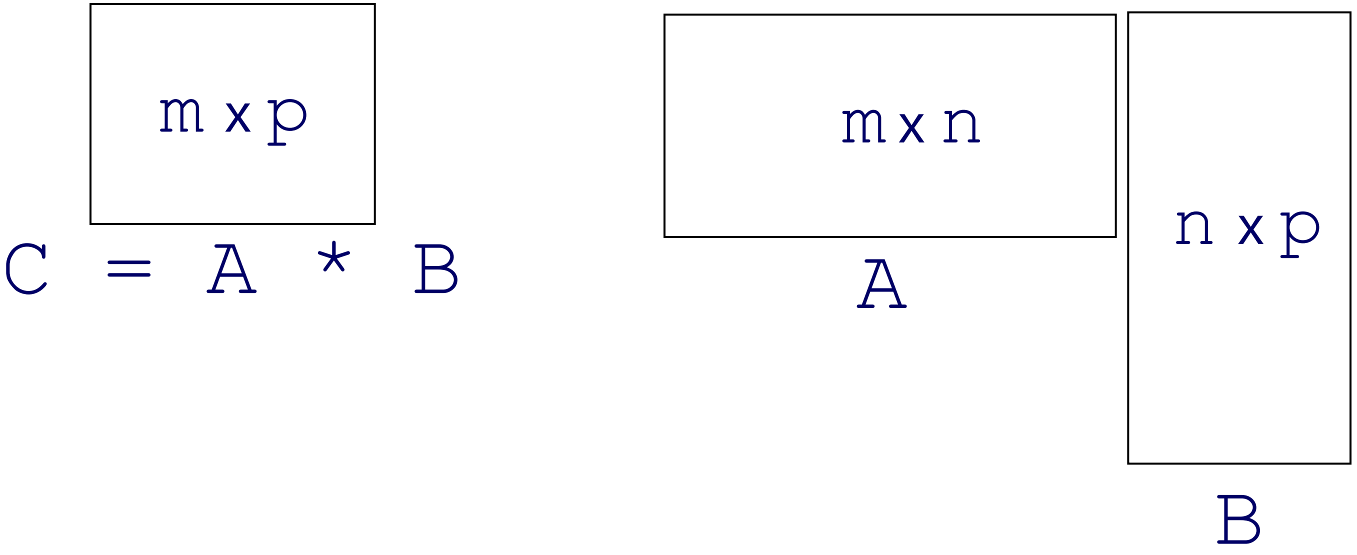
Arithmetic With 2D Arrays

- Two 2D arrays can be added or subtracted using the + and - operators … as long as arrays have same size

  Hint Use `size` command to find out how big an array is or check in the workspace window

Multiplication With 2D Arrays

- Two 2D arrays multiplied with `*` operator
  - first array must have same number of columns as second array has rows
  - `size(A, 1)` gives number of rows of `A`
  - `size(A, 2)` gives number of columns of `A`



Multiplication With 2D Arrays

```
>> A = [3 1 0;
        1 -2 4];
>> B = [2;
        4;
        1];
>> C = A * B

C =

    10
    -2

>>
```

$$C = A \times B$$

$$= \begin{bmatrix} (3\times 2) + (1\times 4) + (0\times 1) \\ (1\times 2) + (-2\times 4) + (4\times 1) \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ -2 \end{bmatrix}$$

Multiplication With 2D Arrays

- In mathematically based work this kind of array multiplication is very useful
- However in some applications we want to perform an element-wise multiplication
  - Multiply each element in first array by corresponding element in second array
  - Two arrays must be same size

Element-wise Multiplication

- To perform multiplication element-wise use a . before operator

```
>> A = [3 1 0;
        1 -2 4];
>> B = [4 2 -1;
        0 1 3];
>> C = A .* B

C =

    12     2     0
     0    -2    12
```

$$C = A.*B$$

$$= \begin{bmatrix} (3\times 4) & (1\times 2) & (0\times -1) \\ (1\times 0) & (-2\times 1) & (4\times 3) \end{bmatrix}$$

$$= \begin{bmatrix} 12 & 2 & 0 \\ 0 & -2 & 12 \end{bmatrix}$$

Dot Operator

- Dot operator can also be applied with other mathematical operations
  - `.^ 2` squares elements in array term by term instead of multiplying whole array by itself
  - `./` divides array element by element

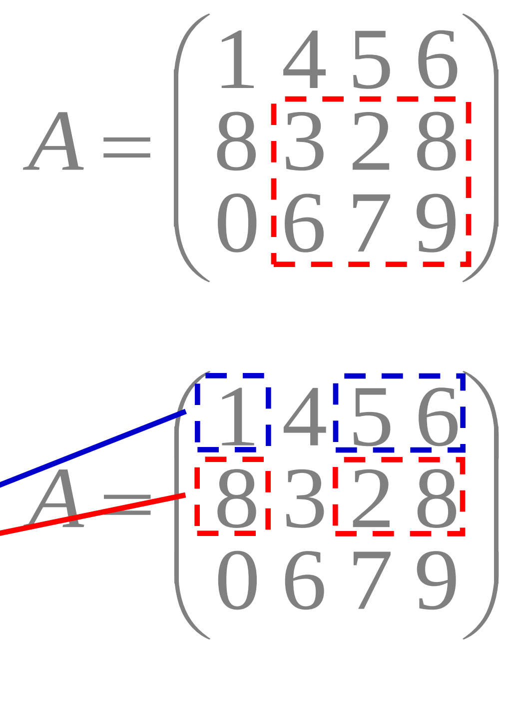
```
>> denom = [2, 3, 4, 5, 6];
>> numer = [1, 2, 3, 4, 5];
>> fracs = numer ./ denom

fracs =

    0.5000    0.6667    0.7500    0.8000    0.8333
```

$\frac{1}{2}$ $\frac{2}{3}$ $\frac{3}{4}$ $\frac{4}{5}$ $\frac{5}{6}$

Subranges

- Can select any submatrix using 1D arrays of indices

```
>> A = [1 4 5 6; 8 3 2 8; 0 6 7 9];
>> B = A(2:3, 2:4)

B =


     3     2     8
     6     7     9

>> C = A([2 1], [1 3 4])

C =


     8     2     8
     1     5     6
```



Colon Operator

- Using a colon : instead of an index array refers to ALL rows or columns of the array

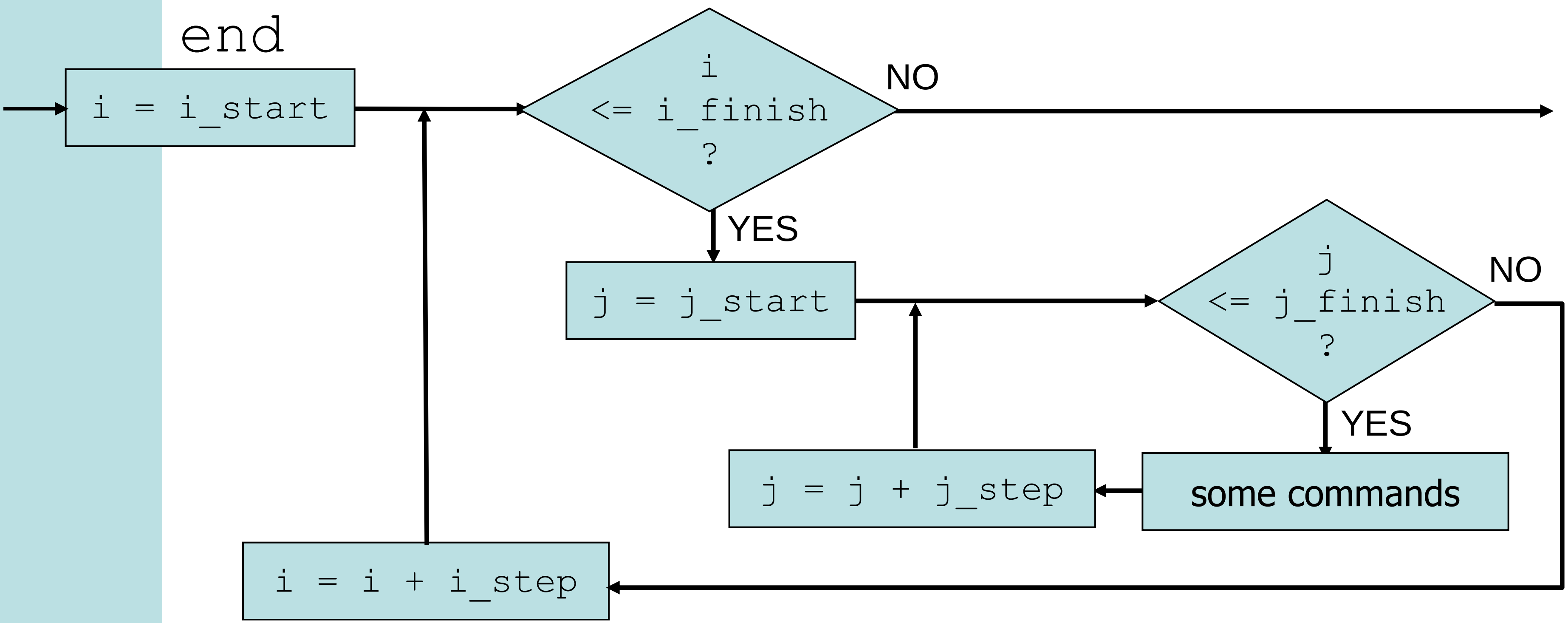
```
>> A = [1 4 5 6;
        8 3 2 8;
        0 6 7 9];
>> B = A(2, :)

B =

     8     3     2     8
```

```
>> C = A(:, 2)

C =

     4
     3
     6

>> D = A(1:2, :)

D =

     1     4     5     6
     8     3     2     8
```

