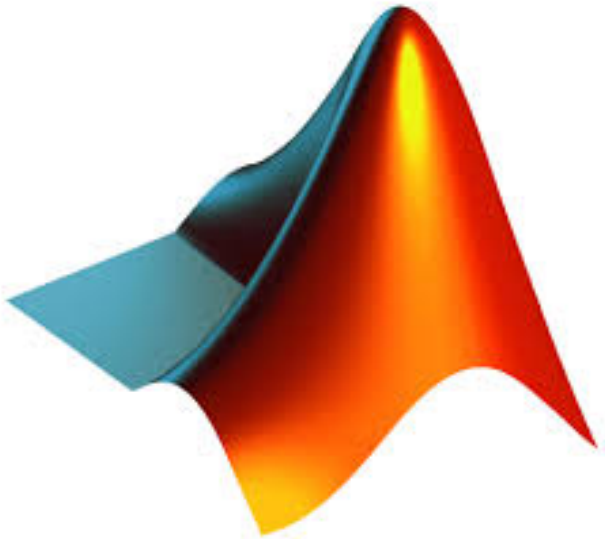
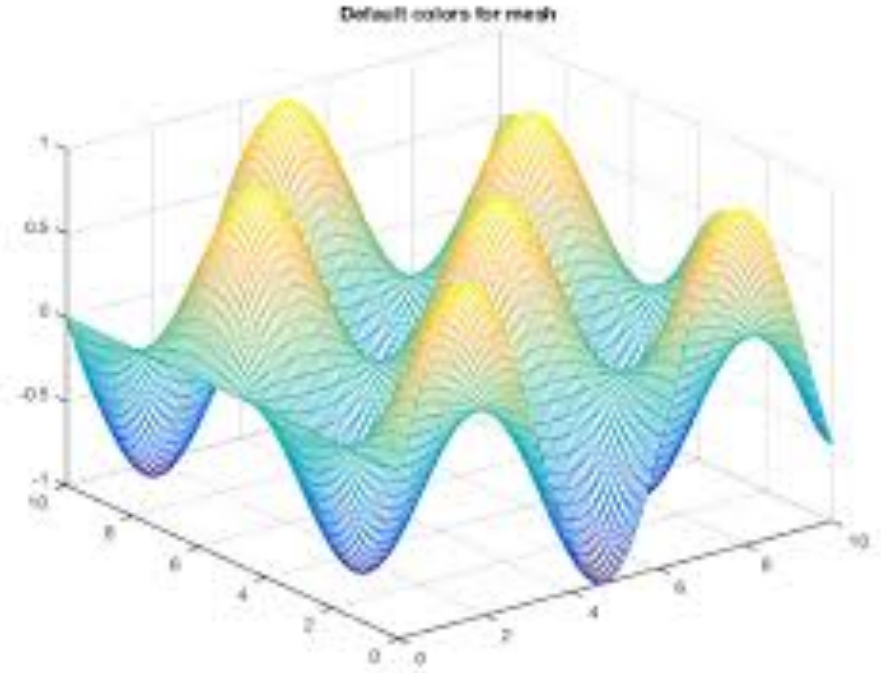
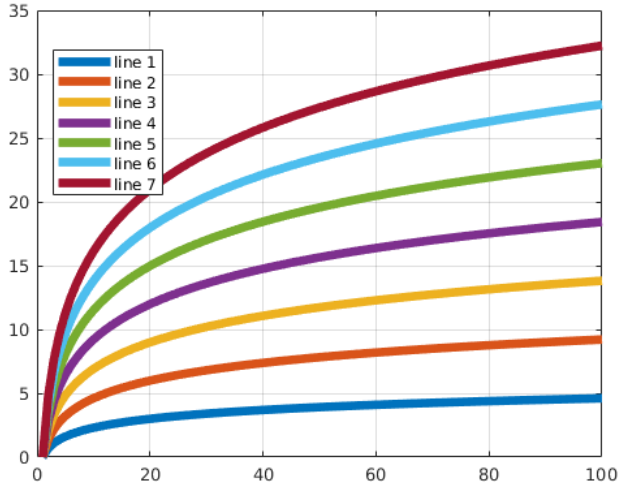
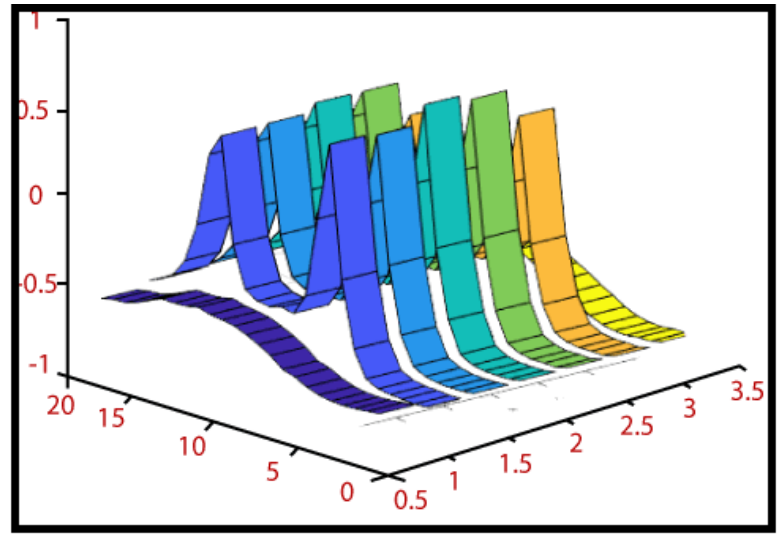
MATLAB Graphics









