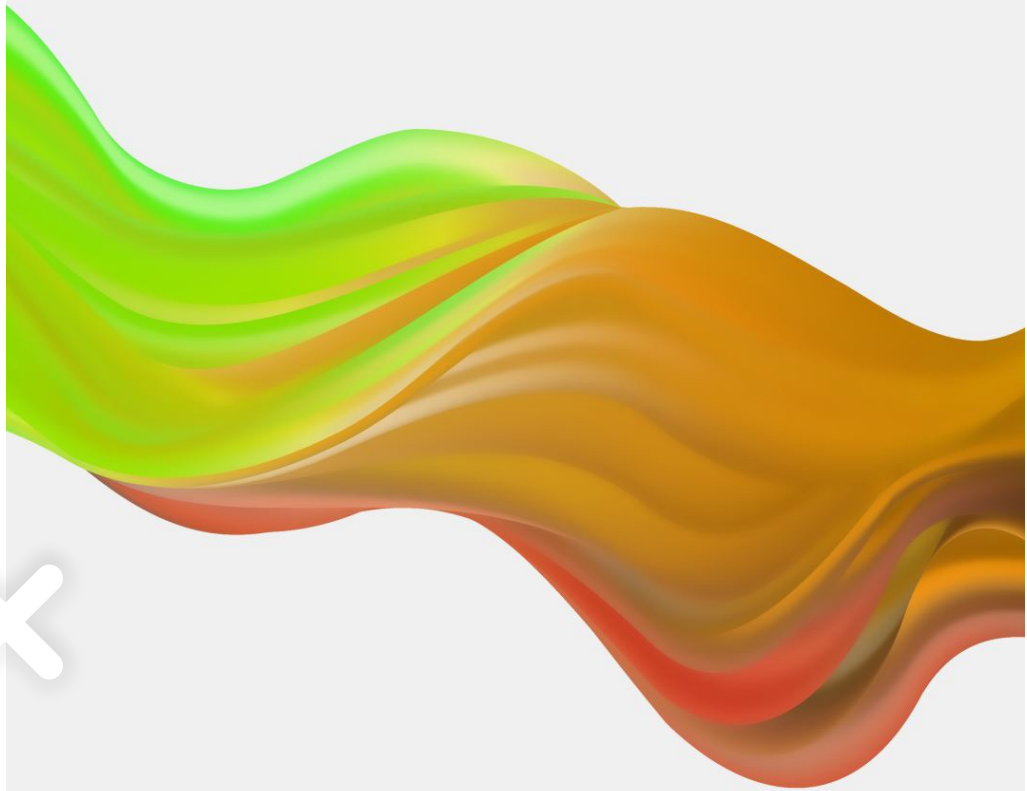
Computer Programming

MENG2020



REVIEW

Plots in 2D and 3D

Plotting Vectors

- Let's create a vector v1.

```
>> x = [0:0.04*pi:10];
>> y = sin(x);
```

- You can plot the value against the index (1,2,3...).

```
>> plot (y);
```

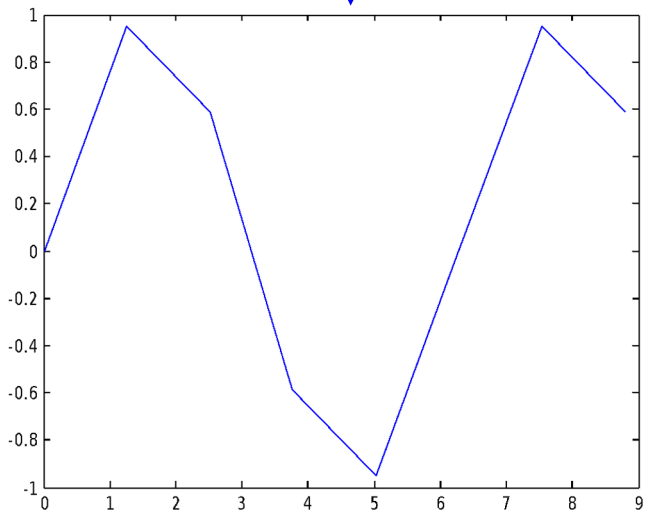
- Or you can plot against another variable (the most common usage).

```
>> plot (x,y);
```

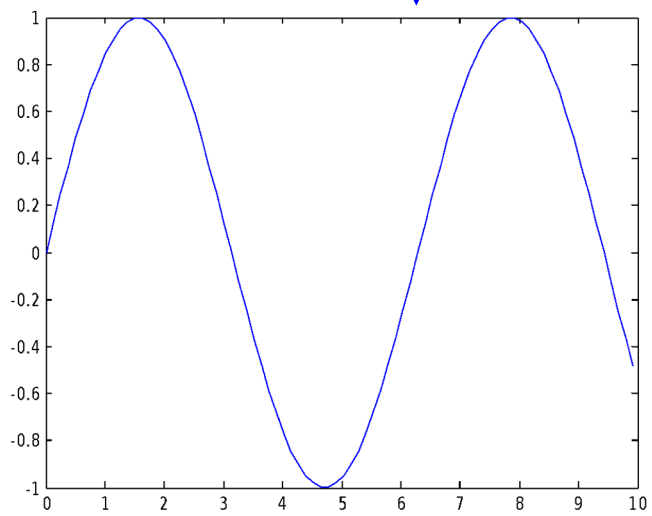
The plot Command

- **plot** generates dots at each (x,y) pair and then connects the dots with a line. To make the plot look smoother, just use more points.

```
>> x = [0:0.4*pi:10];
```




```
>> x = [0:0.04*pi:10];
```



The plot Command

- x and y vectors must be same size or else you'll get an error.

```
>> plot ([1 2], [1 2 3]);
```

- You can change the line color, marker style, and line style by adding a string argument.
  Ex: `plot (x,y,'k.-');`

- You can plot without connecting the dots by omitting the line style argument.
  Ex: `plot (x,y,'.');`

Plot Line Specifiers

Line specifiers define the style and color of lines, and marker types.

STYLES

Line Style	Specifier
solid (default)	-
dashed	--

Line Style	Specifier
dotted	:
dash-dot	-.

DEFAULTS

COLOURS

Line Color	Specifier
red	r
green	g
blue	b
cyan	c

Line Color	Specifier
magenta	m
yellow	y
black	k
white	w

Plot Marker Types

Marker Type	Specifier		Marker Type	Specifier
plus sign	+		square	s
circle	o		diamond	d
asterisk	*		five-pointed star	p
point	.		six-pointed star	h
cross	x		triangle (pointed left)	<
triangle (pointed up)	^		triangle (pointed right)	>
triangle (pointed down)	v			

Plot Command Syntax

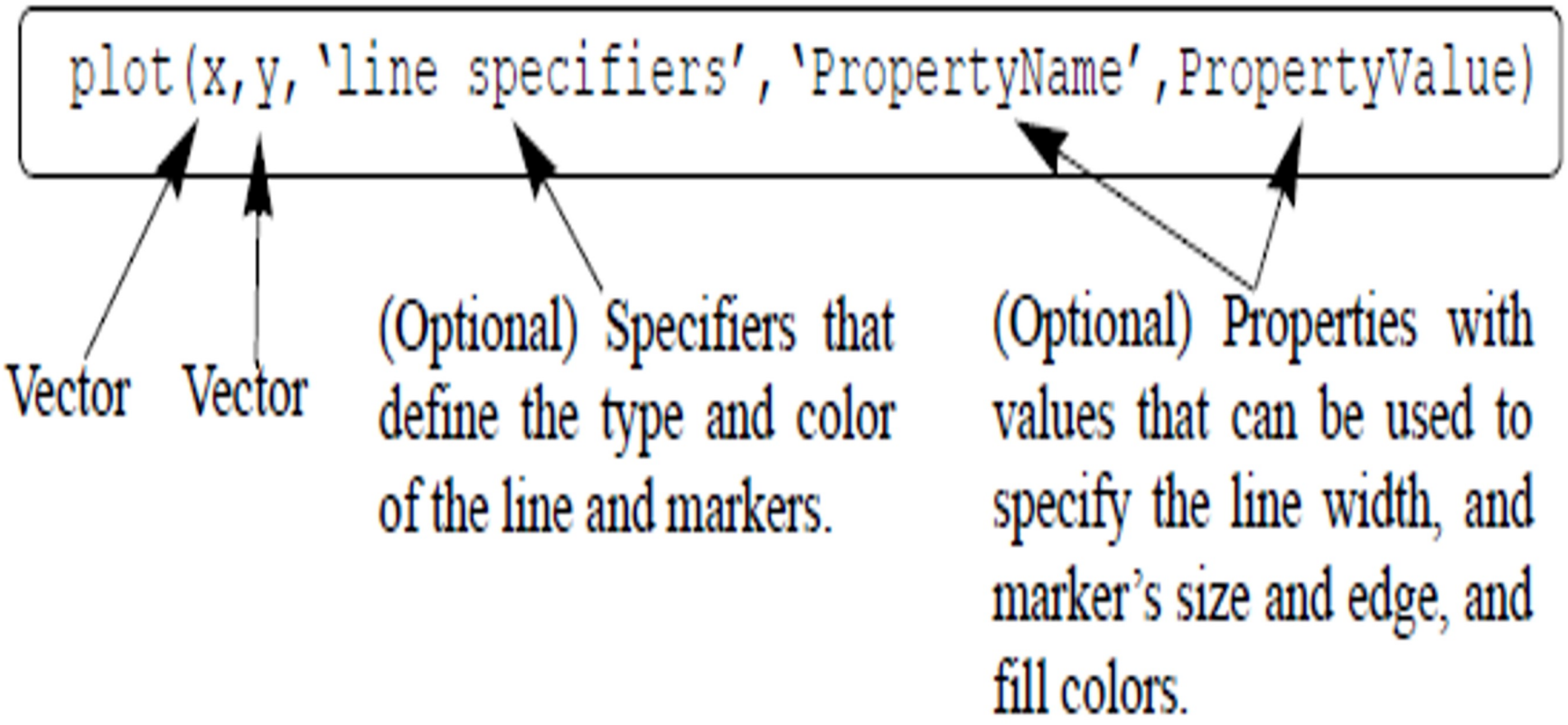
Notes about using the specifiers:

- The specifiers are typed inside the `plot` command as strings.
- Within the string the specifiers can be typed in any order.
- The specifiers are optional. This means that none, one, two, or all the three can be included in a command.

Some examples:

`plot(x,y)`	A blue solid line connects the points with no markers (default).
`plot(x,y,'r')`	A red solid line connects the points.
`plot(x,y,'--y')`	A yellow dashed line connects the points.
`plot(x,y,'*')`	The points are marked with * (no line between the points).
`plot(x,y,'g:d')`	A green dotted line connects the points that are marked with diamond markers.

The *plot* Command Generic Syntax



Plot Property Names and Values

In the *plot* command, type the property name in quotes marks, then a comma, then a value. Repeat for multiple properties.

Property Name	Description	Possible Property Values
`LineWidth` (or `linewidth`)	Specifies the width of the line.	A number in units of points (default 0.5).
`MarkerSize` (or `markersize`)	Specifies the size of the marker.	A number in units of points.
`MarkerEdgeColor` (or `markeredgecolor`)	Specifies the color of the marker, or the color of the edge line for filled markers.	Color specifiers from the table above, typed as a string.
`MarkerFaceColor` (or `markerfacecolor`)	Specifies the color of the filling for filled markers.	Color specifiers from the table above, typed as a string.

For example, the command:

```
plot(x,y,'-mo','LineWidth',2,'markersize',12,
      'MarkerEdgeColor','g','markerfacecolor','y')
```

creates a plot that connects the points with a magenta solid line and circles as markers at the points. The line width is two points and the size of the circle markers is 12 points. The markers have a green edge line and yellow filling.

EXAMPLE

YEAR	1988	1989	1990	1991	1992	1993	1994
SALES (millions)	8	12	20	22	18	24	27

```
>> yr=[1988:1:1994];
>> sle=[8  12  20  22  18  24  27];
>> plot(yr,sle,'--r*','linewidth',2,'markersize',12)
>>
```

Line Specifiers: dashed red line and asterisk marker.

Property Name and Property Value: the line width is 2 points and the markers size is 12 point.

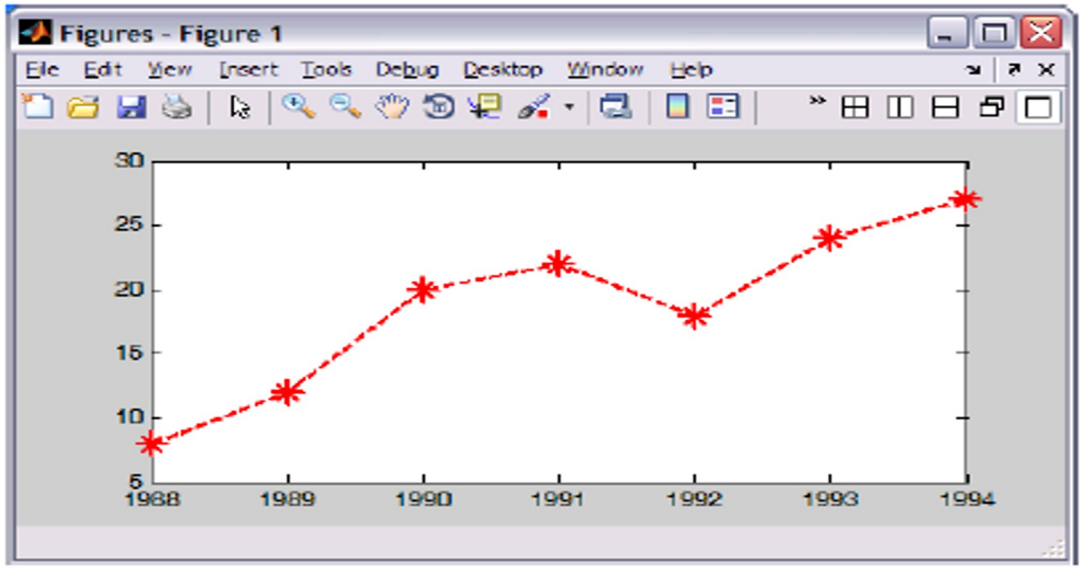


Figure 5-3: The Figure Window with a plot of the sales data.

The plot Command Hints

- Look at *help plot* for a full list of colours, markers, and line styles.
- You can customize your plot by using the menus in the figure window.
- Figures can be saved in many formats like bmp, gif, png, jpg or pdf.