Nested Loops

```
for i = i_start:i_step:i_finish
   for j = j_start:j_step:j_finish
         some commands
   end
end
```

Indenting helps simplify debugging

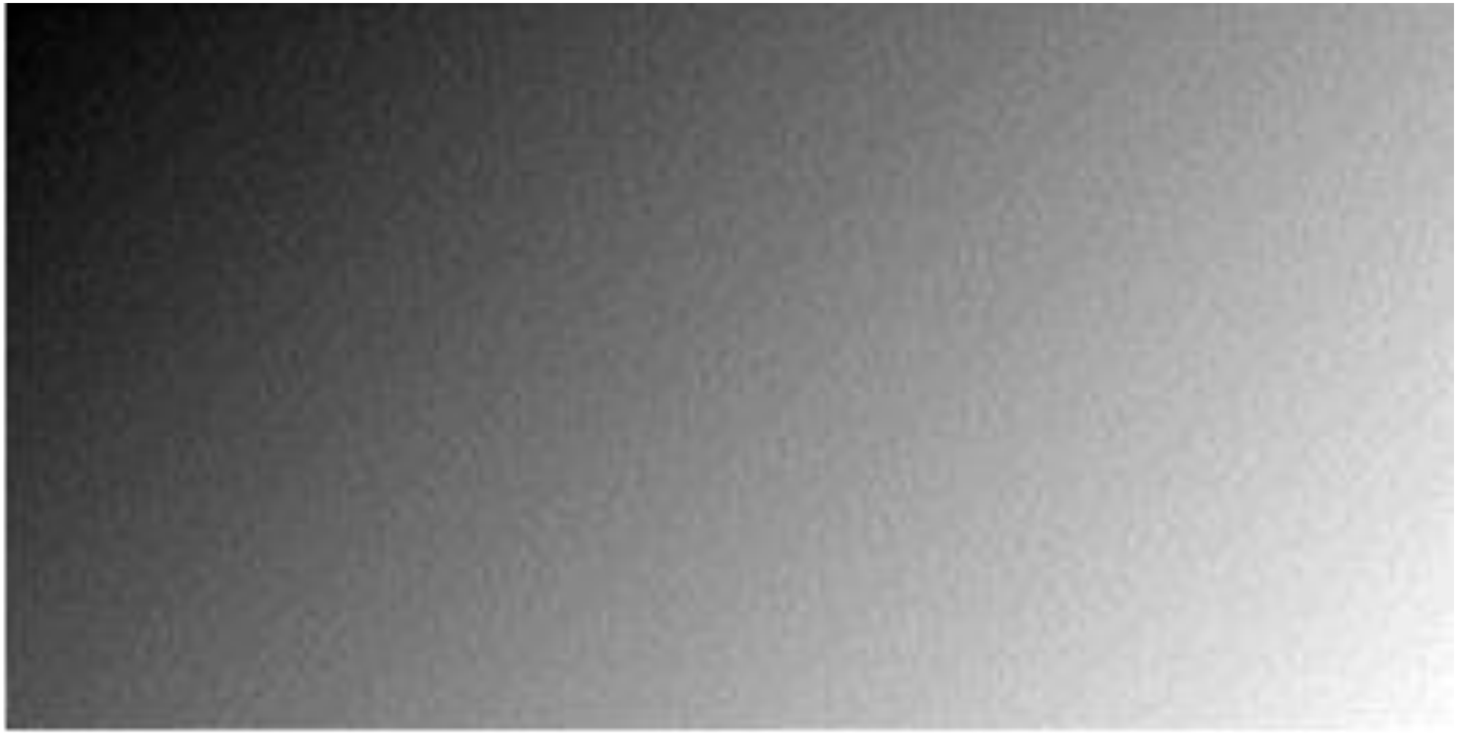


2D arrays and for loops

Editing a greyscale image

```
% cycle through each row
for i = 1:100
    % cycle through each column
    for j = 1:200
        % set the pixel value for row i, column j
        image(i,j) = (i+j)/300;
    end;
end;
```

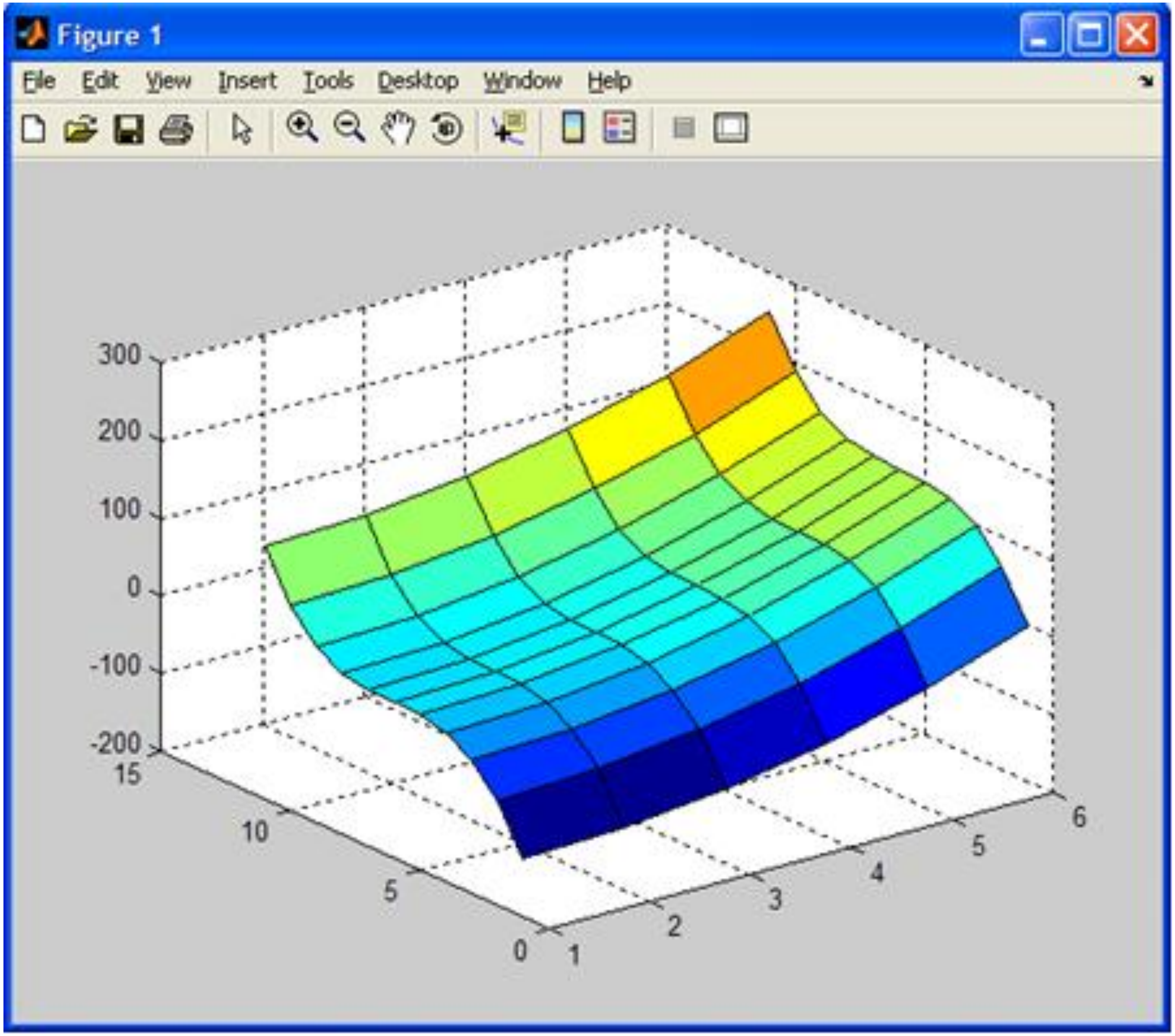
Gray Scale from black to white



Plotting 3D polynomials

```
x = 0:5;
y = -5:5;
for i = 1:length(x),
  for j = 1:length(y),
    Z(j, i) = 5 * x(i)^ 2 + y(j)^ 3;
  end;
end;

surf(Z)
```

Surface plot



3D arrays and image processing

```
myPicture = imread('photo.jpg')
[rows,cols,colours] = size(myPicture);
for i=1:rows
    for j=1:cols
        for k=1:3
            myPicture(i,j,k) = 255 - myPicture(i,j,k);
        end
    end
end
imshow(myPicture);
```

Negative (inverted colours)

Recommended Reading

Chapter 6 2D and 3D Arrays	Introduction to Matlab 7 for Engineers (2nd ed)		A Concise Introduction to Matlab (1st ed)	
Topic	**Section**	**Pages**	**Section**	**Pages**
Multidimensional Arrays	2.2	81-83	2.2	49
Nested for loops	4.5	211-212	4.4	172-173
Plotting surfaces	5.8	335-338	5.7	251-254-

The Department of Engineering Science
The University of Auckland

Chapter 7

Graphics

Learning outcomes

- Label your plots
- Create different types of 1D data plots (log graphs, bar graphs and polar plots)
- Control line types, axis types and colours on 1D plots