MATLAB includes good tools for visualization. For example,

- ✓ Basic 2-D plots,
- ✓ fancy 3-D graphics with lighting and colorrnaps,
- ✓ *Handle Gmphics* tools for designing sophisticated graphics user interfaces (user control of the graphics)
- ✓ an- animation

Why to use MATLAB for graphics?

- ➢ Matlab graphics is easy to use and expandable
- ➢ Commands are easy, simple and intuitive
- ➢ If you are not satisfied with what you get, you can control and manipulate virtually everything in the graphics win-dow.

Lesson 20: Two dimensional graphics

Learning objectives

To learn how to generate two dimensional plots in MATLAB.

✓ Basic 2-D plots,

Basic 2D graphics command → `plot(xvalues, yvalues,'style-option')`

x-coordinate

y-coordinate

optional argument that specifies the color, the line style (e.g., solid, dashed, dotted), and the point-marker style (e.g., o, +, *).

NOTE

- All three style options can be specified together.
- xvalues and yvalues MUST have the same length.
- Unequal length of the two vectors is the most common source of error in the plot commamd.
- The plot function also works with a single-vector argument, in which case the elements of the vector are plotted against row or column indices.

```
plot (x , y)          plot y vs x with a solid line (the default line style.)
plot(x,y, '--’)       plot y vs x with a dashed line
plot (x)              plots the elements of x against their row index.
```

You indicate the line styles, markers, and colors you want to display,

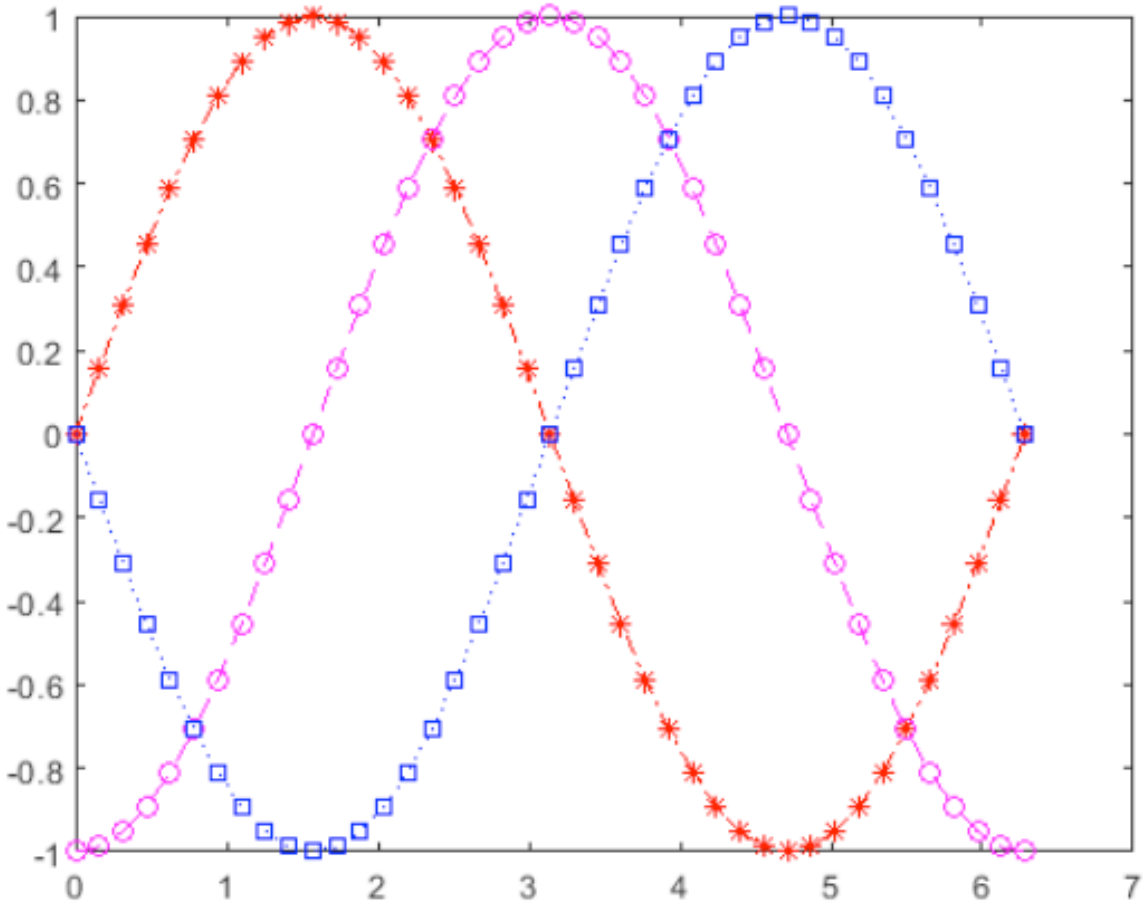
Line Style	Description
-	Solid line
--	Dashed line
:	Dotted line
-.	Dash-dot line