Axes Options for Plots

You can turn on the grid to make it easier to read values.

```
>> grid on;
```

You can also turn off the grid.

```
>> grid off;
```

xlim sets the x axis limits.

```
>> xlim ([-pi pi]);
```

ylim sets the y axis limits.

```
>> ylim ([-1 1]);
```

Other Axes Options for Plots

```
>> axis square;
```
Axes look like a box without changing the scale.

```
>> axis tight;
```
Fits axes to data with no wasted space (default).

```
>> axis equal;
```
Makes the x and y axes with the same scale.

```
>> axis xy;
```
Places the origin (0,0) at the bottom-left corner (default).

```
>> axis ij;
```
Places the origin (0,0) at the top-left corner (good for viewing matrices).

Multiple Plots on Multiple Figures

Use the **figure** command to open a new figure or to activate an already opened figure.

```
>> figure;
%creates a new blank figure and
activates it.

>> figure(1);
%activates figure 1 or creates it
if it doesn't exist.
```

Closing Figure Windows

Use the **close** command to close figure windows.

`close`: Closes the active figure window.

`close(n)`: Closes the figure window **n**.

`close all`: Closes all the opened figure windows.

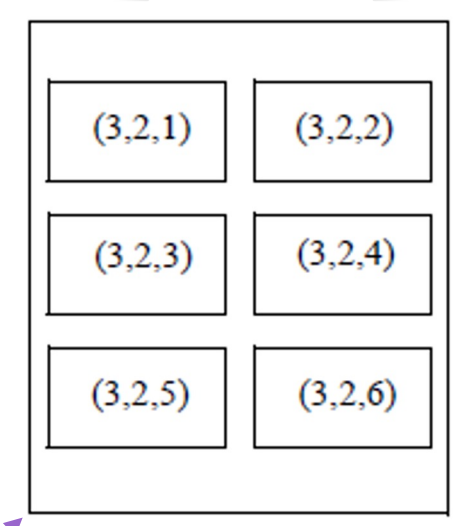
Multiple Plots on a Figure

You can have multiple plots in the same figure. For example,

```
>> subplot(3,2,1);
```

makes a figure with 3 rows and 2 columns of axes, and activates the first subplot.

*The subplots are numbered from left to right and top to bottom, with the upper left being subplot 1 and the lower right subplot m*n.*



Subplot numbers for 3x2 set of subplots

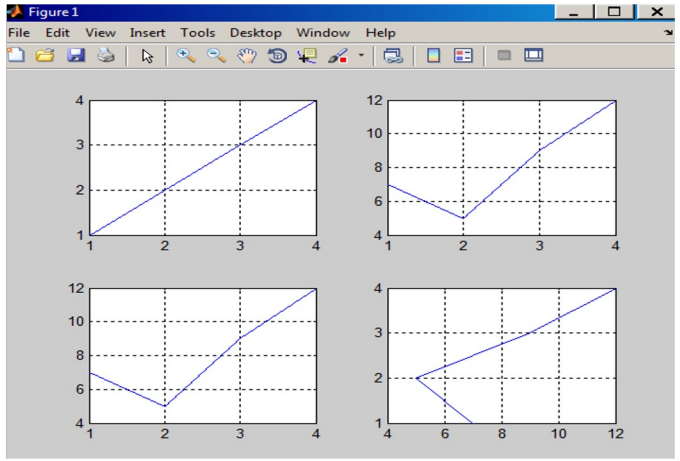
Multiple Plots on a Figure

```
x = [1 2 3 4]
y = [7 5 9 12]
subplot (2,2,1)
plot (x)
grid
subplot (2,2,2)
plot (y)
grid
subplot (2,2,3)
plot (x,y)
grid
subplot (2,2,4)
plot (y,x)
grid
```

subplot(m,n,p)

*If subplots don't exist, subplot creates them and makes subplot **p** the current subplot.*

*If subplots already exist, subplot makes subplot **p** the current one.*



EXAMPLE

```
%  A script file that creates a plot of
%  the function: 3.5.^(-0.5*x).*cos(6x)
x=[-2:0.01:4];
y=3.5.^(-0.5*x).*cos(6*x);

plot(x,y)
```

Create vector x with the domain of the function.

Create vector y with the function value at each x.

Plot y as a function of x.

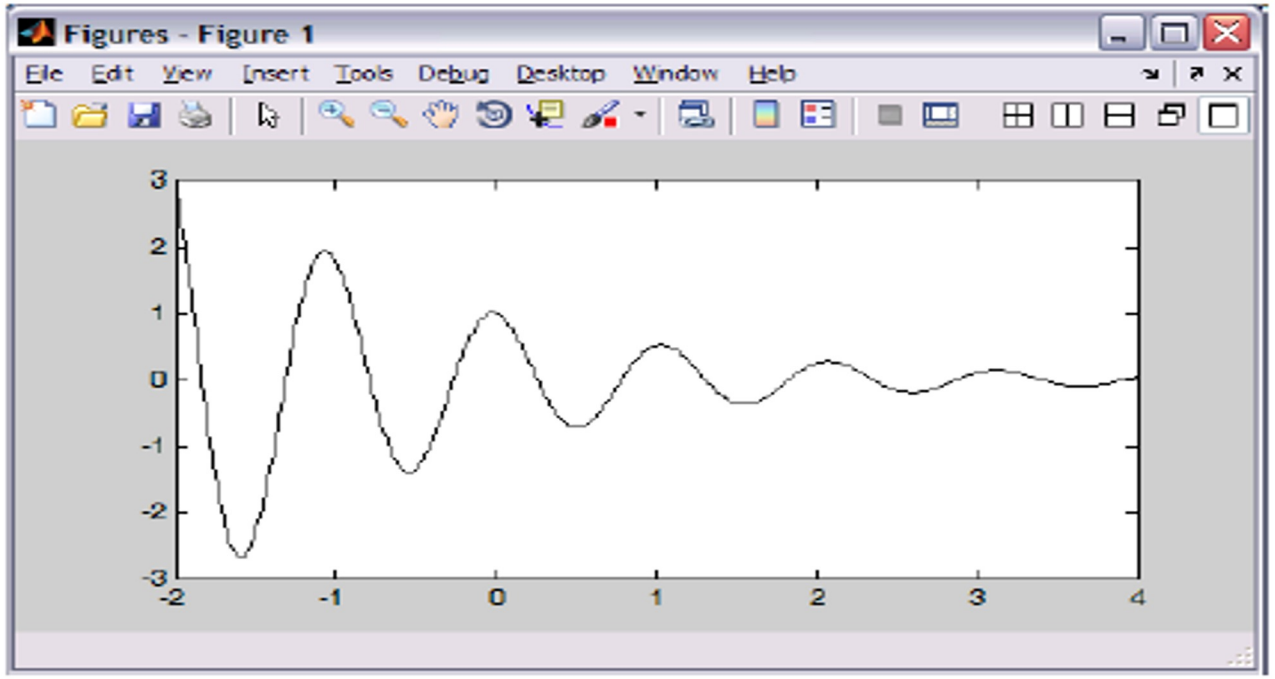
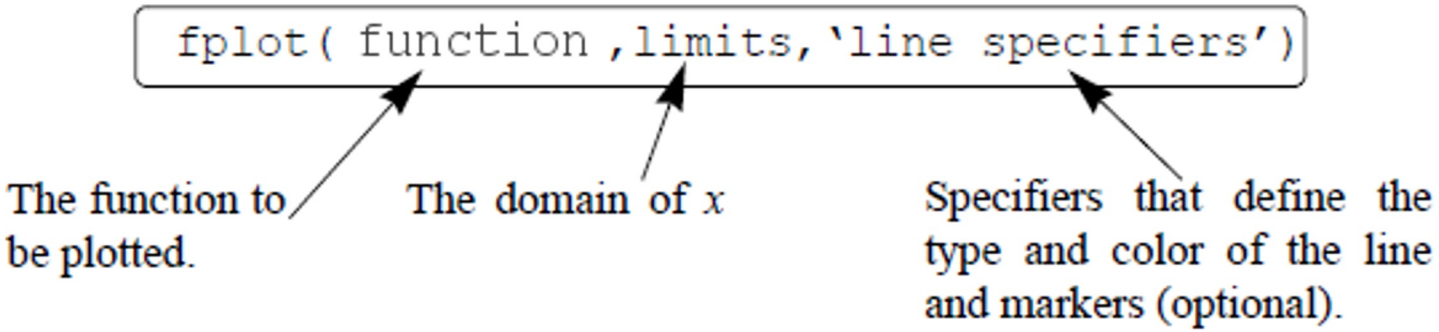


Figure 5-4: The Figure Window with a plot of the function: $y = 3.5^{-0.5x}\cos(6x)$.

Plotting a Function with *fplot*

The **fplot** command plots the curve defined by a function like $y = f(x)$ between given limits on the x axis.



The function is like a formula and is specified like this:

```
formula = @(x) x.^2 + 4.*sin(2.*x) - 1
```

The function variable

Always @(independent variable)

The formula containing the independent variable

Plotting a Function with *fplot*: *Example*

```
>> fplot(formula,[-3,3],'k')
```

function — limits of x — line specifiers (same as **plot**)

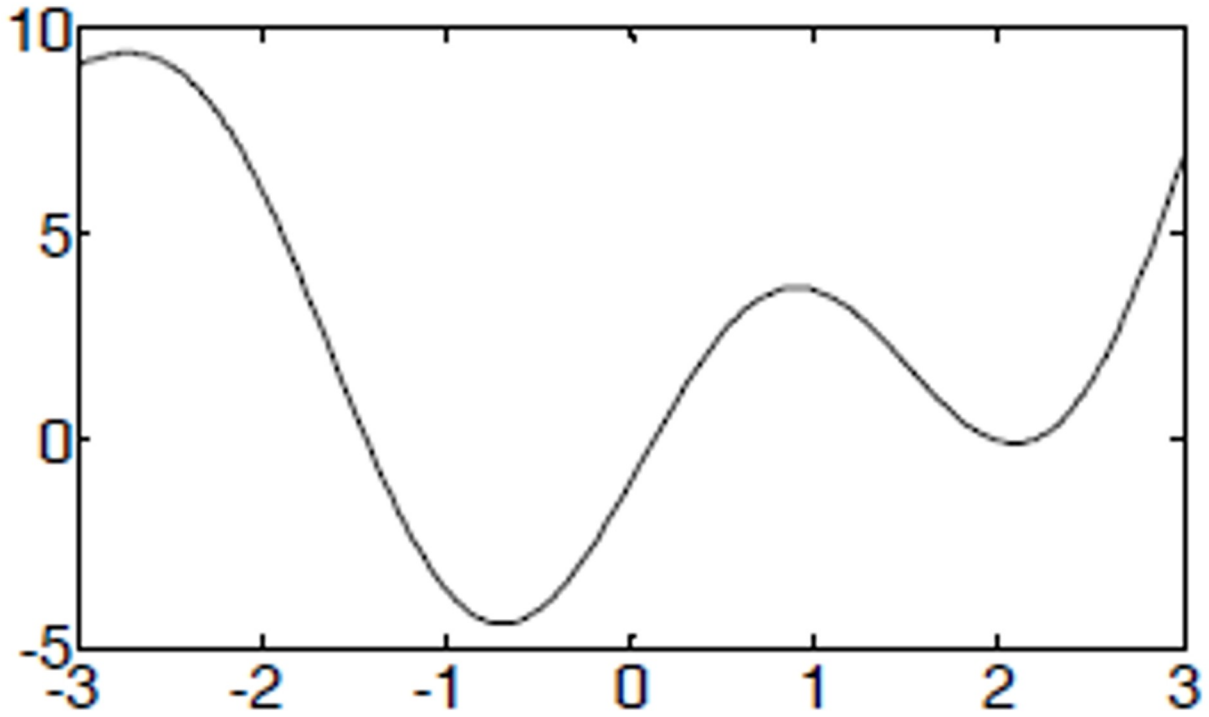


Figure 5-6: A plot of the function $y = x^2 + 4\sin(2x) - 1$.

Putting Multiple Plots on the Same Figure (This is different than subplots!)

Method #1: Multiple Pairs of Vectors

Here is an example for 3 plots on the same figure:

```
>> plot(x,y,u,v,t,h)
```

It plots **y** vs. **x**, **v** vs. **u**, and **h** vs. **t**.

The vectors of ***each*** pair must be same size b*ut it can be different than sizes in other pairs.*

It is recommended to use line specifiers to distinguish between the plots:

```
>> plot(x,y,'-b', u,v,'--r', t,h,'g:')
```

METHOD 1

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b',x,yd,'--r',x,ydd,':k')
```

Create vector x with the domain of the function.

Create vector y with the function value at each x.

Create vector yd with values of the first derivative.

Create vector ydd with values of the second derivative.

Create three graphs, y vs. x, yd vs. x, and ydd vs. x in the same figure.

The plot that is created is shown in Figure 5-7.

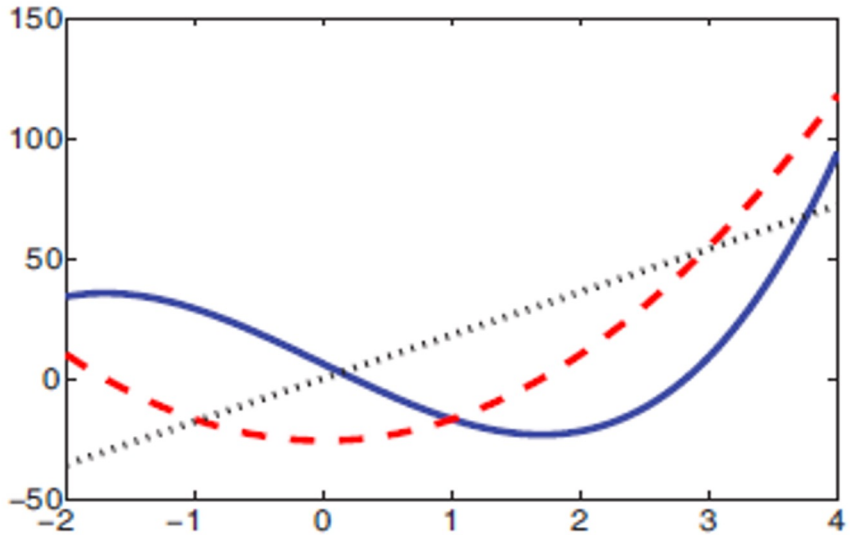


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

Putting Multiple Plots on the Same Figure

Method #2: Use the *hold on* Command

Normally, each time you execute *plot* or *fplot* it erases the previous plot and draws a new one. ***hold on*** changes that.

1. Issue the command **hold on**.
2. Call **plot** or **fplot** for each of the remaining graphs.
3. Issue the command **hold off**.
4. Graphs drawn after **hold on** are added to the plot.
   Graphs drawn after **hold off** erase the plot.
5. Draw the first graph with **plot** or **fplot**.

METHOD 2

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b')
hold on
plot(x,yd,'--r')
plot(x,ydd,':k')
hold off
```

The first graph is created.