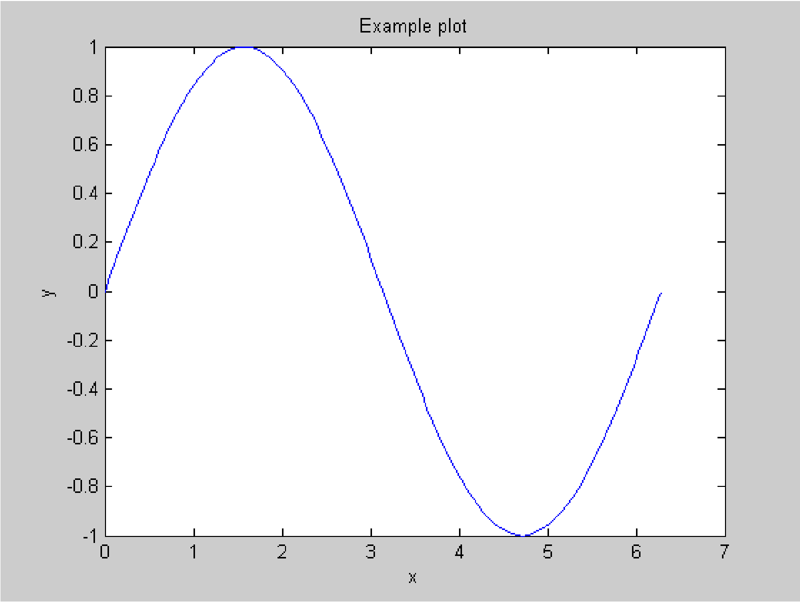
Learning Outcomes

- Create several figures at the same time
- Plot several sets of data on the same graph
- Create subplots
- Create different types of 2D data plots (surface maps, contour plots and quiver plots)
- Make Matlab movies

Labelling plots

- You have already seen basic plotting of one array against another. This is simple to do in Matlab using the **plot** command. It is also simple to label plots using the title, xlabel and ylabel commands

```
x = 0 : 2*pi/100 : 2*pi;
y = sin(x);
plot(x,y)
xlabel('x')
ylabel('y')
title('Example plot')
```

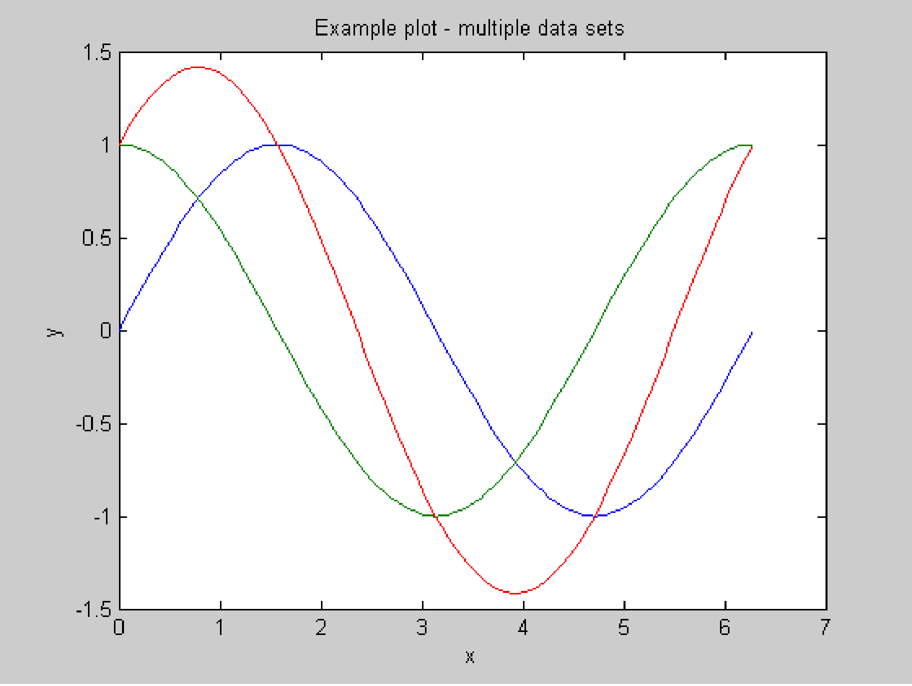


Always label axes on plots you produce in labs or projects.

Plotting Multiple Data Sets

- The plot command can be used to plot several lines on the same graph, e.g.:

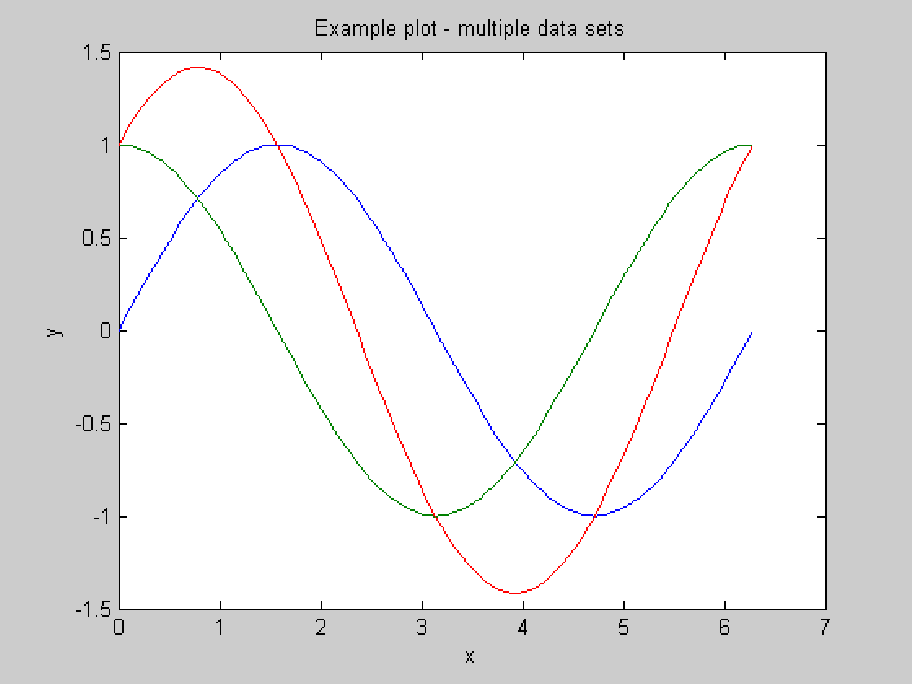
```
x = 0 : 2*pi/100 : 2*pi;
y1 = sin(x);
y2 = cos(x);
y3 = sin(x) + cos(x);
plot(x,y1,x,y2,x,y3)
xlabel('x')
ylabel('y')
title('Example plot - multiple data sets')
```



Plotting Multiple Data Sets

- An alternative is to use the **hold on** command to hold on to your current plot:

```
x = 0 : 2*pi/100 : 2*pi;
y1 = sin(x);
y2 = cos(x);
y3 = sin(x) + cos(x);
plot(x,y1)
hold on
plot(x,y2)
plot(x,y3)
xlabel('x')
ylabel('y')
title('Example plot - multiple data sets')
```



Line Colors, Symbols and Types

- You can also specify your own line styles in the plot command.

b blue . point - solid

g green o circle : dotted

r red x x-mark -. dashdot

c cyan + plus -- dashed

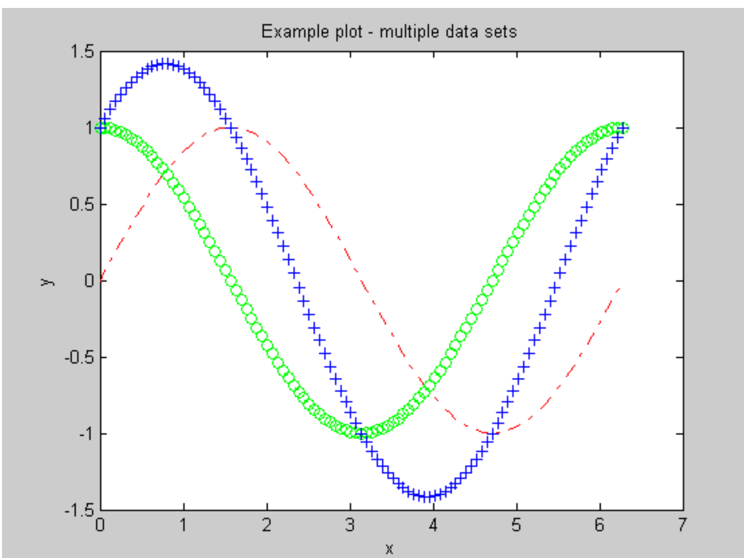
m magenta * star

etc.

- For full details enter **help plot** in Matlab.

Line Colors, Symbols and Types

- To specify line types combine your desired color and symbol/line type into a string and use it as an argument in the **plot** command.




```
plot(x,y1,'r-.',x,y2,'go',x,y3,'b+')
```

red dashdot

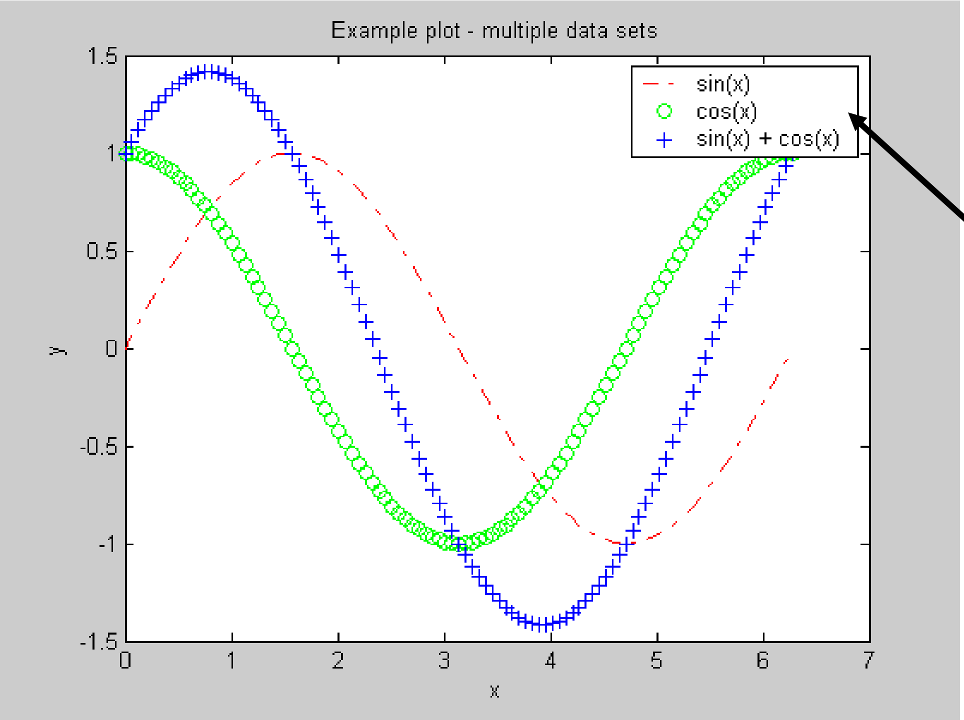
green circle

blue plus

Legends

- With multiple lines on the same plot it is a good idea to add a legend.