Color	Description
y	yellow
m	magenta
c	cyan
r	red
g	green
b	blue
w	white
k	black

Marker	Description
'o'	Circle
'+'	Plus sign
'*'	Asterisk
'.'	Point
'x'	Cross
'_'	Horizontal line
'|'	Vertical line
's'	Square
'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'p'	Pentagram
'h'	Hexagram

Example: Modify Line Appearance

```
t = 0:pi/20:2*pi;
plot(t,sin(t),'-.r*')
hold on
plot(t,sin(t-pi/2),'--mo')
plot(t,sin(t-pi),':bs')
hold off
```



Note that, when no style-option is specified, MATLAB uses a blue solid line by default.

Labels, title, legend, and other text objects

Plots may be annotated with xlabel, ylabel, title, and text commands.

xlabel (' Pipe Length ‘)	labels the x-axis with Pipe Length,	take string arguments
ylabel (' Fluid Pressure ‘)	labels the y-axis with Fluid Pressure,	
title('Pressure Variation’)	titles the plot with Pressure Variation,	

text(x-coordinate, y-coordinate, 'text'), where the coordinate val-ues are taken from the current plot.

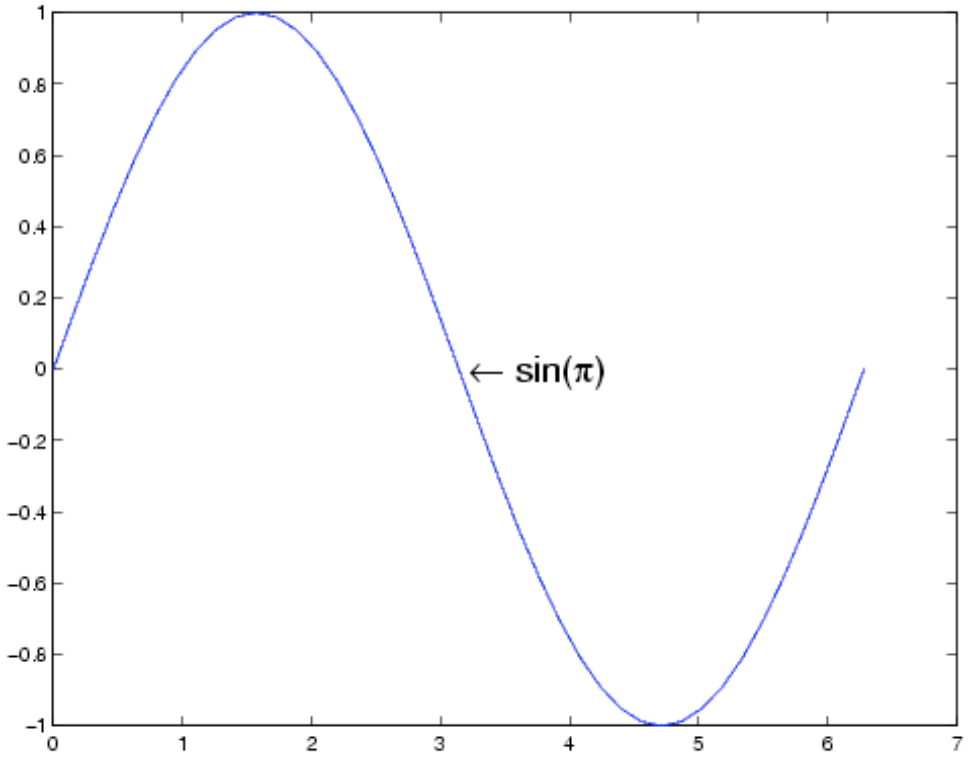
text(2,6,'Note this dip’) → writes"Note this dip" atthelocation (2.0,6.0) in the plot coordinates.

- The arguments of text(x,y, 'text') command may be vectors, in which case x and y must have the same length and text may be just one string or a vector of string.

- If text is vector then it must have the same length as x.