Two more graphs are added to the figure.

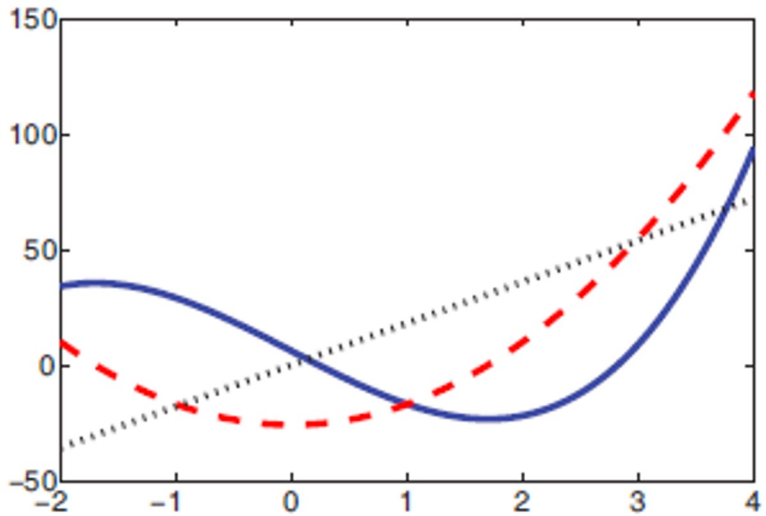


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

Multiple Plots with the *line* Command

Method 3: Using the *line* command

The line command is another method still to add additional graphs to an existing plot.

```
line(x,y,'PropertyName','PropertyValue')
```

Example:

```
>> line(x,y,'linestyle','--','color','r','marker','o')
```

Adds a graph drawn with a dashed red line and circular markers to the current plot.

METHOD 3

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'LineStyle','-','color','b')
line(x,yd,'LineStyle','--','color','r')
line(x,ydd,'linestyle',':','color','k')
```

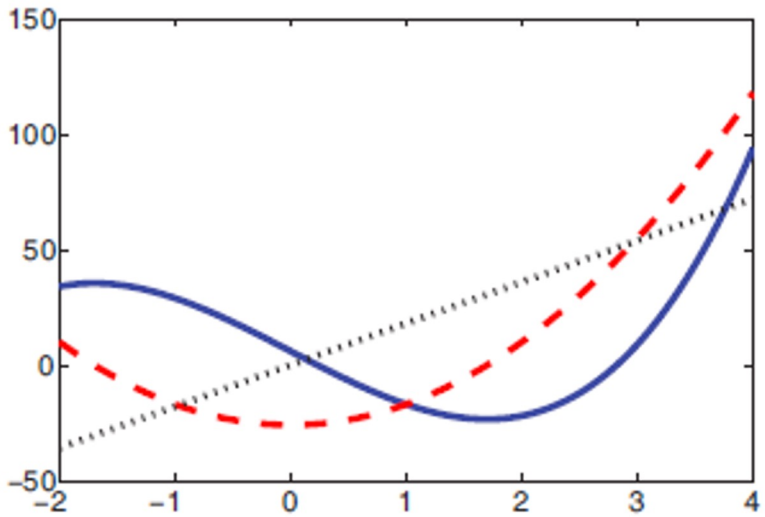


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

About our lecture

Formatting a Plot Using Commands

You can easily format your figure by using the options in the menu of the figure window but you can also format your figure by programming using MATLAB commands.

xlabel('some text'): Writes a label below the horizontal axis.

- Example: `xlabel('Time (sec)')`

ylabel('some text'): Writes a label to the left of the vertical axis.

- Example: `ylabel('Current (mA)')`

Formatting a Plot Using Commands

title('Some text'): Writes a title above the plot.

- Example: `title('Diode Current')`

text(x,y,'Some text'): Places a text in the figure with the first character at coordinates (x,y).

- Example: `text(x,y,'Peak 3.5 sec after first')`

gtext('Some text'): The figure window opens and the user clicks on the graph to specify where to put the text.

- Example: `gtext('Final Graph')`

Putting a Legend on a Plot

```
legend('text1','text2',...,'location',loc)
```

Examples:

```
legend ('this year', 'last year')
legend ('this year', 'last year', 'location', 'NorthWest')
```

For each graph (data set), it displays a short line in the same style as the graph line and adds the specified text. It is most useful for plots having at least two graphs.

'**location**' and **loc** are optional arguments that specify where to put the legend (top-right corner if omitted).

Legend Location Property

'north'	Inside top of axes
'south'	Inside bottom of axes
'east'	Inside right of axes
'west'	Inside left of axes
'northeast'	Inside top-right of axes (default for 2-D axes)
'northwest'	Inside top-left of axes
'southeast'	Inside bottom-right of axes
'southwest'	Inside bottom-left of axes
'northoutside'	Above the axes
'southoutside'	Below the axes
'eastoutside'	To the right of the axes
'westoutside'	To the left of the axes
'northeastoutside'	Outside top-right corner of the axes (default for 3-D axes)
'northwestoutside'	Outside top-left corner of the axes
'southeastoutside'	Outside bottom-right corner of the axes
'southwestoutside'	Outside bottom-left corner of the axes
'best'	Inside axes where least conflict with data in plot
'bestoutside'	Outside top-right corner of the axes (when the legend has a vertical orientation) or below the axes (when the legend has a

Plot Window Text Formatting

You can change the appearance of the text displayed by the *xlabel*, *ylabel*, *title*, *text*, and *legend* commands.

You can change the font, size, text color, background color, subscript and superscript, positions, and style (like bold, italic, etc...).

You can also display Greek letters.

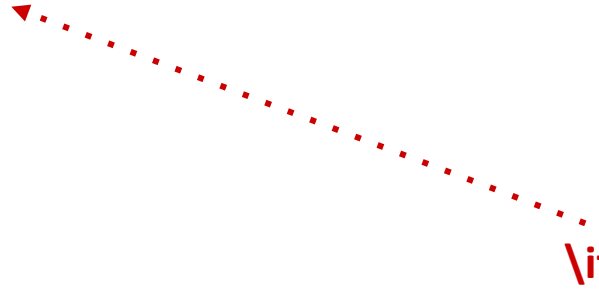
Plot Window Text Formatting

You can format using modifiers within the text itself or by adding property names and values to the command.

All the text that follows the modifier gets modified.

To only modify a specific part of the text, open a brace ({), then the write the modifier, then the text-to-be-modified, and finally close the brace (}).

For example, you can make the word Good in italics by writing `'Life Is {\it Good}'`



Plot Window Text Formatting : Common Modifiers

`\bf`: **bold face**

`\it`: *italic*

`\rm`: normal font

`\fontname{fontname}`: name of the font (details coming in a few slides).

`\fontsize{fontsize}`: size of the font, a higher number means a larger font.

`\color{colorname}`: the font color, must be red, green, yellow, magenta, blue, black, white, gray, darkGreen, orange, or lightBlue.

Plot Window Text Formatting : Examples

All text in italics:

```
title ('\it If you come to a fork in the road, take it.')
```

If you come to a fork in the road, take it.

Only the word fork in italics:

```
title('If you come to a {\it fork} in the road, take it.')
```

If you come to a *fork* in the road, take it.

Only the work fork in bold:

```
title('If you come to a {\bf fork} in the road, take it.')
```

If you come to a **fork** in the road, take it.

These examples and the one on the next slide use the title command but the text modifiers work with all plot text commands like text, xlabel, ylabel, and legend.

Text Formatting : Subscript and Superscript

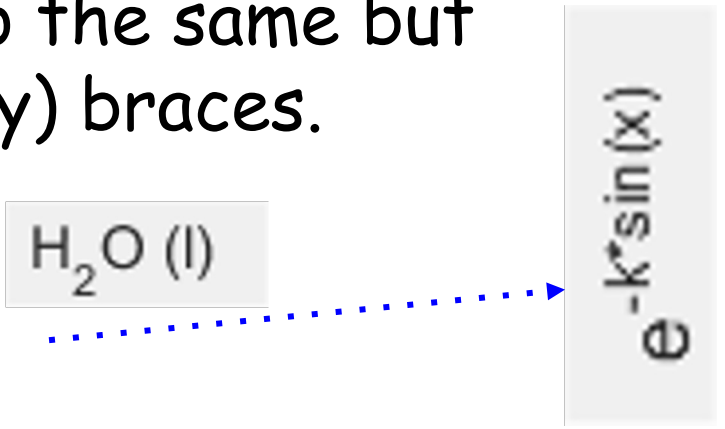
Subscript and superscripts are useful for mathematical and chemical formulas.

To make a single character a subscript: precede it by an underscore (_).

To make a single character a superscript: precede it by a caret(^).

For multiple characters, you do the same but enclose the characters in (curly) braces.

```
xlabel('H_2O (l)')
ylabel('e^{-k*sin(x)}')
```



Text Formatting : Greek Letters

To make a Greek letter, follow the backslash with the letter name in English like **\alpha**. α

A name in lowercase letters makes a lowercase Greek letter like **\gamma**. γ

A name with a capitalized first letter makes an uppercase Greek letter like **\Sigma**. Σ

Greek Letters on Plots

```
ylabel('Standard deviation (\sigma) of resistance in M\Omega')
```

Will write **Standard deviation (σ) of resistance in MΩ** next to the vertical axis.

Characters in the string	Greek Letter
\alpha	α
\beta	β
\gamma	γ
\theta	θ
\pi	π
\sigma	σ

Characters in the string	Greek Letter
\Phi	Φ
\Delta	Δ
\Gamma	Γ
\Lambda	Λ
\Omega	Ω
\Sigma	Σ

Text Formatting with Properties and Values

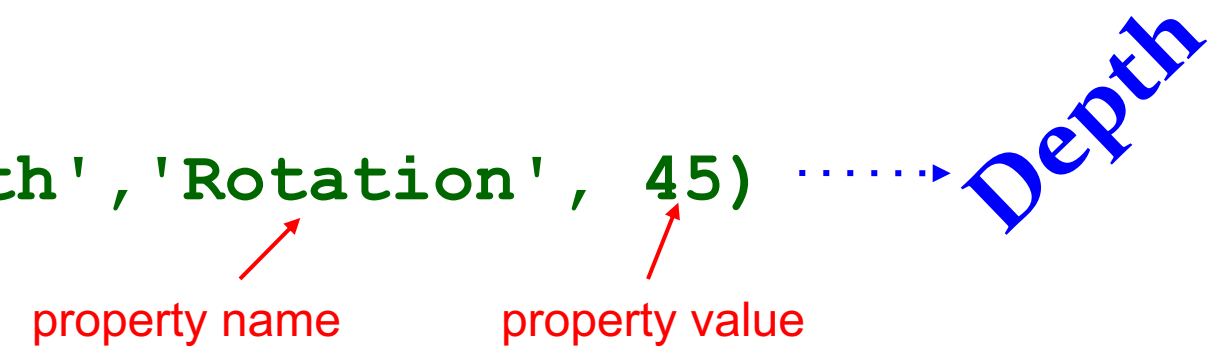
Instead of text modifiers, you can also change the display of the entire text string by using **propertyName, propertyValue** pairs.

```
text(x,y,'Some text',PropertyName,PropertyValue)
```

- **PropertyName** is a text string.
- **PropertyValue** is a number if the value is a number or a text string if the value is a letter or a word.

Example:

```
text(x,y,'Depth','Rotation', 45)
```



Property Name	Description	Possible Property Values
`Rotation`	Specifies the orientation of the text.	Scalar (degrees) Default: 0
`FontAngle`	Specifies italic or normal style characters.	`normal`, `italic` Default: `normal`
`FontName`	Specifies the font for the text.	Font name that is available in the system.
`FontSize`	Specifies the size of the font.	Scalar (points) Default: 10
`FontWeight`	Specifies the weight of the characters.	`light`, `normal`, `bold` Default: `normal`
`Color`	Specifies the color of the text.	Color specifiers (See Section 5.1).
`Background-Color`	Specifies the background color (rectangular area).	Color specifiers (See Section 5.1).
`EdgeColor`	Specifies the color of the edge of a rectangular box around the text.	Color specifiers (See Section 5.1). Default: none.
`LineWidth`	Specifies the width of the edge of a rectangular box around the text.	Scalar (points) Default: 0.5

Look at this script:

```
x=[10:0.1:22];
y=95000./x.^2;
xd=[10:2:22];
yd=[950   640   460   340   250   180   140];
plot(x,y,'-','LineWidth',1.0)
xlabel('DISTANCE (cm)')
ylabel('INTENSITY (lux)')
title('\fontname{Arial}Light Intensity as a Function of Distance','FontSize',14)
axis([8 24 0 1200])
text(14,700,'Comparison between theory and experiment.','EdgeColor','r','LineWidth',2)
hold on
plot(xd,yd,'ro--','linewidth',1.0,'markersize',10)
legend('Theory','Experiment')
hold off
```

Formatting text inside the `title` command.

Formatting text inside the `text` command.

It produces the plot on the next slide...

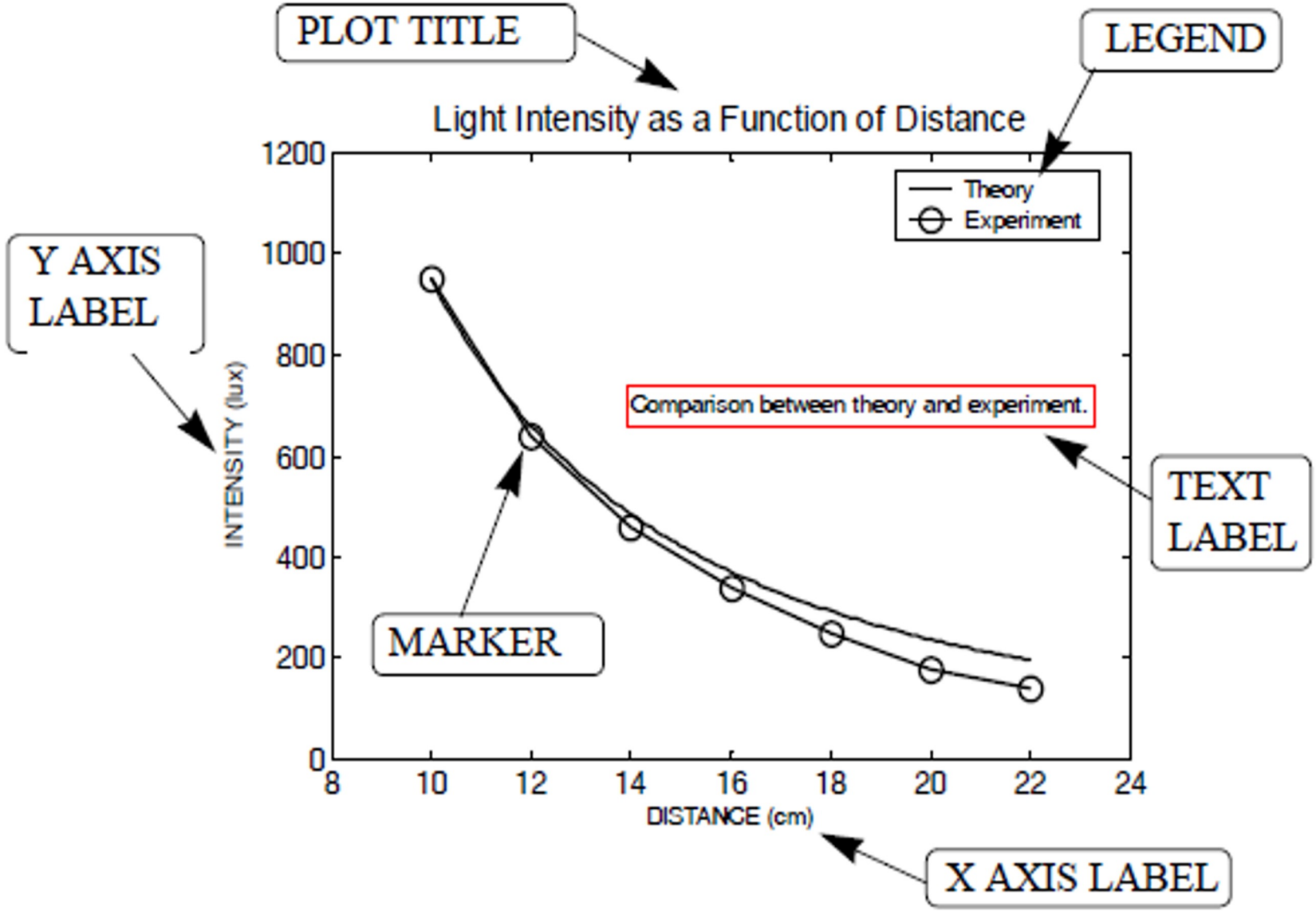


Figure 5-1: Example of a formatted two-dimensional plot.

Plots with Logarithmic Axes

- Sometimes we need a plot with one or both axes being logarithmic (log).
- They are used to display data with large range of values.
- They are also used to make some functional relationships more apparent.
- For example, `y = 2.^(-0.2*x+10)` gives a straighter line on a semilog plot.

Plots with Logarithmic Axes

MATLAB commands for log plots:

`semilogy(x,y)`	Plots y versus x with a log (base 10) scale for the y axis and linear scale for the x axis.
`semilogx(x,y)`	Plots y versus x with a log (base 10) scale for the x axis and linear scale for the y axis.
`loglog(x,y)`	Plots y versus x with a log (base 10) scale for both axes.

You can use line specifiers and *property-name, property-value* pairs as in the **plot** command.

On a logarithmic axis, make sure all data is larger than zero because otherwise log is undefined!

Plots with Logarithmic Axes

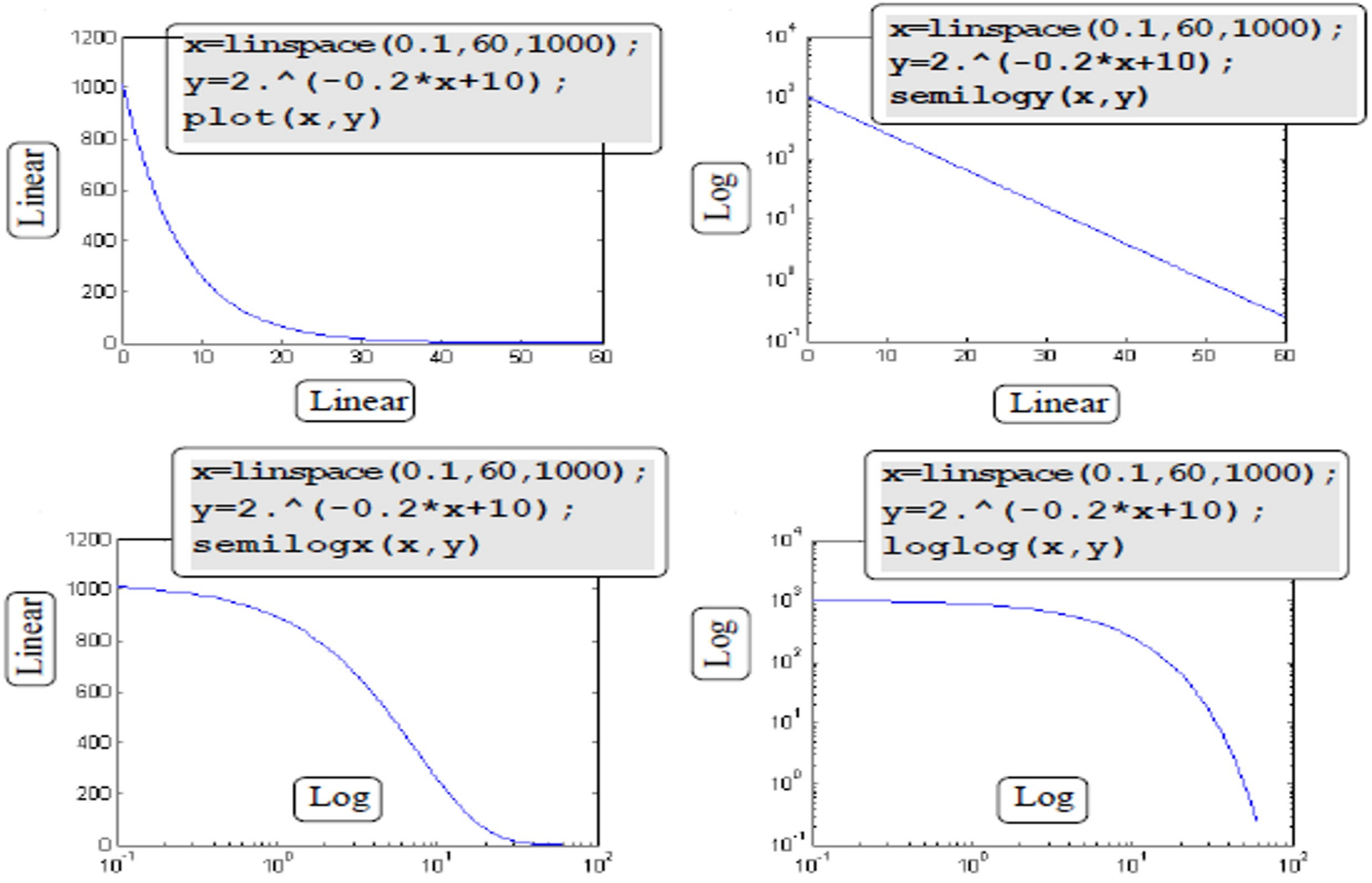


Figure 5-9: Plots of $y = 2^{(-0.2x + 10)}$ with linear, semilog, and loglog scales.

Plots with Error Bars

- Experimental data that is plotted usually also has some measure of the uncertainty in the measurements.
- Often shown by *error bars* (usually small vertical lines above and below data points), their size is the size of the uncertainty
- Uncertainty measure is often the *standard error*, which is approximately the standard deviation of the samples used to compute a data point.

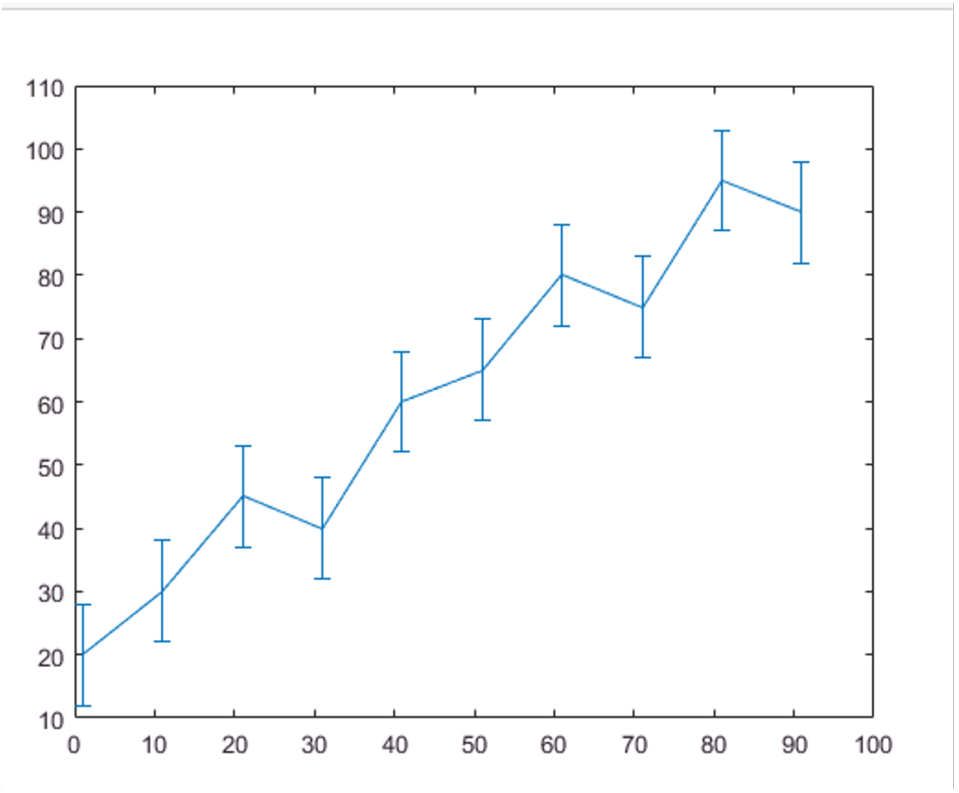
Plots with Error Bars

`errorbar(x,y,e)`

- All vectors in the command must be of the same size.
- x and y are the horizontal and vertical axis data.
- e is the error measurement at each point.
    - *At each y(i), MATLAB draws a vertical error bar from y(i)-e(i) to y(i)+e(i).*

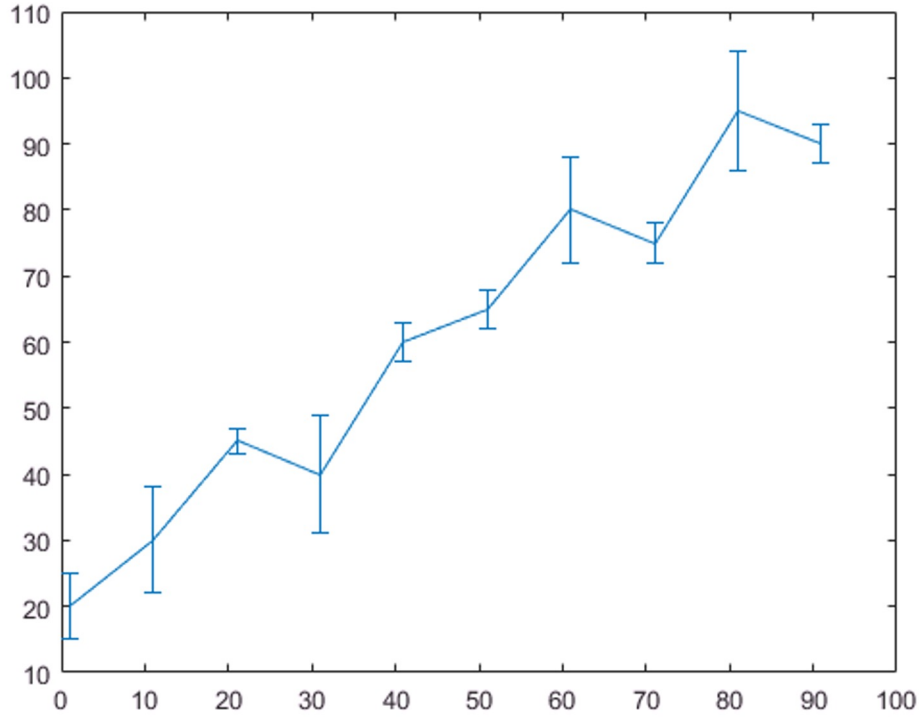
Plots with Error Bars : Example 1

```
x = 1:10:100;
y = [20 30 45 40 60 65 80 75 95 90];
err = 8*ones(size(y));
errorbar(x,y,err)
```



Plots with Error Bars : Example 2

```
x = 1:10:100;
y = [20 30 45 40 60 65 80 75 95 90];
err = [5 8 2 9 3 3 8 3 9 3];
errorbar(x,y,err)
```



Specialized Plots: Bar Plots

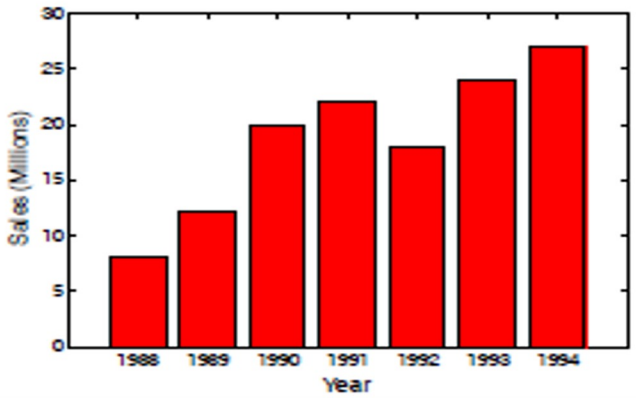
- MATLAB has lots of special types of plots like bar, stairs, stem, pie…

bar

Vertical Bar Plot

Function format:

```
bar(x,y)
```




```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
bar(yr,sle,'r')
xlabel('Year')
ylabel('Sales (Millions)')
```

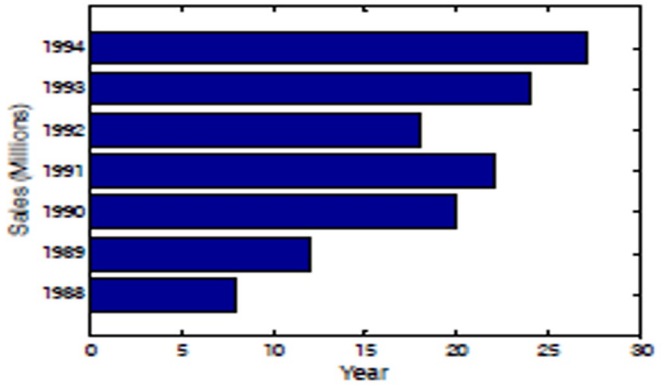
The bars are in red.

barh

Horizontal Bar Plot

Function format:

```
barh(x,y)
```

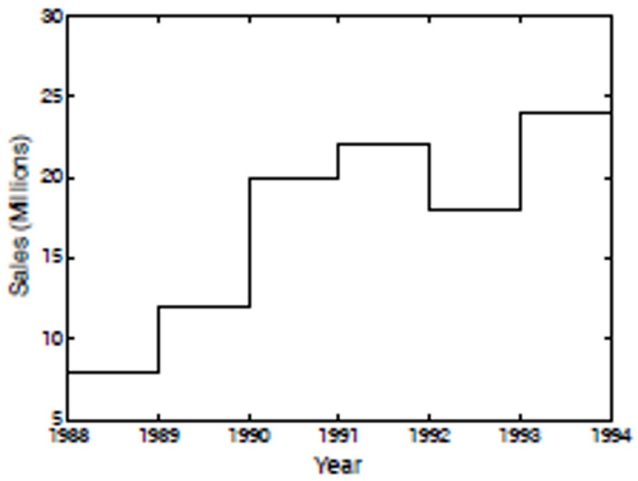


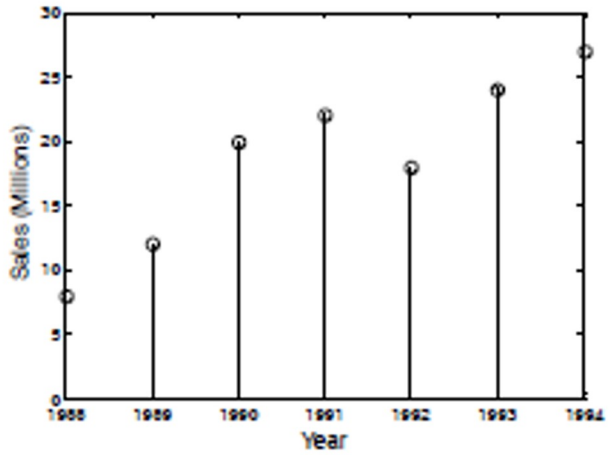

```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
barh(yr,sle)
xlabel('Sales (Millions)')
ylabel('Year')
```

Specialized Plots: Stairs and Stem Plots

stairs

stem

Stairs Plot
 	
Function format:
 	
`stairs(x,y)`
		`yr=[1988:1994];`
		`sle=[8 12 20 22 18 24 27];`
		`stairs(yr,sle)`

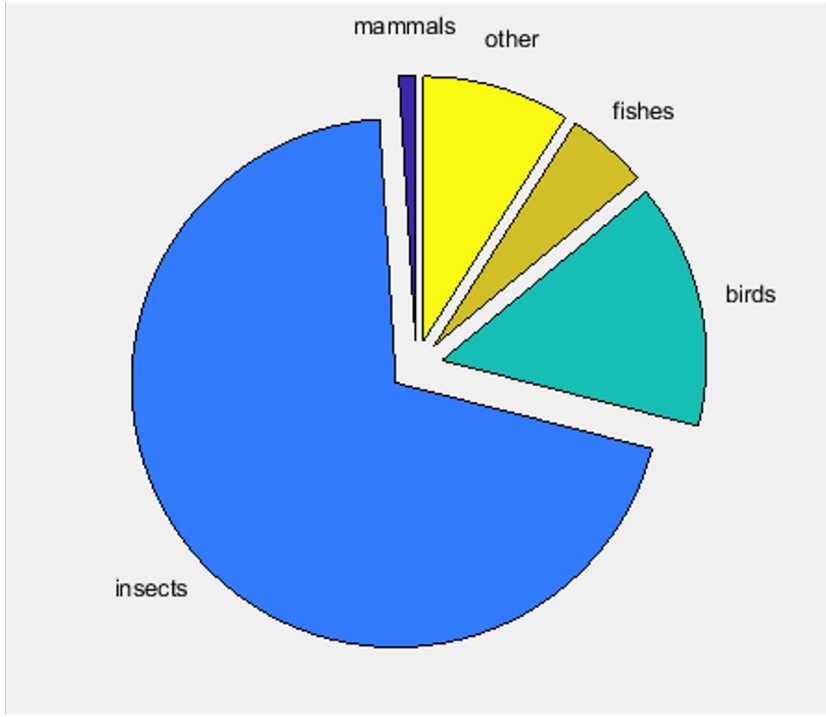
Stem Plot
 	
Function Format
 	
`stem(x,y)`
		`yr=[1988:1994];`
		`sle=[8 12 20 22 18 24 27];`
		`stem(yr,sle)`

Specialized Plots: Pie Chart Plot

- **pie(X)** draws a pie chart using the data in X. Each slice of the pie chart represents an element in X.
- If sum(X) ≤ 1, then the values in X directly specify the areas of the pie slices: **pie** draws only a partial pie if sum(X) < 1 but if sum(X) > 1, then pie normalizes the values by X/sum(X) to determine the area of each slice of the pie.
- You can add a vector of labels for each part of the pie. As many labels as there are values in X.
- You can add a vector to explode some parts of the pie. As many values as there are in X (1=explode, 0=not).

Pie Chart Plot : An Example

```
%percentage of animals in a small park
%mammals, insects, birds, fishes, other
pc = [0.01, 0.70, 0.15, 0.05, 0.09];
labels = {'mammals', 'insects', 'birds', 'fishes', 'other'};
explode = [1 1 1 1 1];
pie (pc, explode, labels);
```



Histograms

. A *histogram* is a plot of the distribution of data. The entire range of data is broken into consecutive sub-ranges or *bins*.

In a histogram plot,

- ✓ *Each bin is represented by a vertical bar.*
- ✓ *The left and right of the vertical bar shows the range of data in the bin.*
- ✓ *The height of the vertical bar is the number of data points in the bin.*

Histograms: The histogram Command

The MATLAB command **histogram** makes a histogram. The simplest form is **histogram(x)** where **x** is a vector of data points.

histogram uses an automatic binning algorithm that returns bins with a uniform width, chosen to cover the range of elements in **x** and reveal the underlying shape of the distribution. The bins are shown as rectangles such that the height of each rectangle indicates the number of elements in the bin.

- *Older books mention a command named **hist**. It is deprecated and its usage is no longer recommended.*

Let’s roll a die 1000 times and place the rolls into a vector.

Histograms : A Simple Example

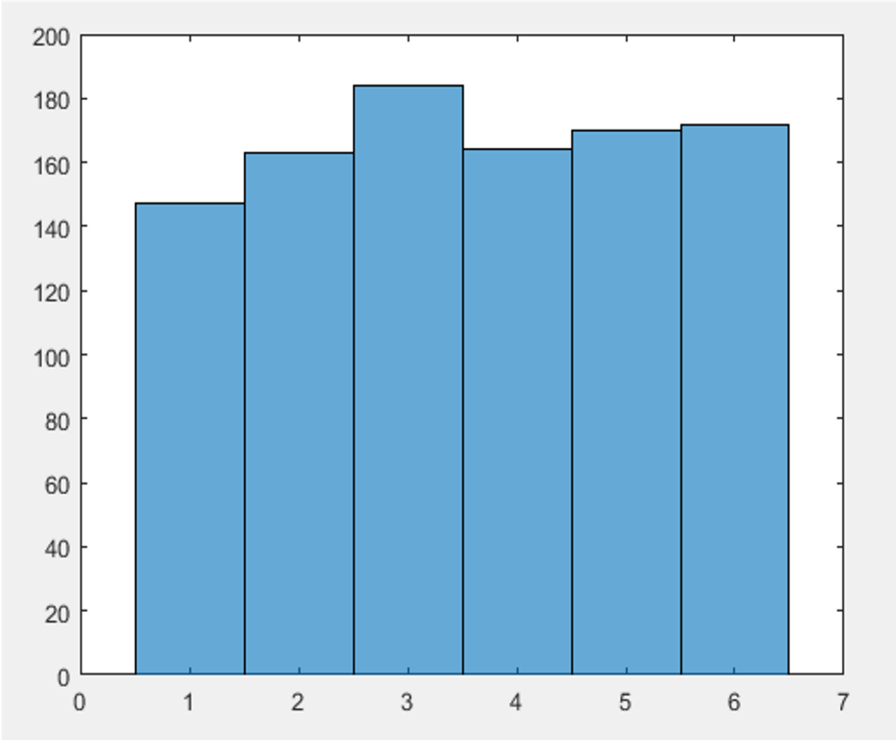
Let's have an example. Let's roll a die 1000 times and place the rolls into a vector.

```
rolls = randi (6, 1, 1000);
```

Array 1 x 1000 (row vector size 1000) of numbers between 1 and 6

Now let's make a simple histogram of these rolls.

```
histogram (rolls)
```



Note: The number of bins is automatically determined by MATLAB. To fix the number of bins yourself, look at the next slide.

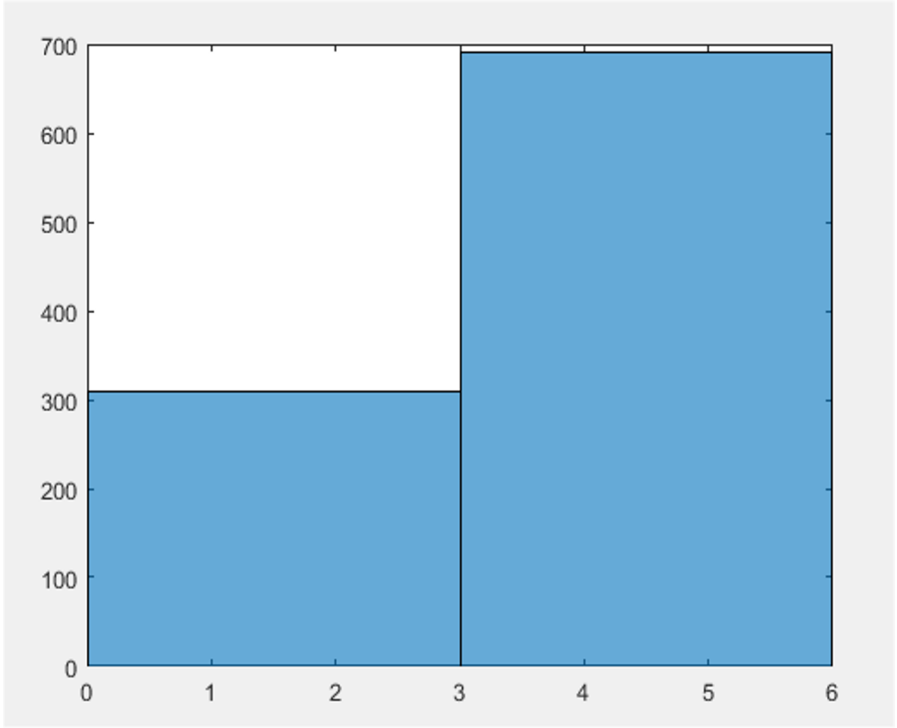
Histograms: Number of Bins

You can also specify the number of bins.

histogram (vector, number_of_bins)

Let's try this with two bins, one for the low numbers (1,2,3) and the other for the high numbers (4,5,6)

```
histogram (rolls, 2)
```



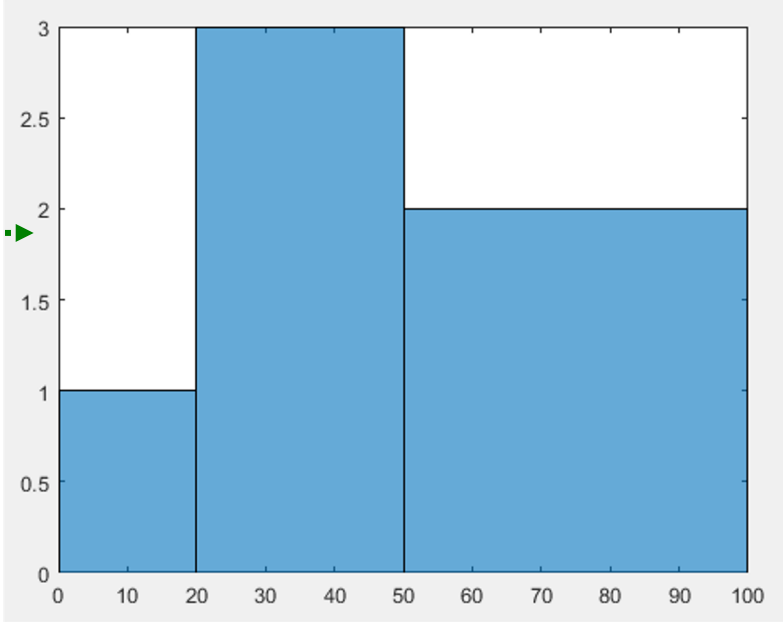
Histograms: Specifying Bin Edges

`histogram(vector, edges)`

- Places the vector into bins with the bin edges specified by the vector named **edges**. Each bin includes the left edge, but does not include the right edge, except for the last bin which includes both edges.

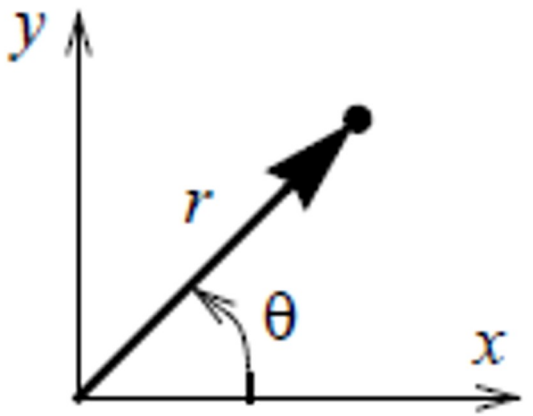
```
v = [20 40 10 30 60 50];
histogram (v, [0 20 50
100])
```

Creates 3 bins: 0 to 20, 20 to 50, and 50 to 100.



Polar Plots

- In *polar coordinates*, points in a plane are specified by (r,θ).
- r is the distance from the origin.
- θ is the angle from the positive, horizontal axis. θ is positive in the counterclockwise direction.



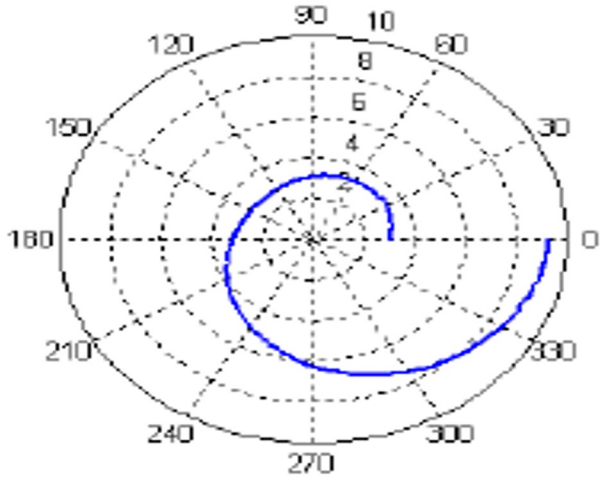
Polar Plots: The polar Command

To make polar a plot in MATLAB, use

```
polar(theta,radius,'line specifiers')
```

- *theta* is expressed in radians.
- Both *theta* and *radius* must be vectors, and **of the same size**.
- The line specifiers are the same as in *plot*.