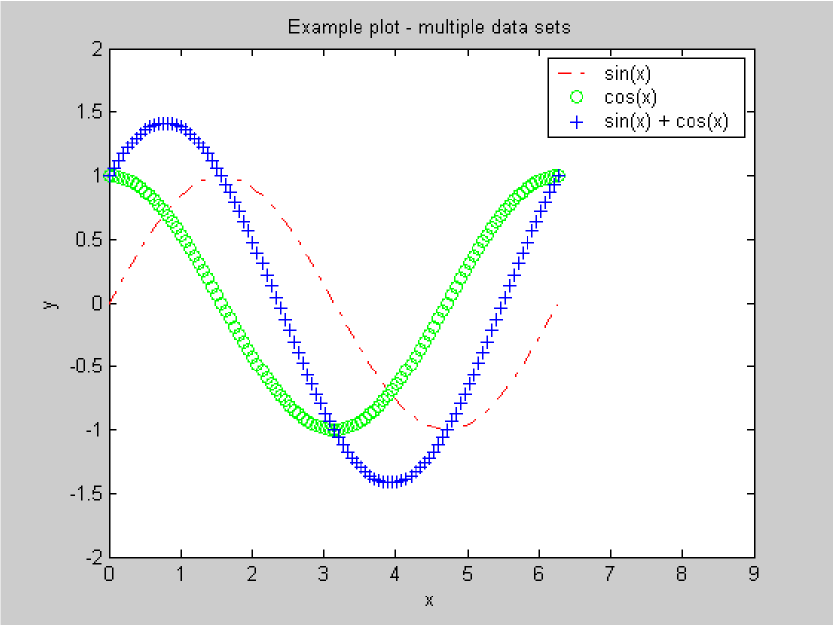
```
legend('sin(x)', 'cos(x)', 'sin(x) + cos(x)')
```



You can move the position of the legend on the figure with the mouse.

Axes

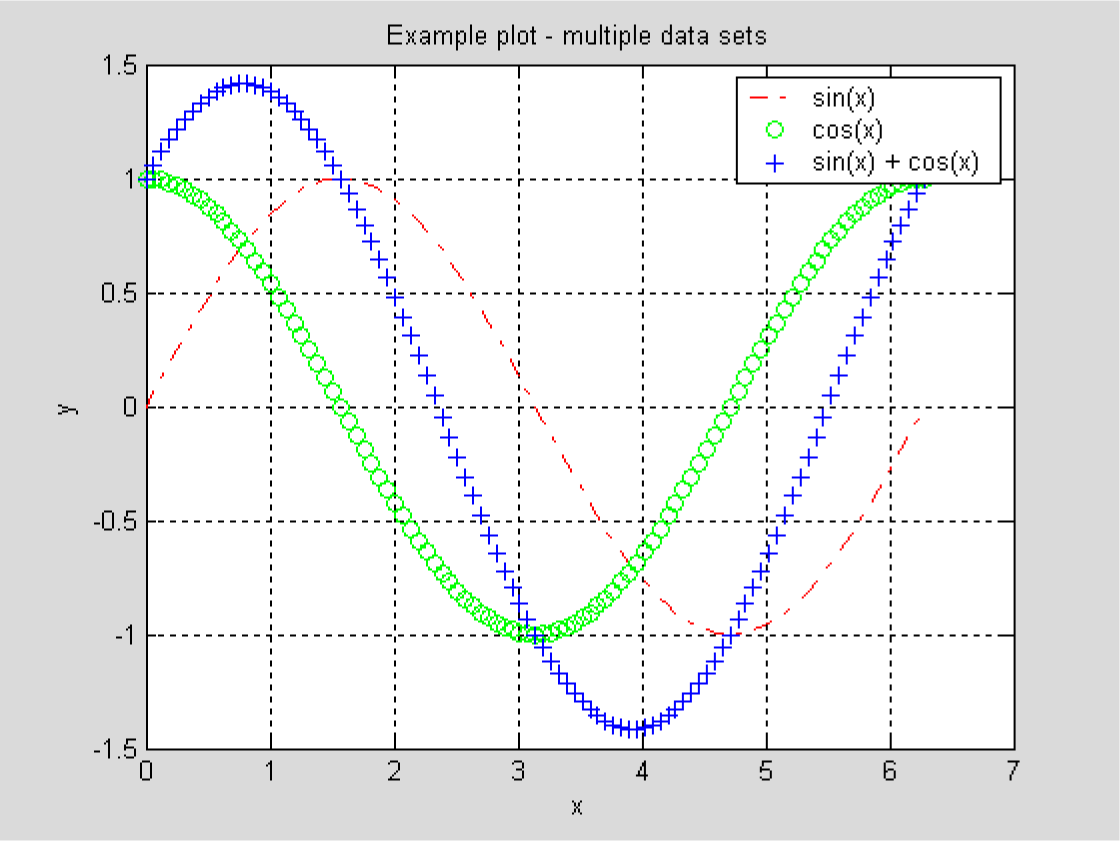
- Matlab will automatically determine the maximum and minimum values for the axes. To override these use the **axis** command to enter an array containing xmin, xmax, ymin, ymax.



axis([0, 9, -2, 2])

Grid Lines

- If you like grid lines on your plots you can add them using the **grid on** command.



grid on

Creating Additional Figures

- What happens if you enter the following?

```
x = 0 : 2*pi/100 : 2*pi;
y1 = sin(x);
y2 = cos(x);

plot(x,y1)
title('Example plot #1')

plot(x,y2)
title('Example plot #2')
```

Creating Additional Figures

… you end up with one figure window and it contains a plot of *y=cos(x)*.

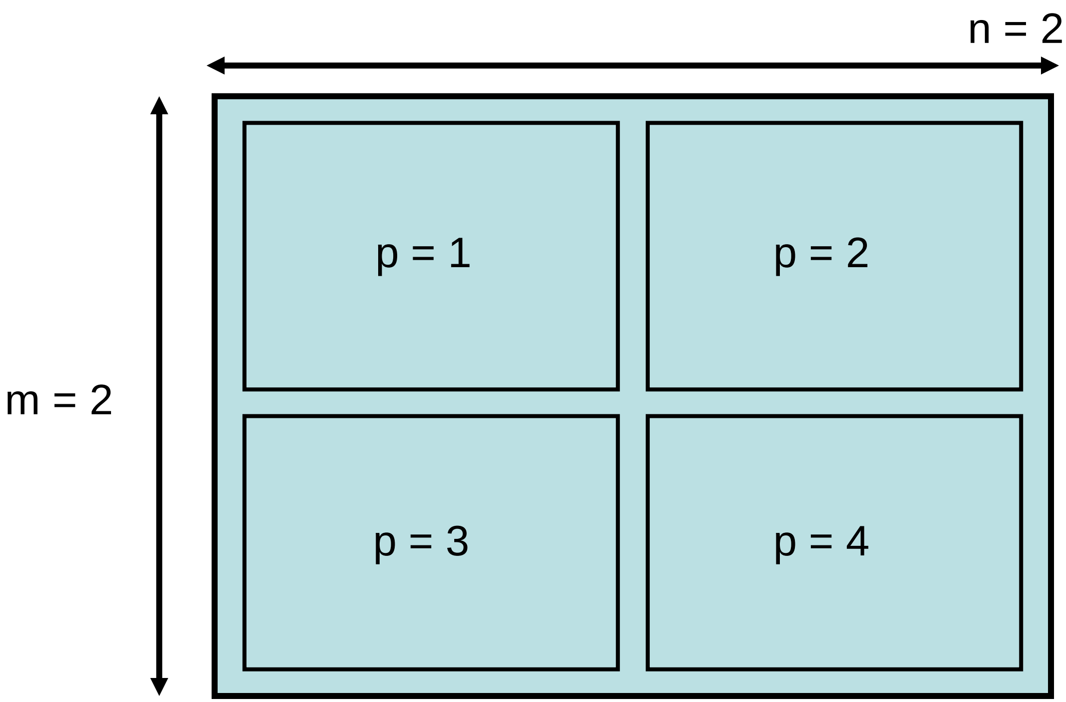
- To make an additional figure window enter the command **figure** before making the second plot.