- A useful variant of the text command is gtext , which only takes a string argument (a single string or a vector of strings) and lets the user specify the location of the text by clicking the mouse at the desired location in the graphics window .

```
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
```



Annotate the point (pi,0) with the string

```
text(pi,0,' \leftarrow sin(\pi)','FontSize',18)
```

Property name

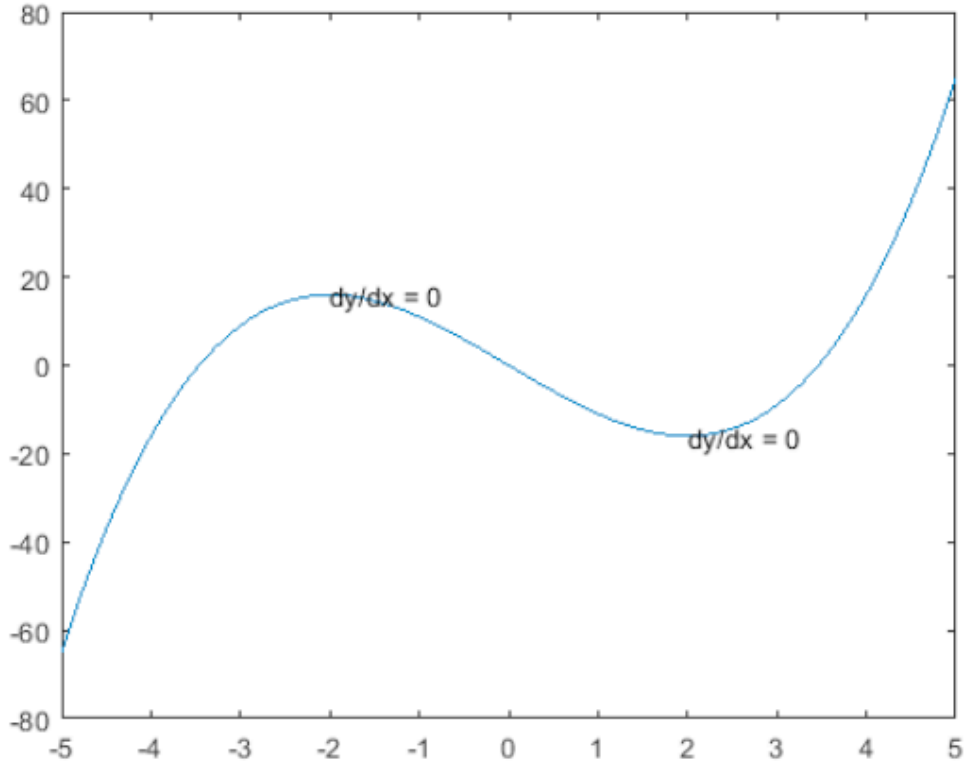
Value

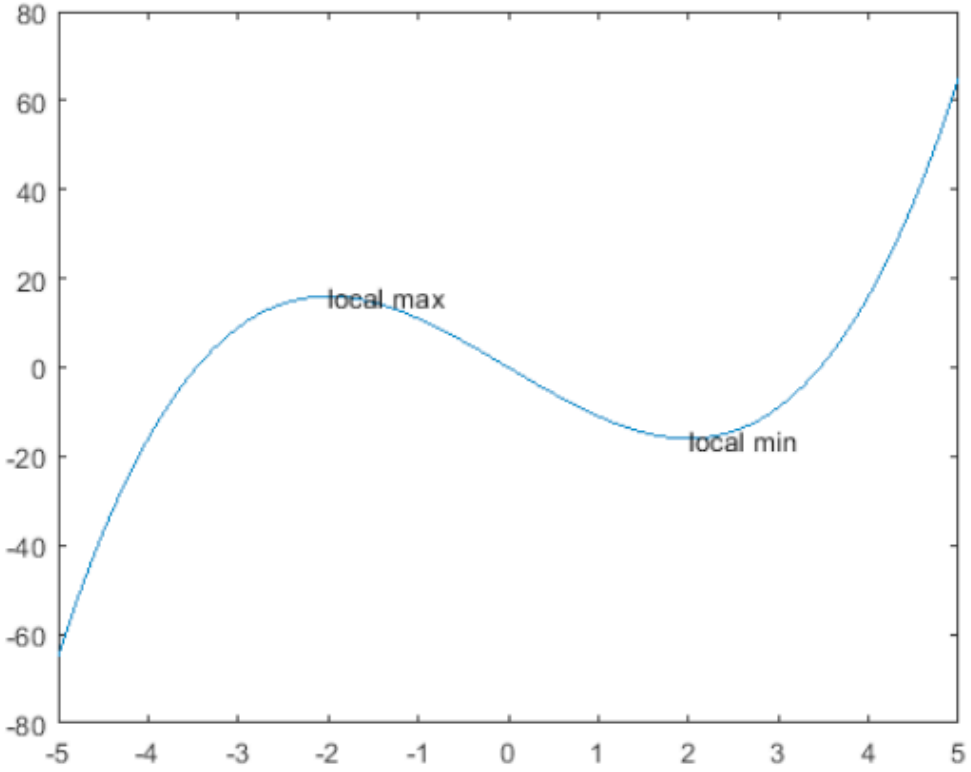
The statement,

```
text(x,y,'\it e^{i\omega\tau} = cos(\omega\tau) + i sin(\omega\tau)')
```

uses embedded TeX sequences to produce $e^{i\omega\tau} = cos(\omega\tau) + i\, sin(\omega\tau)$

```
x = linspace(-5,5);
y = x.^3-12*x;
plot(x,y)
xt = [-2 2];
yt = [16 -16];
str = 'dy/dx = 0';
text(xt,yt,str)
```






```
x = linspace(-5,5);
y = x.^3-12*x;
plot(x,y)
xt = [-2 2];
yt = [16 -16];
str = {'local max','local min'};
text(xt,yt,str)
```