```
t=linspace(0,2*pi,200);
r=3*cos(0.5*t).^2+t;
polar(t,r)
```



Three-Dimensional Plotting

Three-dimensional (3D) plots are useful for presenting related 3D points like:

- Scalar or vector functions of two independent variables
- Scalar or vector data measurements in three-dimensional space.
- Movement over time in a three-dimensional space.

3D Line Plots

A three-dimensional line plot is a plot obtained by connecting points in 3D space. The MATLAB command for a 3D line plot is **plot3**.

```
plot3(x,y,z,'line specifiers','PropertyName',property value)
```

x, y, and z are vectors of the coordinates of the points.

(Optional) Specifiers that define the type and color of the line and markers.

(Optional) Properties with values that can be used to specify the line width, and marker's size and edge and fill colors.

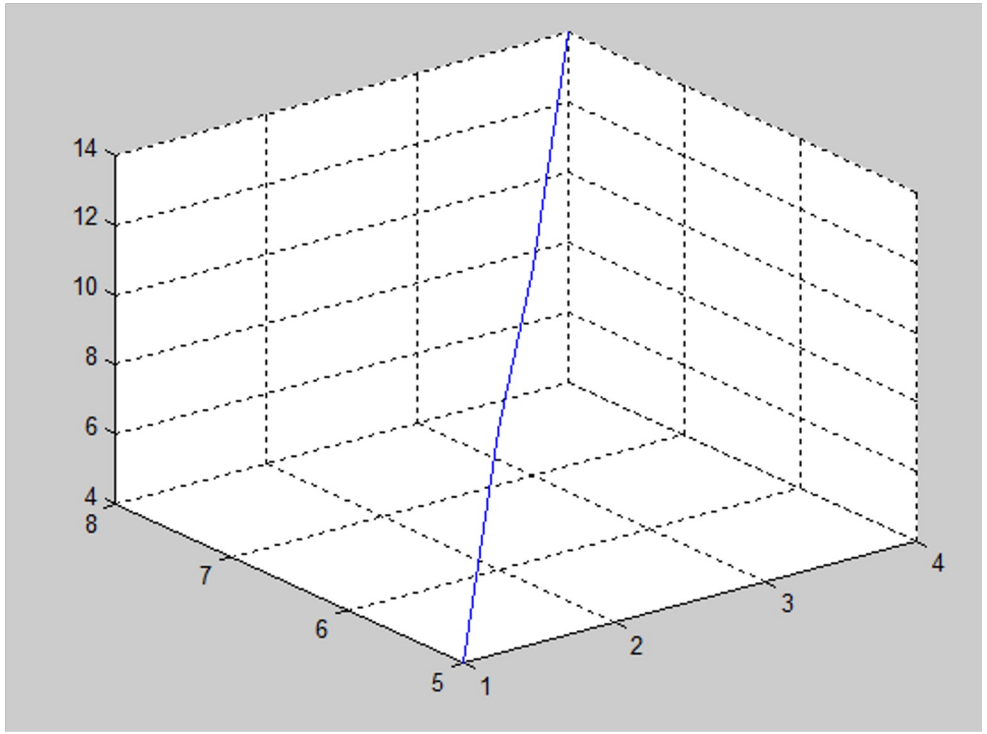
- *x,y, and z must be vectors of the same size.*

- *The line specifiers and the property and values pairs are the same as in 2-D plots.*

3D Line Plots : Example 1

Here is a first example of the **plot3** command. Simple, linear, but rather uninteresting.

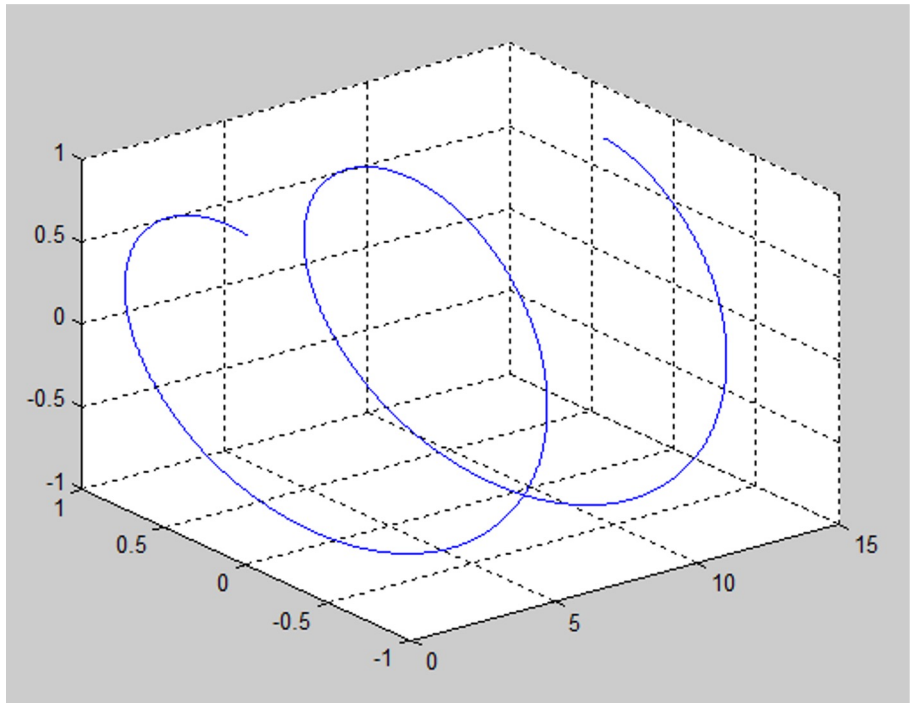
```
x = [1 2 3 4];
y = [5 6 7 8];
z = [4 8 10 14];
plot3 (x,y,z)
grid on
```



3D Line Plots : Example 2

Let's look at a more beautiful plot now.

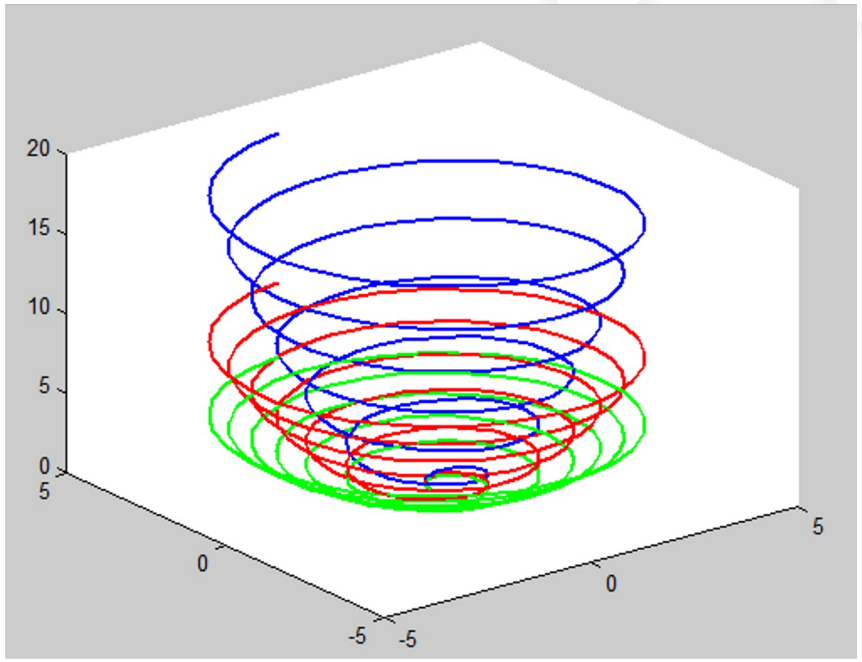
```
a = [0:0.001:4*pi];
b = sin (a);
c = cos (a);
plot3 (a,b,c)
grid on
```



3D Line Plots : Example 3

Now let's look at this much prettier plot!

```
t = 0:0.1:6*pi;
x = sqrt(t) .* sin (2*t);
y = sqrt(t) .* cos (2*t);
z1 = 0.5 * t;
z2 = 0.5 * t;
z1 = t;
z3 = 0.25*t;
plot3(x,y,z1,'b',x,y,z2,'r',x,y,z3,'g','linewidth',2)
grid off
```



3D Mesh and Surface Plots

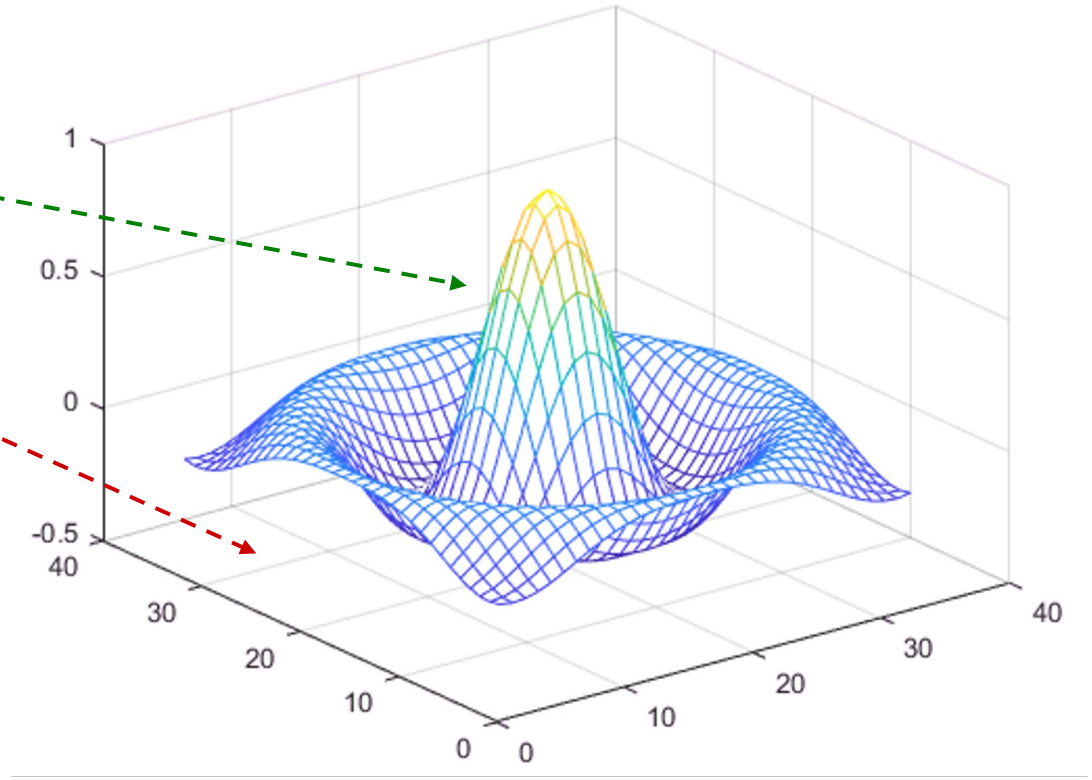
- Mesh and surface plots are 3D plots used to graph functions of the form $z = f(x,y)$.
- x and y are independent variables, z is a dependent variable
- A *mesh plot* connects values of z with lines to form the outline of a surface.
- A *surface plot* connects lines in a mesh plot with planes to show a solid representation of the surface.

3D Mesh and Surface Plots

- You can think of 3D mesh and surface plots like floating graphs hovering above a base plane that is like a two-dimensional grid.

The dependent variable (z) defines the floating graph itself

The two independent (x,y) variables define the two-dimensional base plane



Steps to Make a 3D Mesh Plot

There are three steps to make a mesh (or a surface) plot:

1) Create a grid in the x-y plane that contains the points you're interested in.

2) Calculate the value of z at every point of the grid.

3) Make the 3D plot.

Making 3D Mesh and Surface Plots

Step 1: Creating a grid in the *x-y* plane:

The grid is the set of points on which you want to evaluate *z*. This is the base of the plot. It is like a domain or a range of values.

1a. Determine the domain (range) of the first independent variable (x) and create a row vector.
Example: For a domain of -1 to 3, use `x = [-1:3];`

1b. Determine the domain (range) of the 2nd independent variable (y) and create a row vector.
Example: For a domain of 1 to 4, use `y = [1:4];`

Making 3D Mesh and Surface Plots

1c. We now have two vectors x and y. Let's use now the **meshgrid** command to create a 2-D grid based on the coordinates contained in vectors x and y. X is a matrix where each row is a copy of x, and Y is a matrix where each column is a copy of y. See the result on the next slide.

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

X is the matrix of the x coordinates of the grid points.
Y is the matrix of the y coordinates of the grid points.

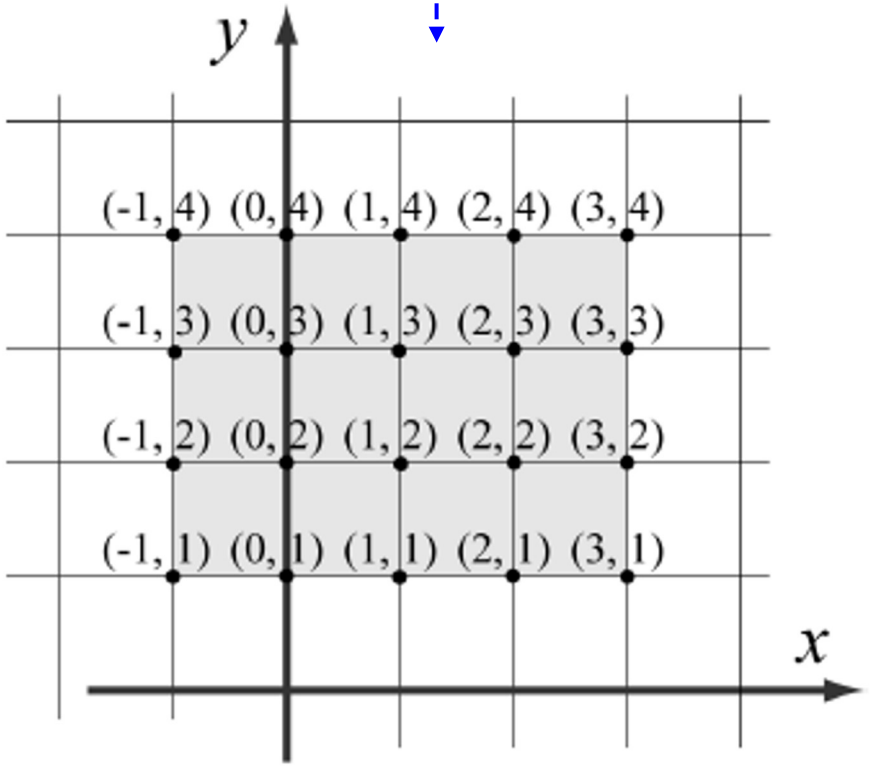
x is a vector that divides the domain of x.
y is a vector that divides the domain of y.

Making 3D Mesh and Surface Plots

Step 1 in MATLAB.
The base plane grid is ready.