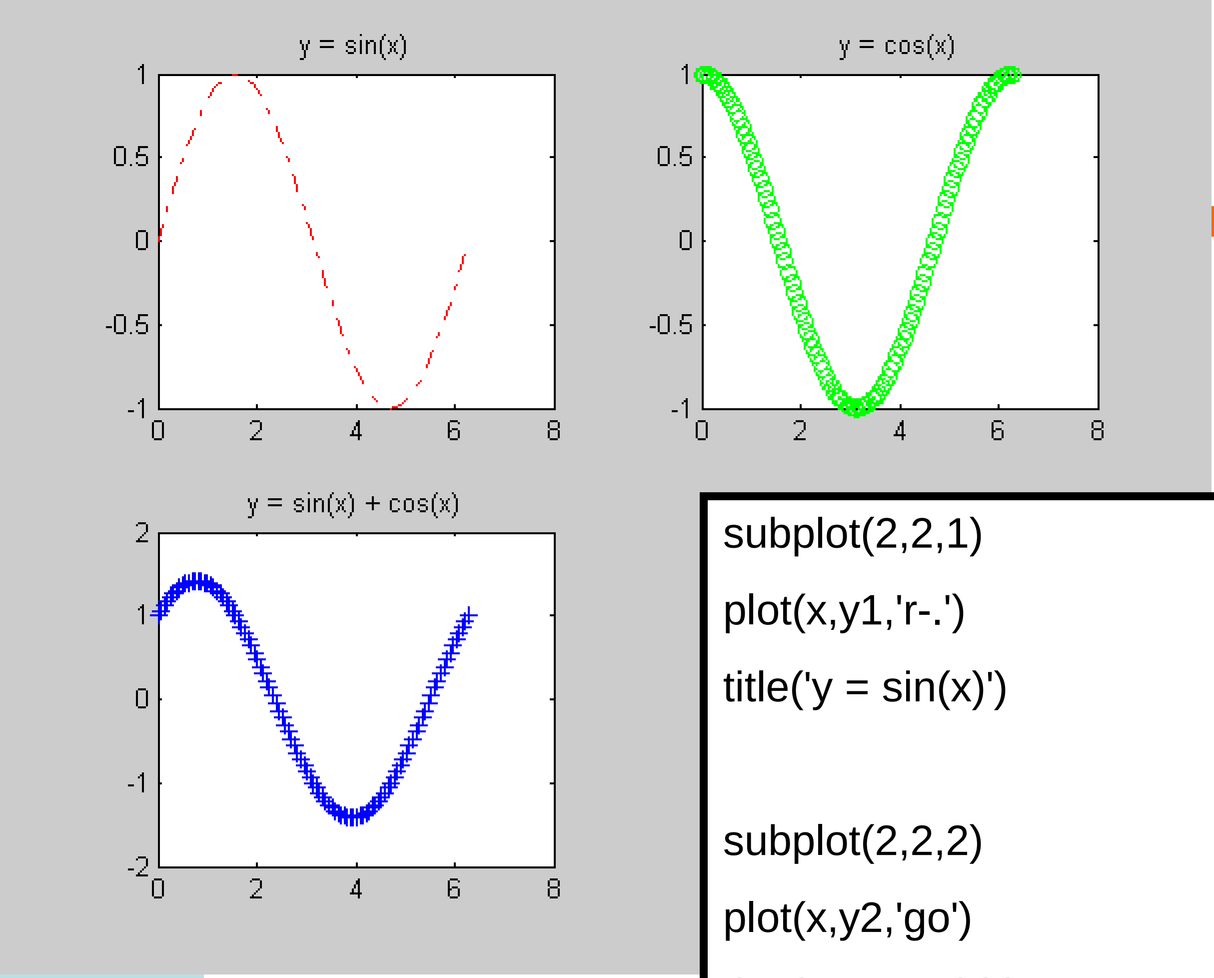
```
plot(x,y1)
title('Example plot #1')


figure
plot(x,y2)
title('Example plot #2')
```

Note: The second figure window often appears on top of first figure window by default.

Subplots

- Sometimes it makes sense to present data as a set of plots contained inside the same figure, this can be done with the **subplot(m,n,p)** command.





You may want to resize the figure window with the mouse if you are using subplots.

```
subplot(2,2,1)
plot(x,y1,'r-.')
title('y = sin(x)')

subplot(2,2,2)
plot(x,y2,'go')
title('y = cos(x)')

subplot(2,2,3)
plot(x,y3,'b+')
title('y = sin(x) + cos(x)')
```

m = 2
n = 2
p = 3

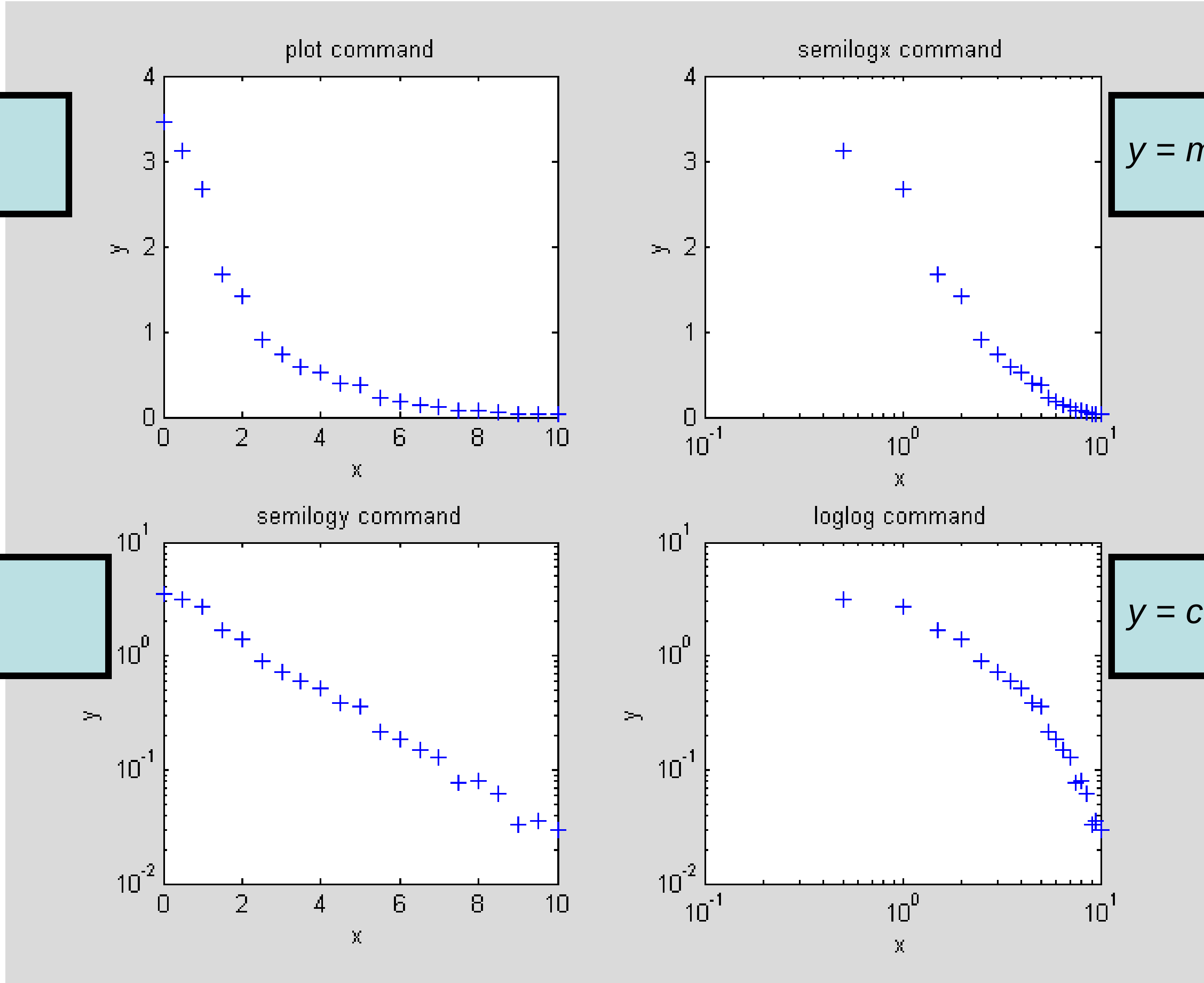
Log graphs

- You can create line graphs with log scaling on either or both axes using the commands **semilogx**, **semilogy**, and **loglog**. Syntax is the same as **plot**.

- This can be useful when you are deciding on models to fit to data sets.