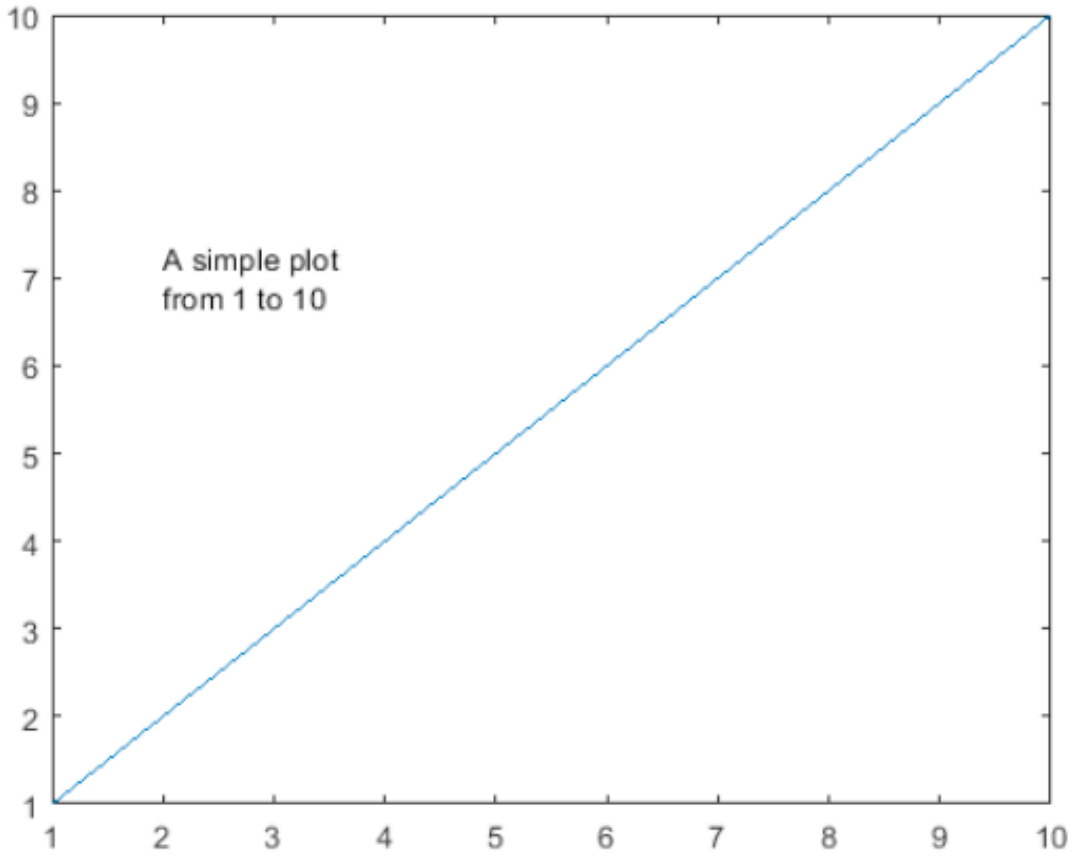
Display Multiline Text

Create a line plot and add one text description to the axes. Display multiline text by specifying str as a cell array.

```
>> plot(1:10)
>> str = {'A simple plot','from 1 to 10'};
>> text(2,7,str)
```

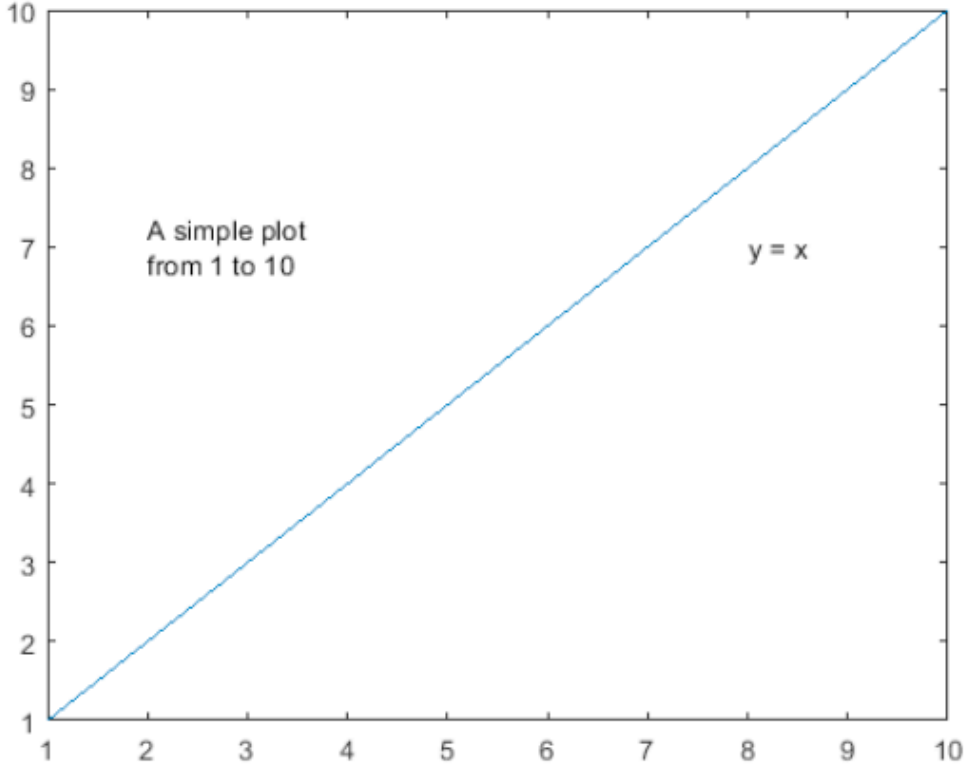
text (x_pos, y_pos, str).

```
>> whos
Name        Size                Bytes  Class
Attributes

  str       1x2                   274  cell
```