```
>> x=-1:3;
>> y=1:4;
>> [X,Y]=meshgrid(x,y)
X =
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
Y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

This is what the plane looks like



Making 3D Mesh and Surface Plots

Step 2: Calculate Z using the function (formula) and the base plane matrix calculated in step 1.

For example let's say we want to display the function $z = xy^2/(x^2+y^2)$

Command

```
Z = X.*Y.^2 ./ (X.^2 + Y.^2)
```

Result

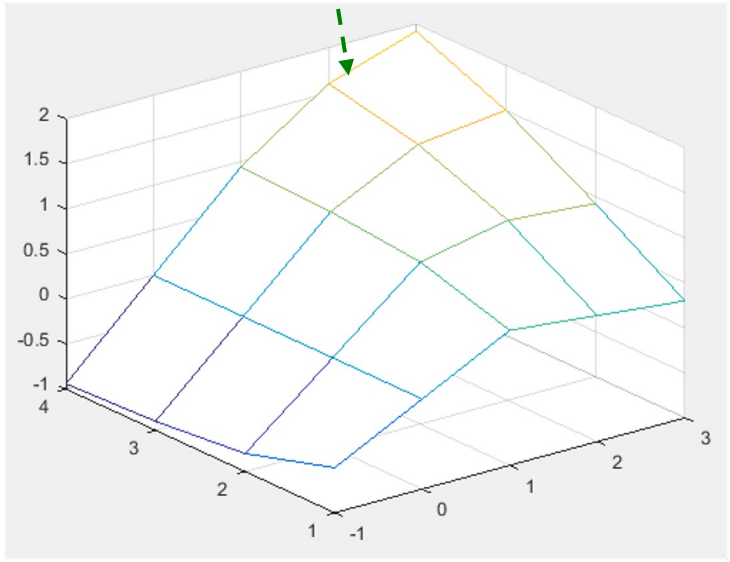
```
Z =
   -0.5000         0    0.5000    0.4000    0.3000
   -0.8000         0    0.8000    1.0000    0.9231
   -0.9000         0    0.9000    1.3846    1.5000
   -0.9412         0    0.9412    1.6000    1.9200
```

Making 3D Mesh and Surface Plots

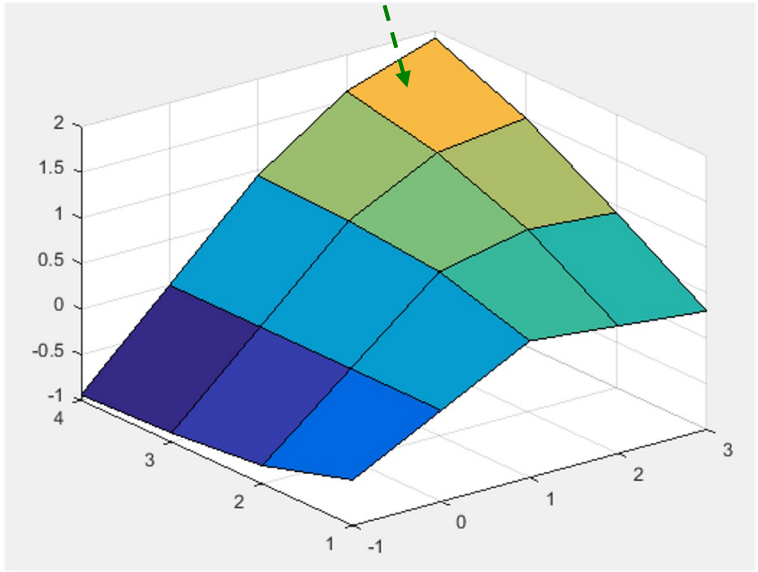
Step 3: Create the 3D plot

This step is very easy. You can create here a mesh plot or a surface plot using the three calculated matrices X, Y, and Z. We use the MATLAB commands **mesh** and **surf**.

mesh (X,Y,Z)



surf (X,Y,Z)

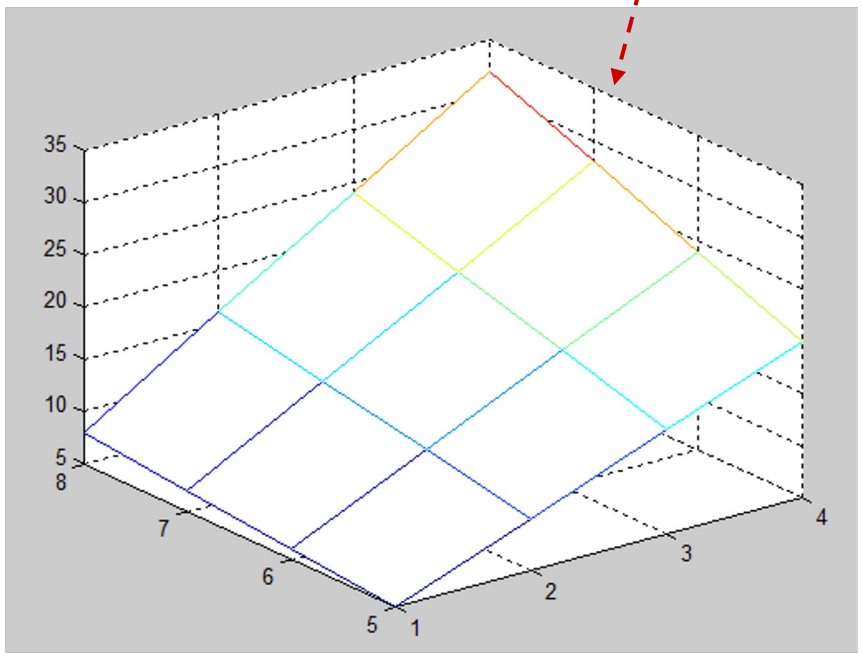


3D Mesh Plot : Another Example

- This is another simple mesh plot. The colors are determined by MATLAB depending on the values of Z.

```
x = [1 2 3 4];
y = [5 6 7 8];
[X,Y] = meshgrid (x,y);
Z = X.*Y;
mesh (X,Y,Z)
```

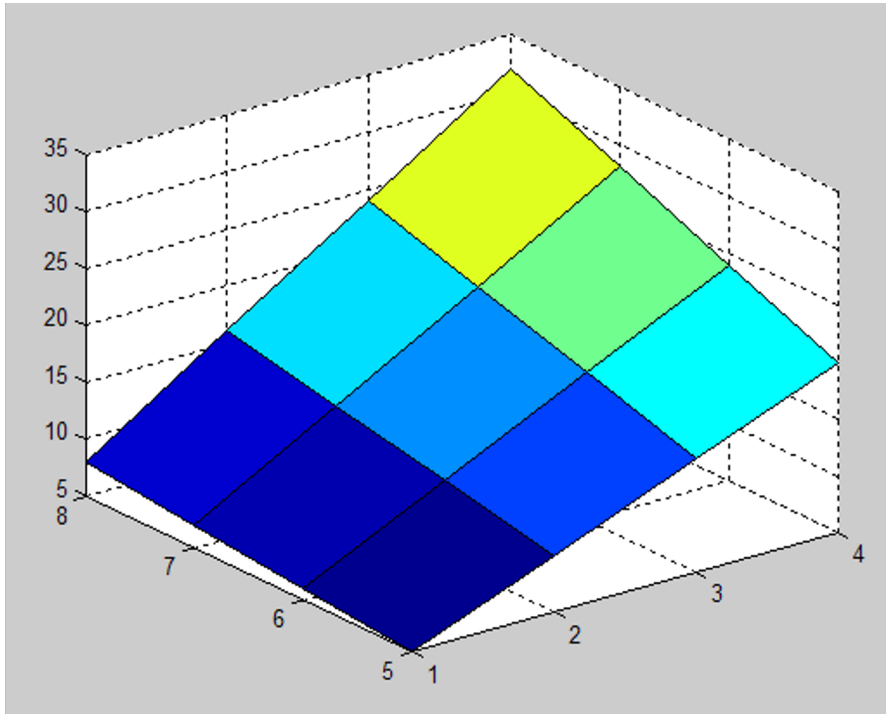
Grid on by default.
grid off to remove



3D Surface Plot : Another Example

- This is the same plot as the previous slide but as a surface plot.

```
x = [1 2 3 4];
y = [5 6 7 8];
[X,Y] = meshgrid (x,y);
Z = X.*Y;
surf (X,Y,Z)
```



3D Mesh and Surface Plots

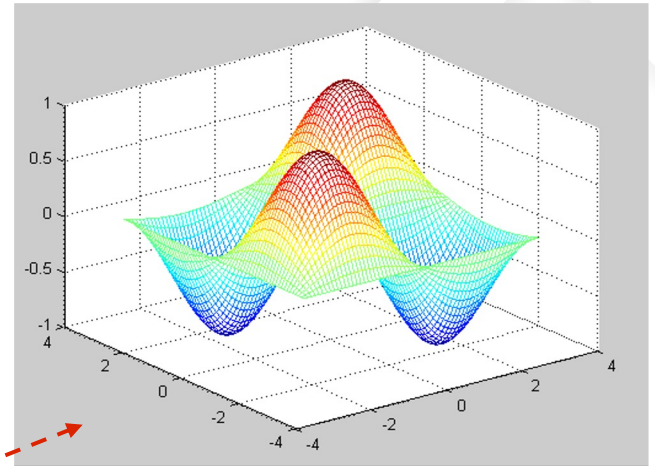
. Let's see more beautiful plots.

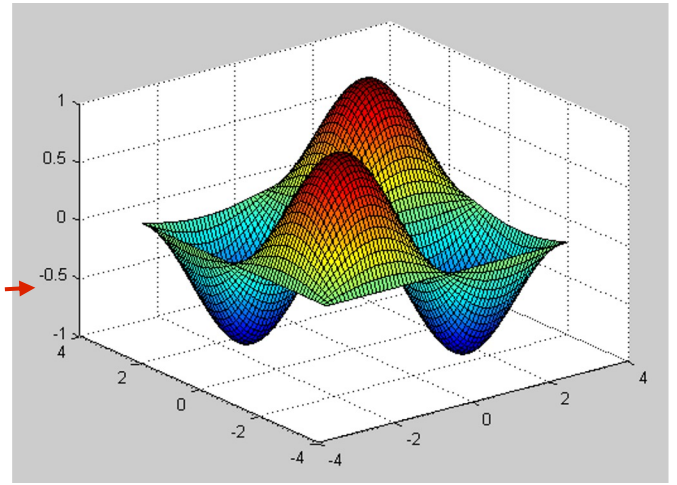
```
c = [-pi:0.1:pi];
d = [-pi:0.1:pi];
[C,D] = meshgrid (c,d);

E = sin (C).* sin (D);

mesh (C,D,E)
surf (C,D,E)
```

Step 1

Step 2

Step 3

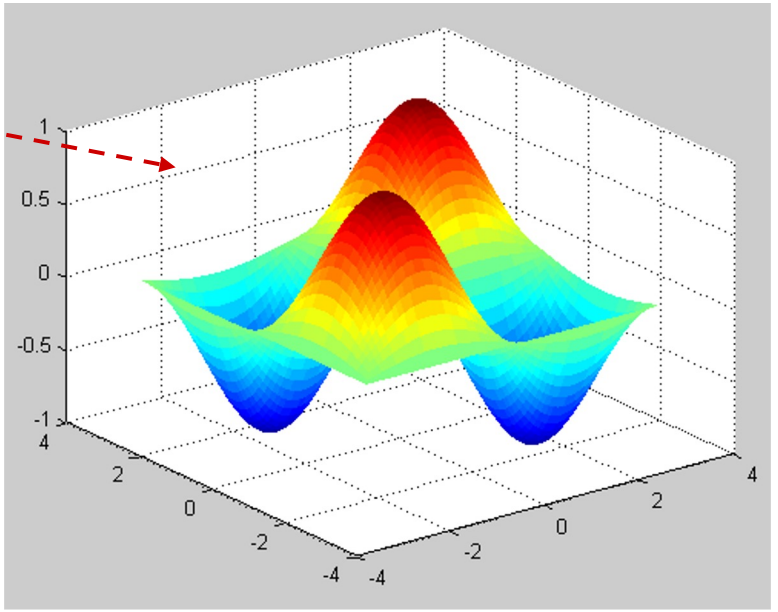


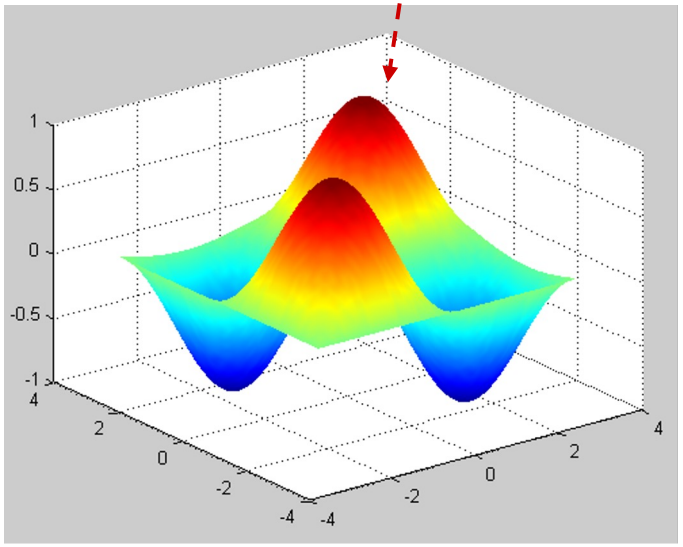


3D Mesh and Surface Plots

You can change the surface shading with the **shading** command (*shading faceted* is the default seen on the previous slide).

```
shading flat
shading interp
```