$y = mx + c$

$y = m \log(cx)$



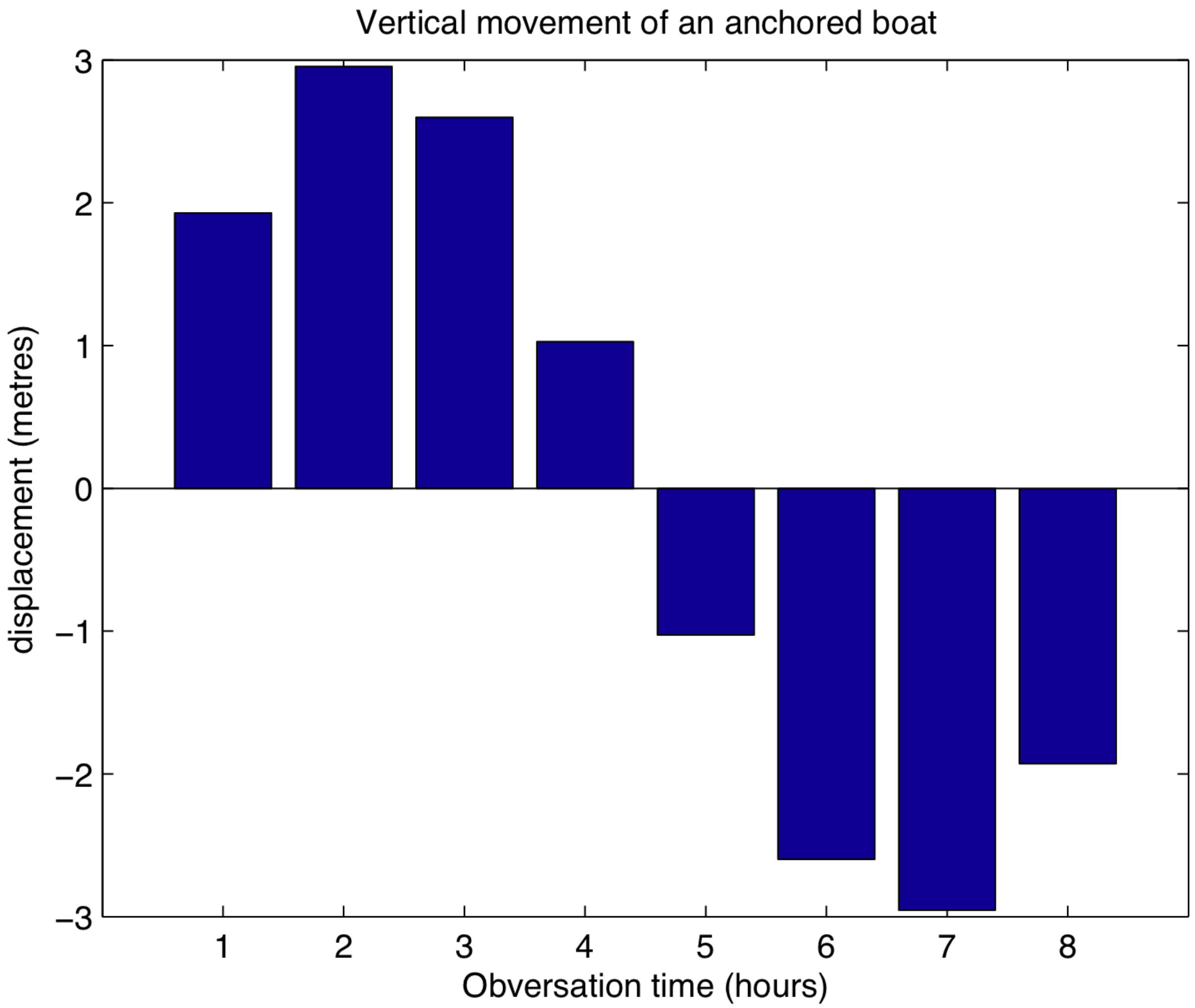
$y = ce^{mx}$

$y = c\,x^{m}$

Bar graphs

- You can create a bar graph with the bar function: bar(x,y)
- Similar to plot but draws bars for each x,y value pair
- Make sure there are no duplicate values in the x array.

Bar graphs: example



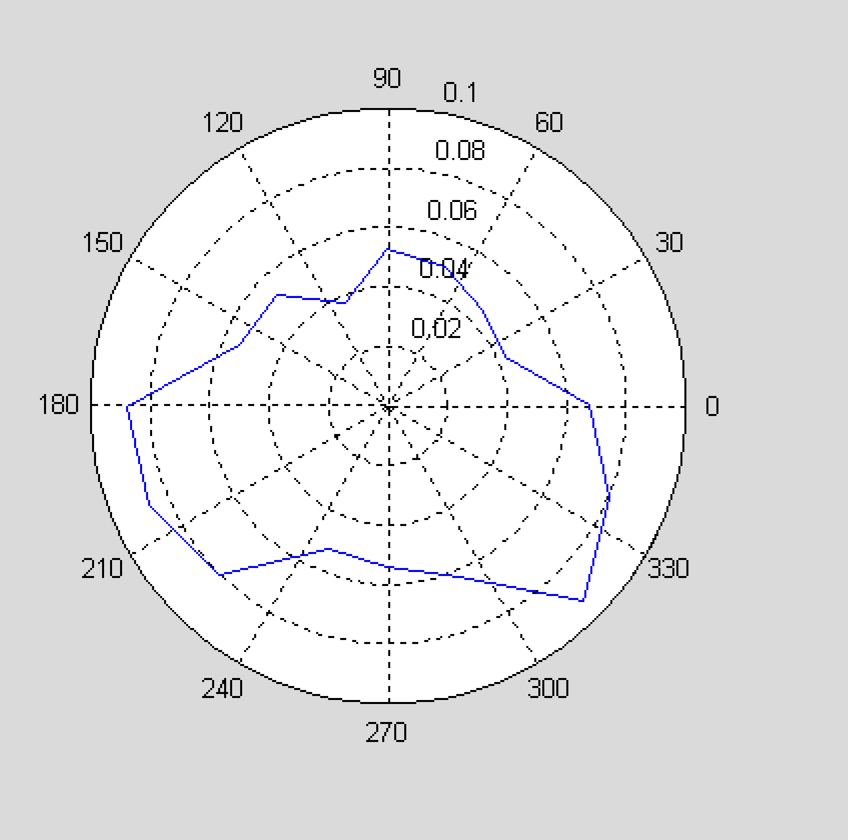
Polar graphs

- In some applications we need to depict data which has an angle dependence.
  - If you were designing navigational software for a yacht you would need to know how often the wind blows from each direction.
- A polar plot is one way to depict such data. The Matlab command for this is **polar(angleData, plotData)**.

Array of angles (in radians)

Array of data to be plotted

Polar graphs: Example



Plot represents the fraction of time the wind blows from each direction.

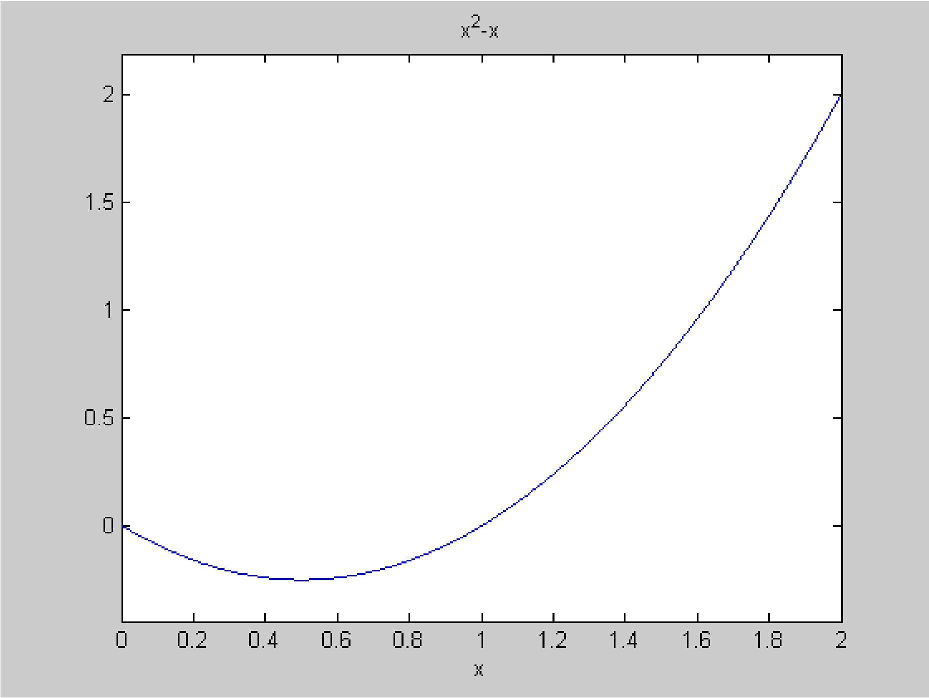
Created using:

polar(angles, fracWind)

Data represents wind directions in Evansville, IN.

Function Plotting

- Note that you can also plot functions directly (instead of building arrays with the function values and plotting them). To do so use the **ezplot** command.



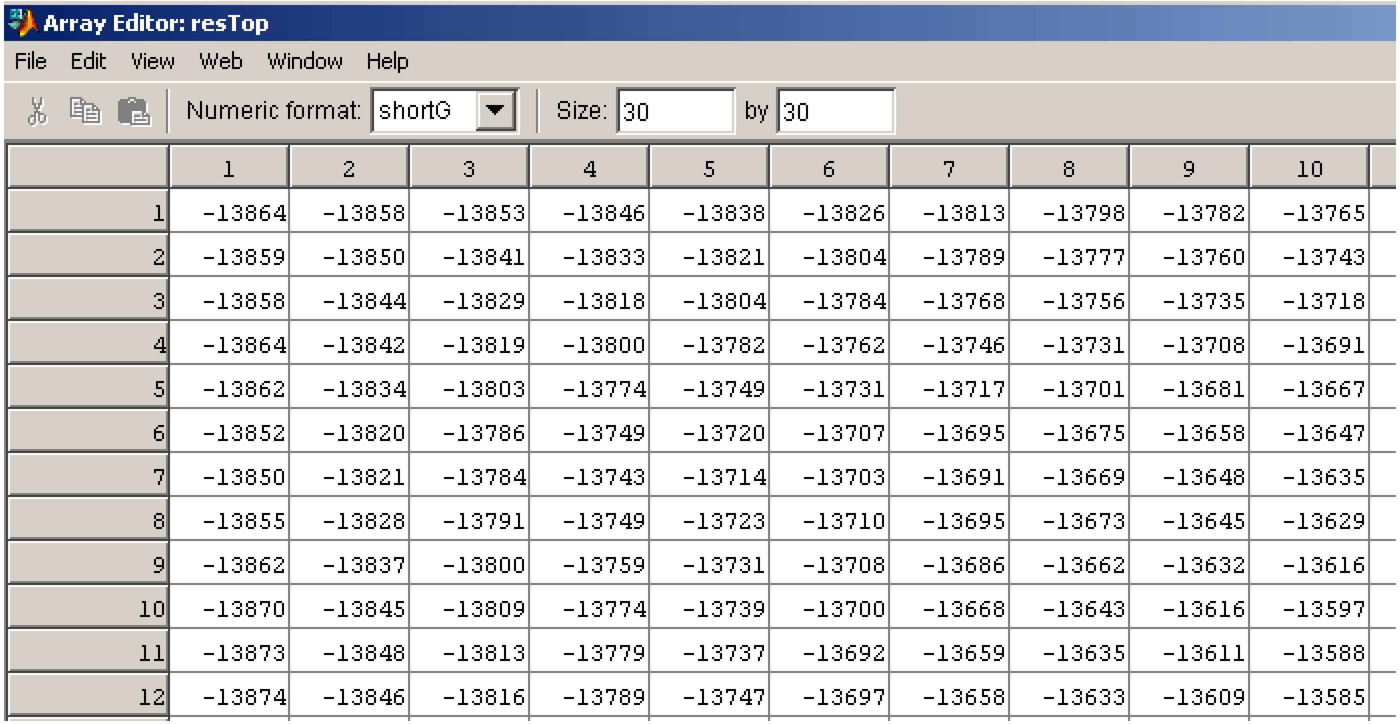

```
ezplot('x^2-x',[0,2])
```

function

domain

Plotting 2D Arrays

- Suppose we have a 2D array containing the depths to the top of an oil reservoir.