Create a line plot and add two text descriptions to the axes. When adding multiple text descriptions to the axes, display multiline text by specifying nested cell arrays.

```
>> plot(1:10)
>> str = {{'A simple plot','from 1 to 10'},'y = x'};
>> text([2 8],[7 7],str)
>> whos
  Name        Size                Bytes  Class     Attributes

  str         1x2                   508  cell
```



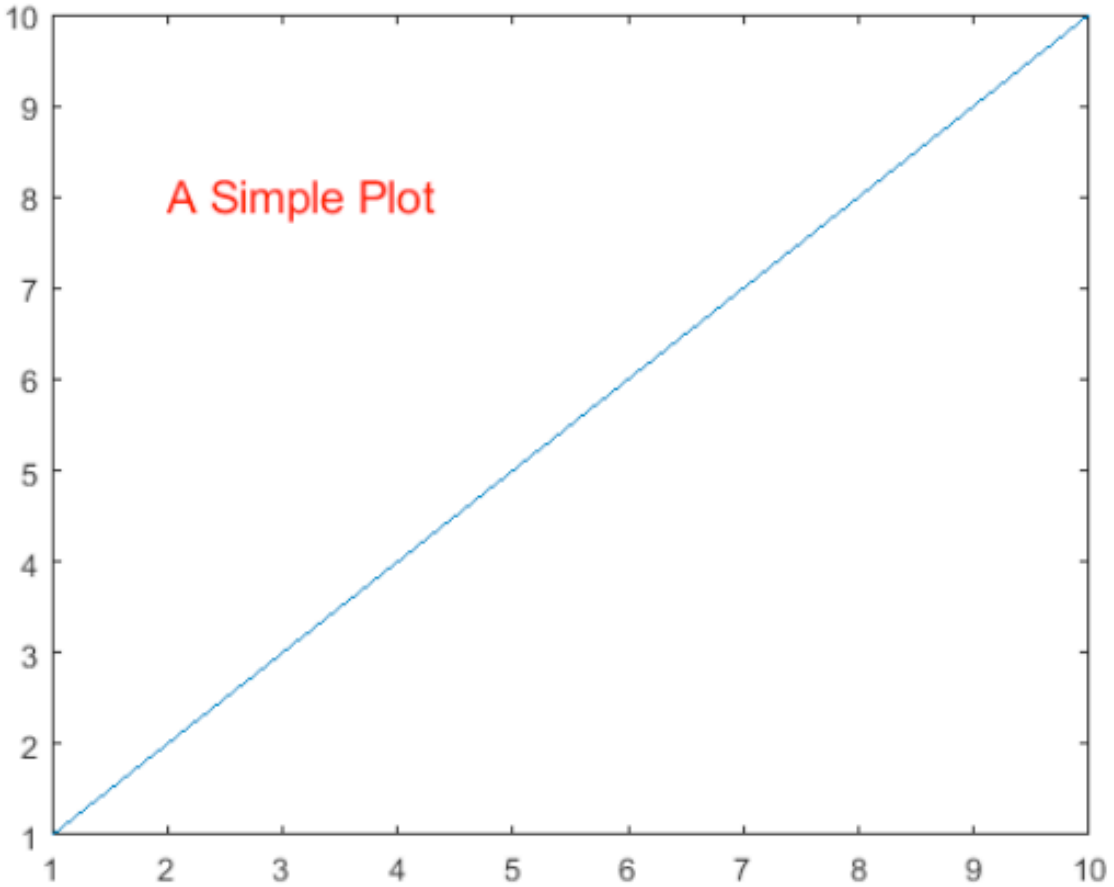
Specify Text Size and Color

Create a line plot and add a text description to the axes. Use red, size 14 font.