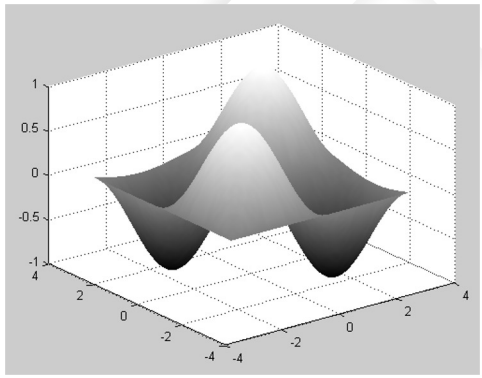


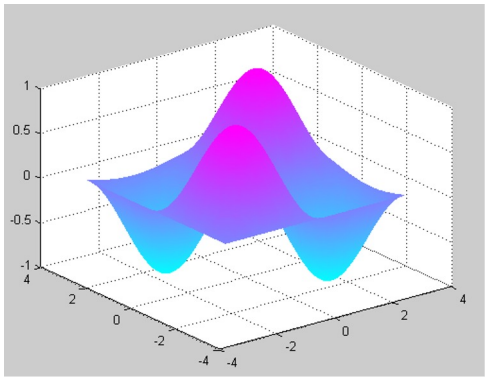


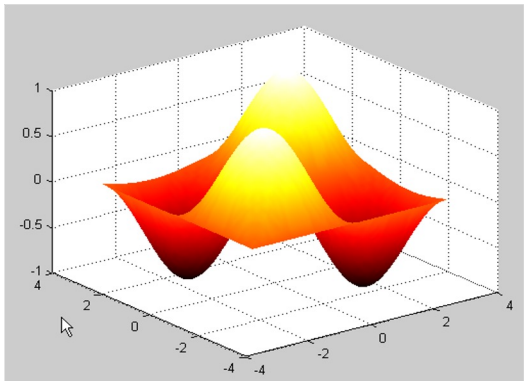
3D Mesh and Surface Plots

You can also change the coloring of the graph with **colormap**.

```
colormap (gray)
colormap (cool)
colormap (hot)
```







Other values for colormap include parula, spring, summer, autumn, winter, bone, copper, lines, colorcube, prism, pink, flag, jet, and hsv. Try them!

3D Polar Plots

To make a 3D polar plot of a function like $z = f(r,\theta)$, follow these steps:

1) Make a grid of values of θ and r with the **meshgrid** command. It creates a polar grid.
2) Compute the value of z at each grid point.
3) Convert the polar grid to a cartesian grid using the **pol2cart** command.
4) Make the 3D plot using values of z and the new cartesian grid.

3D Polar Plots

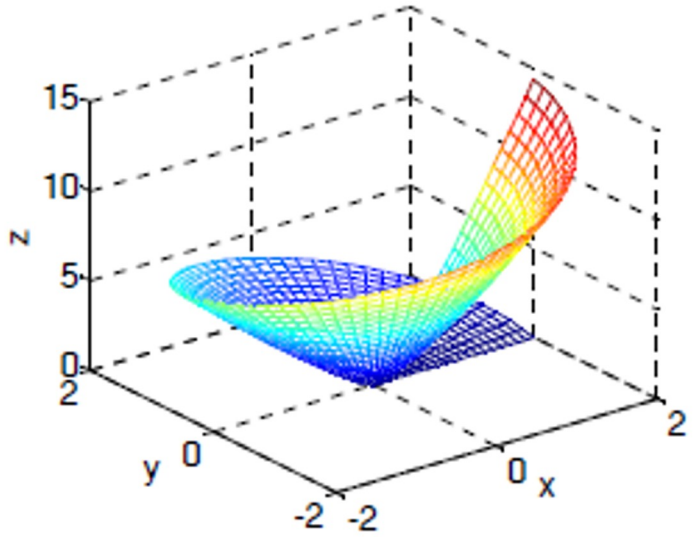
For example the following script creates a plot of the function $z = r\theta$ over the domain $0 \le \theta \le 360°$ and $0 \le r \le 2$.

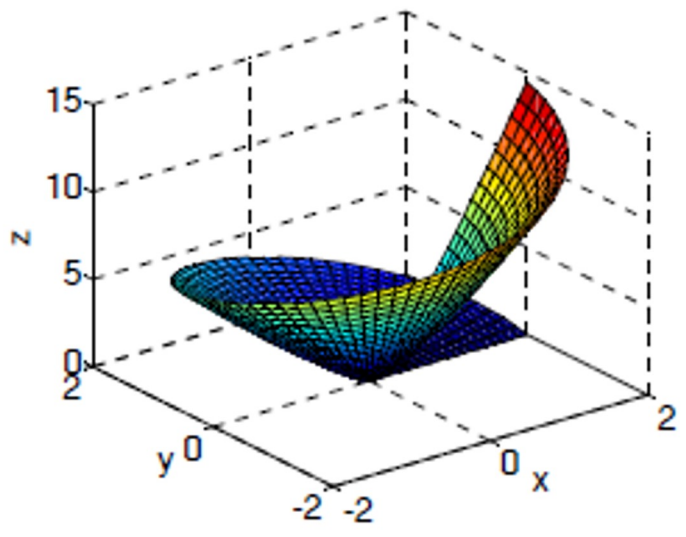
```
[th,r]=meshgrid((0:5:360)*pi/180,0:.1:2);
Z=r.*th;
[X,Y] = pol2cart(th,r);
mesh(X,Y,Z)
```

1. Make base grid (polar).
2. Calculate Z at each point.
3. Convert base grid into cartesian.
4. Plot.

Type `surf(X,Y,Z)` for surface plot.

The figures created by the program are:

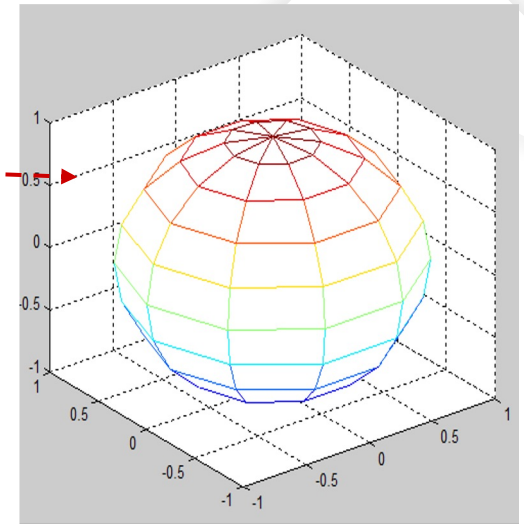




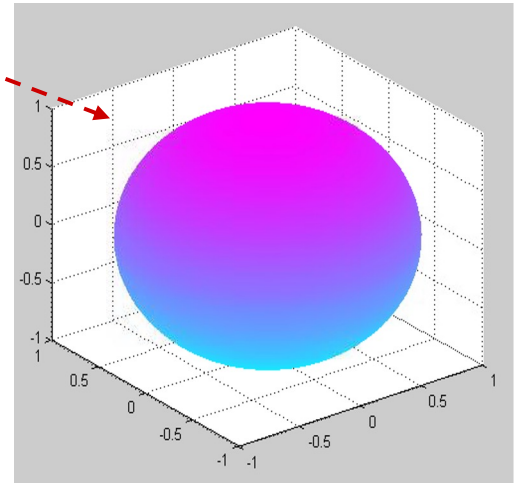
3D Plots for Special Graphs : Sphere

It is extremely easy to plot a sphere. The sphere function returns the coordinates of a sphere with a specified number of faces (20 if the number is not specified).

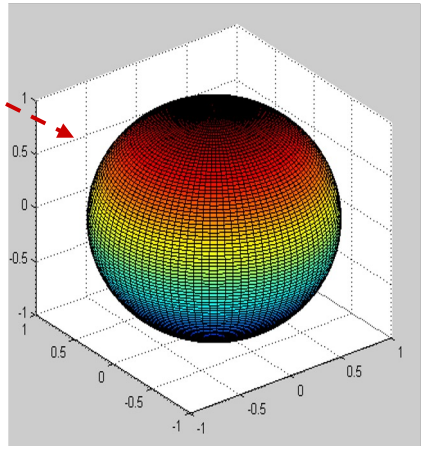
```
[X,Y,Z] = sphere (10);
mesh (X,Y,Z)
```



```
[X,Y,Z] = sphere (50);
surf (X,Y,Z)
shading interp
colormap cool
```



```
[X,Y,Z] = sphere (100);
surf (X,Y,Z)
```



The view Command

The **view** command controls the direction from which you view your plot. The command is **view(az,el)** or **view([az el])**

- **az** (***azimuth***): the angle (in degrees) in the *x y* plane measured from the negative *y* axis and positive is in the counterclockwise direction.
- **el** (***elevation***): the angle of elevation (in degrees) from the *x y* plane. Positive is the direction of the positive *z* axis.

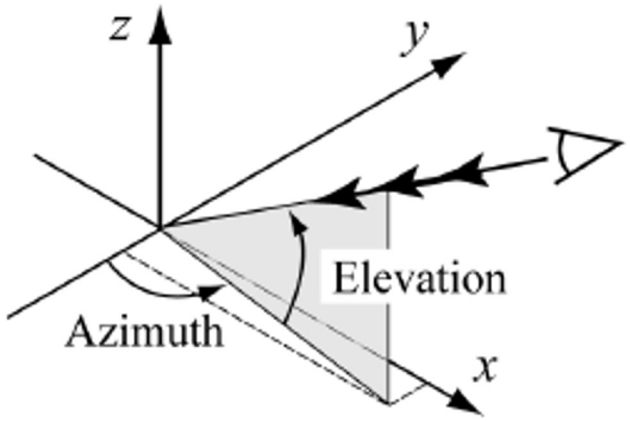
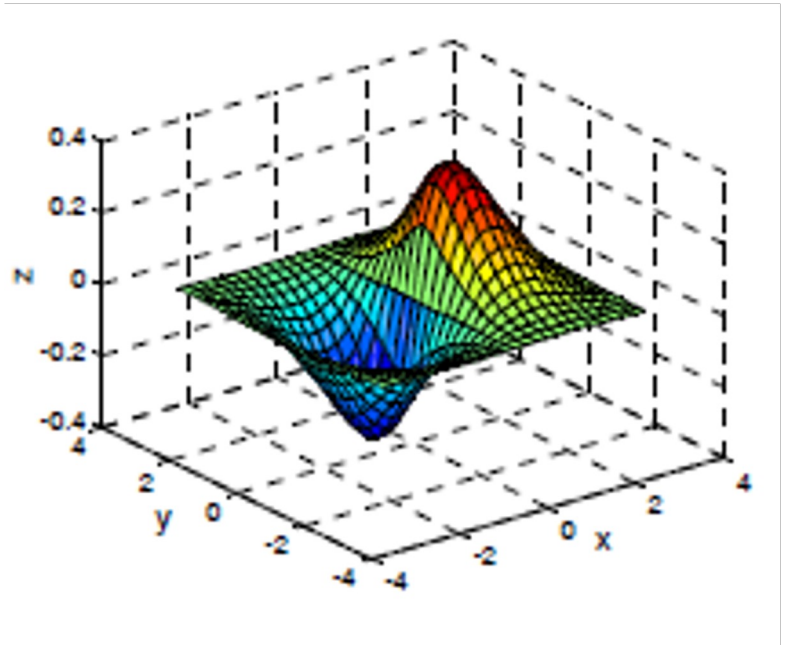


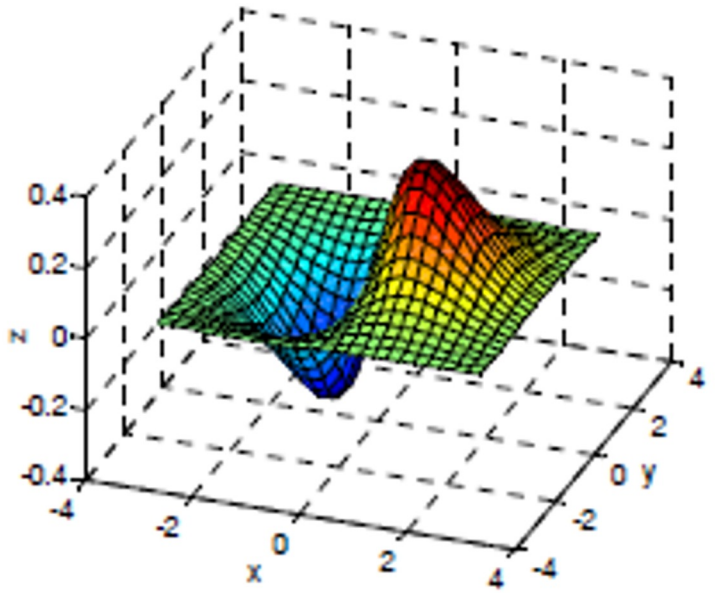
Figure 10-3: Azimuth and Elevation angles.

The view Command : Examples

Default view angles are *az* = -37.5° and *el* = 30°



az = -37.5° and *el* = 30°



az = 20° and *el* = 35°

The view Command : Projection

You can project a 3D curve onto a 2D plane by specific settings, as you would expect, of azimuth and elevation (a value of 0 in one direction projects or "flattens" the 3D plot onto the plane at 0 for that direction. Here a few typical values:

Projection plane	*az* value	*el* value
x-*y* (top view)	0	90
x-*z* (side view)	0	0
y-*z* (side view)	90	0

Using Plots within the Coding Area with Live Script

- With regular scripts (.m), to include a plot, all is needed is to provide the code and the output will appear. That is the same with live scripts (.mlx).
- With live scripts, the figure can be saved beforehand and added the the report with the openfig command. This is similar to the technique used to add graphics seen in the scripting lesson.

Using Plots within the Coding Area with Live Script

- Here is a quick example. In the command window or a regular script:

```
x = linspace (0, 2*pi, 30);
y = sin (x);
z = cos (x);
plot (x, y, x, z);
savefig ('SinCos.fig');
```

- The plot is saved to a file named **SinCos.fig**.
- To use the figure, simply add `openfig ('SinCos.fig');` in the live script.

Using Plots within the Coding Area with Live Script

- Here is a quick example:

```
x = linspace (0, 2*pi, 30);
y = sin (x);
z = cos (x);
plot (x, y, x, z);
```

- Save the plot to a file named **SinCos.fig**.
- To use the figure, simply add `openfig ('SinCos.fig');` in the live script.

This concludes our overview of MATLAB and a taste of things to come!

End of lesson