Array Editor: resTop

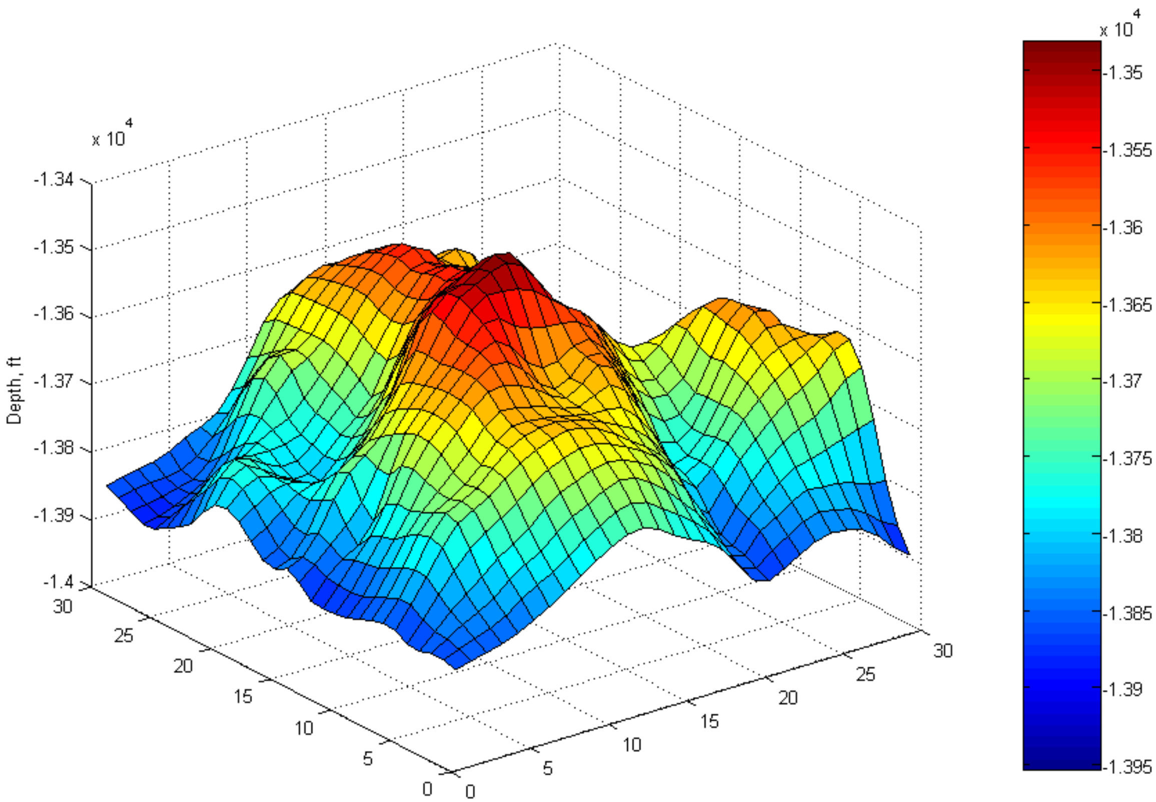
File Edit View Web Window Help

Numeric format: shortG Size: 30 by 30

	1	2	3	4	5	6	7	8	9	10
1	-13864	-13858	-13853	-13846	-13838	-13826	-13813	-13798	-13782	-13765
2	-13859	-13850	-13841	-13833	-13821	-13804	-13789	-13777	-13760	-13743
3	-13858	-13844	-13829	-13818	-13804	-13784	-13768	-13756	-13735	-13718
4	-13864	-13842	-13819	-13800	-13782	-13762	-13746	-13731	-13708	-13691
5	-13862	-13834	-13803	-13774	-13749	-13731	-13717	-13701	-13681	-13667
6	-13852	-13820	-13786	-13749	-13720	-13707	-13695	-13675	-13658	-13647
7	-13850	-13821	-13784	-13743	-13714	-13703	-13691	-13669	-13648	-13635
8	-13855	-13828	-13791	-13749	-13723	-13710	-13695	-13673	-13645	-13629
9	-13862	-13837	-13800	-13759	-13731	-13708	-13686	-13662	-13632	-13616
10	-13870	-13845	-13809	-13774	-13739	-13700	-13668	-13643	-13616	-13597
11	-13873	-13848	-13813	-13779	-13737	-13692	-13659	-13635	-13611	-13588
12	-13874	-13846	-13816	-13789	-13747	-13697	-13658	-13633	-13609	-13585

Plotting 2D Arrays

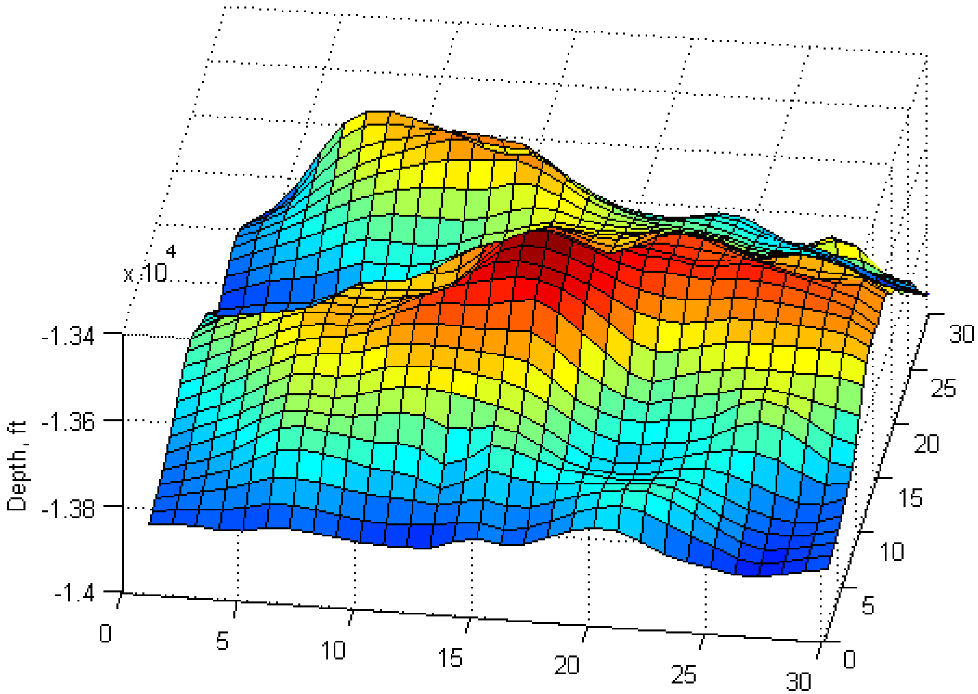
- It would be more useful to visualize this data in 2 or 3 dimensions. Use the **surf** command.




```
surf(resTop)
zlabel('Depth, ft')
colorbar
```

Surface Plots

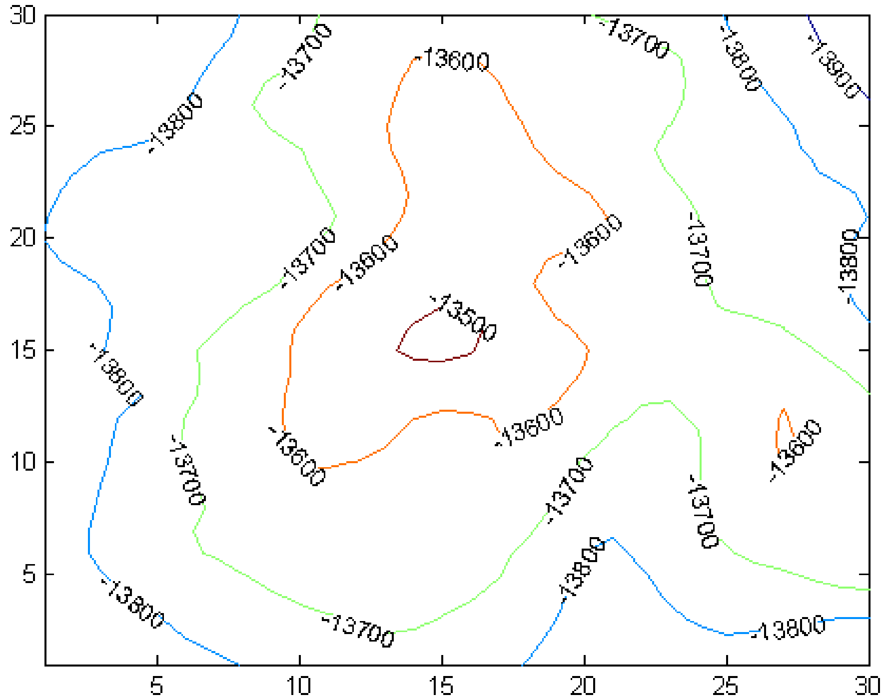
- You can view the data in a surface plot from other angles by rotating the plot using the mouse (choose *Tools->Rotate 3D* from the figure menu).