Property name

```
>> plot(1:10)
text(2,8,'A Simple Plot','Color','red','FontSize',14)
```

Value



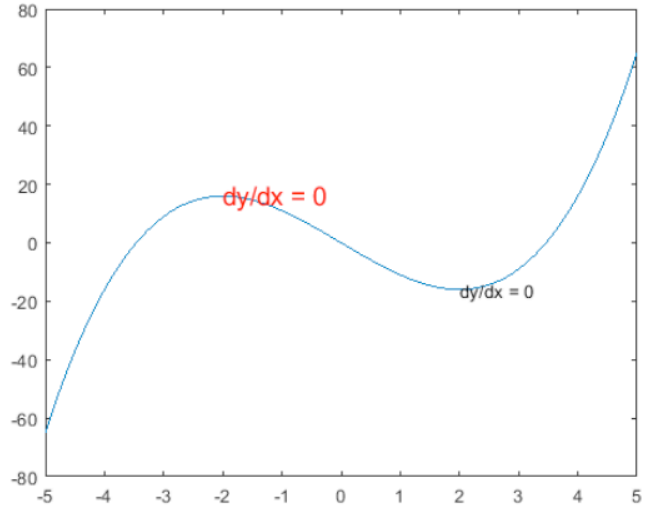
Modify Existing Text

Create a line plot and add two text descriptions along the line. Return the text objects, t.

```
>> x = linspace(-5,5);
y = x.^3-12*x;
plot(x,y)
t = text([-2 2],[16 -16],'dy/dx = 0')

t =

  2×1 Text array:

  Text     (dy/dx = 0)
  Text     (dy/dx = 0)
```

- text function creates one text object for each text description.
- Change the color and font size for the first text object using t(1).
- Use dot notation to set properties.

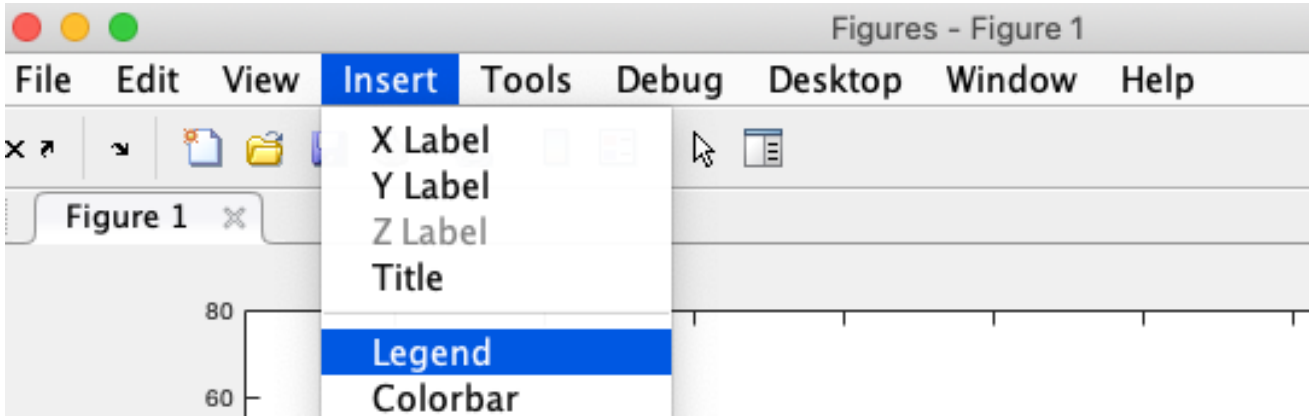
```
>>t(1).Color = 'red';
>>t(1).FontSize = 14;
```




```
Name      Size              Bytes  Class                                Attributes

  t         2x1                16  matlab.graphics.primitive.Text
  x         1x100             800  double
  y         1x100             800  double
```

Legend

- Legends on plots can be produced using the Insert → legend button in the figure window toolbar



or with the legend command.

- The legend command is quite versatile. It can take several optional arguments.