If you want to create a 2D plot which views the surface from directly above you can use pcolor instead of surf.

Contour Plots

- A contour plot is also a useful way to represent this kind of data. Matlab's **contour** command will create contour plots from data in a 2D array.



contour(resTop)

To fill the area between the contours with a color use contourf.

Using Meshgrid to create a mesh

- Some 2D and 3D plots need 2D arrays of *x* and *y* values. The **meshgrid** command generates these from 1D arrays.

1D

```
x = [ 1 2 3 4];
```

1D

```
y = [0 0.5 1];

[X,Y] = meshgrid(x,y)
```

```
X =
     1     2     3     4
     1     2     3     4
     1     2     3     4
```

2D

```
Y =
    0         0         0         0
    0.5000    0.5000    0.5000    0.5000
    1.0000    1.0000    1.0000    1.0000
```

2D

Quiver Plots

- Quiver plots are another useful way to represent many kinds of engineering data.
- These plots are useful for displaying vector quantities (e.g. velocity, electric or magnetic fields etc.) with arrows indicating both direction and magnitude.
- Quiver plots are often combined with surface plots and/or contour plots.

Quiver Plots

- Assume we have:
  - a 2D array variable containing a velocity)
  - 2D arrays of x and y grid points.

Array Editor: potential

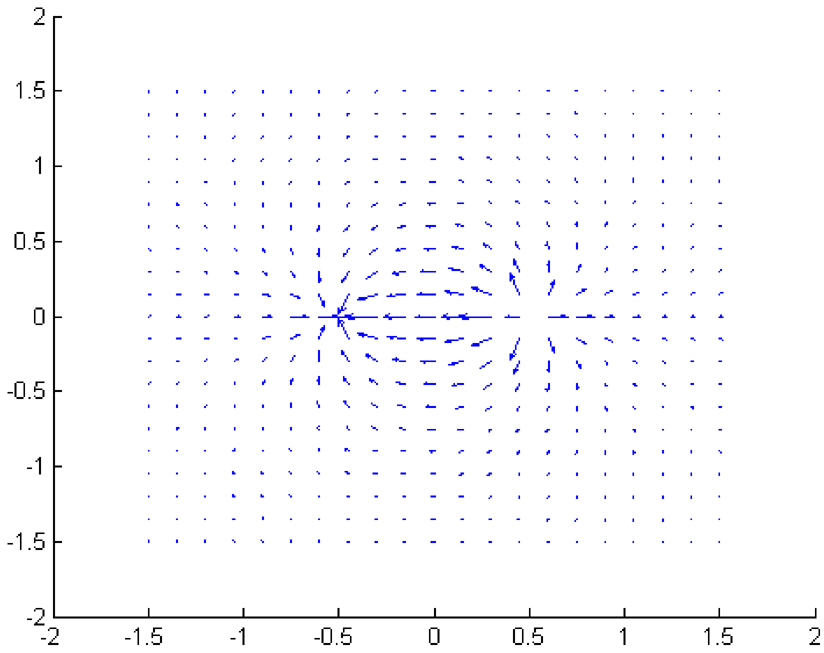
File Edit View Web Window Help

Numeric format: shortG Size: 21 by 21

	1	2	3	4	5	6
1	0.52038	0.51425	0.50062	0.47773	0.44391	0.39785
2	0.57622	0.57546	0.5665	0.54689	0.51409	0.46586
3	0.63803	0.6448	0.64302	0.6293	0.59983	0.55092
4	0.70553	0.72259	0.7314	0.72746	0.70516	0.65849
5	0.77776	0.80842	0.83235	0.84386	0.83501	0.79604
6	0.85273	0.90063	0.94532	0.98031	0.99494	0.97384
7	0.92706	0.9956	1.0673	1.1363	1.1901	1.2057
8	0.99573	1.0869	1.1908	1.3056	1.4212	1.5089
9	1.0523	1.165	1.3024	1.471	1.6744	1.8962

Quiver Plots

- To create the quiver plot enter:



quiver(X,Y,dpdx,dpdy)

The arrows on the quiver plot are vectors with components dp/dx and dp/dy

Putting Plots into Documents

- If you want to put your plot into another document (such as a Microsoft Word document) first choose *Edit->Copy Figure* from the menu on the figure.
- The figure can then be pasted into the other document.

Figure Backgrounds

- If you are pasting figures into other documents it is often nicer to use a white background for the figure.

- You can set this in the *Edit->Copy Options* menu (choose “Force white background”)

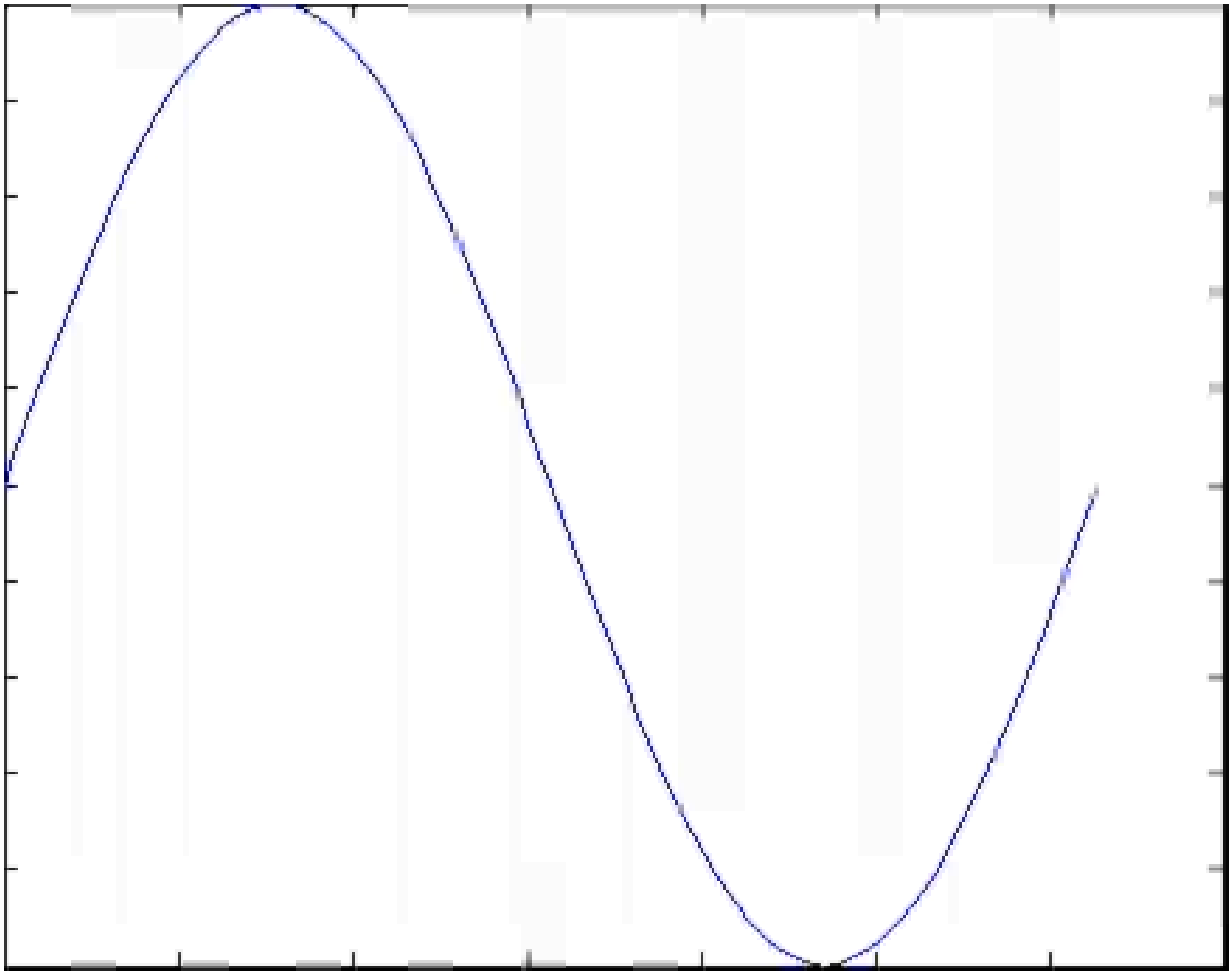
Animation

- Animation is quite simple in Matlab … just plot data repeatedly on a single figure.
- For example to plot the function *y=sin(x+t)*

For loop counting over an array of different times t=0,0.1,0.2, …, 9.9, 10.

Pause of 0.2 seconds between frames.

```
x = 0:2*pi/100:2*pi;
for t=0:0.05:5
    y=sin(x+t);
    plot(x,y)
    pause(0.2)
end
```



Movie Generation

- To create a movie a sequence of frames are “grabbed”, stored in an array and written out as a .avi file.

```
%initialise frame counter
nFrame = 1;
x = 0:2*pi/100:2*pi;
for t=0:0.05:5
    y=sin(x+t);
    plot(x,y)
    pause(0.2)
    % grab frame and store
    movieData(nFrame) = getframe;
    nFrame = nFrame + 1;
end
% output movie
movie2avi(movieData,'animation.avi');
```

Optional Reading

Chapter 7 Graphics and Image Processing	Introduction to Matlab 7 for Engineers (2nd ed)		A Concise Introduction to Matlab (1st ed)	
Topic	**Section**	**Pages**	**Section**	**Pages**
Plotting basics	5.1	259-265 269-271	5.1	205-207 209-211
Subplots and hold	5.2	271-276 279-280	5.2	211-216
Log graphs	5.3	282-285	5.2	217-219
Polar plots	5.3	290-291	5.2	220-221
Surfaces and contour plots	5.8	335-385	5.7	251-254
Animation	B.1	661-663		