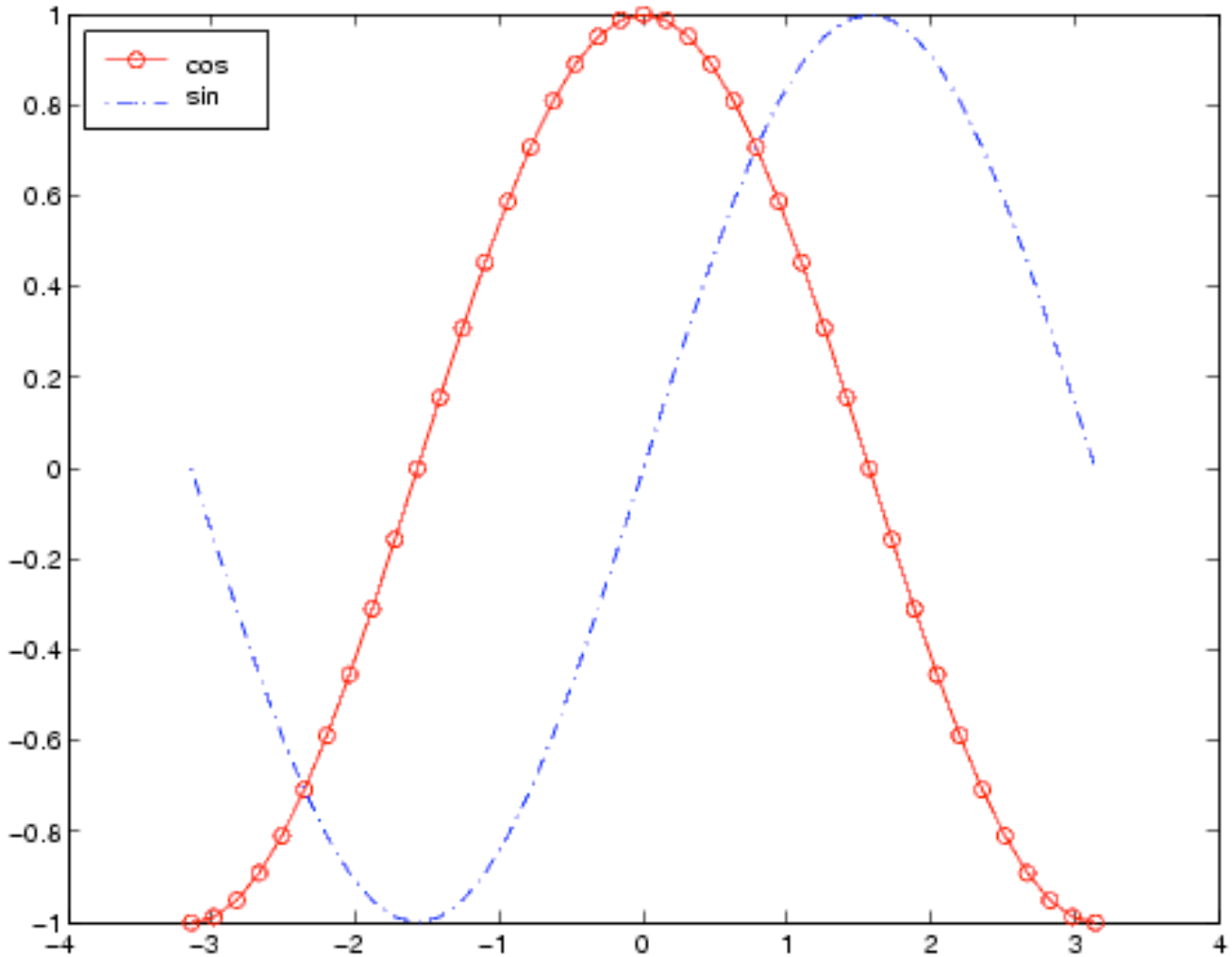
Syntax

```
legend('string1','string2',...)
legend(h,'string1','string2',...)
legend(string_matrix)
legend(h,string_matrix)
legend(axes_handle,...)
legend('off')
legend('hide')
legend('show')
legend('boxoff')
legend('boxon')
legend(h,...)
legend(...,pos)
h = legend(...)
[legend_h,object_h,plot_h,text_strings] = legend(...)
```

- `pos = -1` places the legend outside the axes boundary on the right side.
- `pos = 0` places the legend inside the axes boundary, obscuring as few points as possible.
- `pos = 1` places the legend in the upper-right corner of the axes (default).
- `pos = 2` places the legend in the upper-left corner of the axes.
- `pos = 3` places the legend in the lower-left corner of the axes.
- `pos = 4` places the legend in the lower-right corner of the axes.

Add a legend to a graph showing a sine and cosine function:

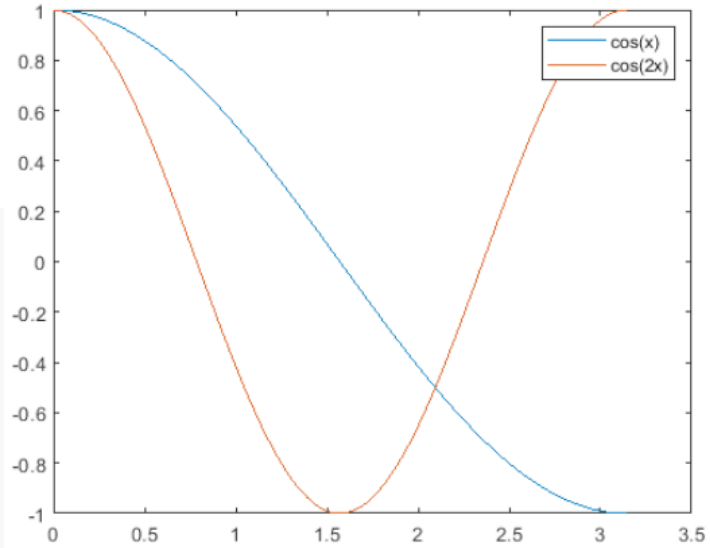
```
x = -pi:pi/20:pi;
plot(x,cos(x),'-ro',x,sin(x),'-.b')
h = legend('cos','sin',2);
```

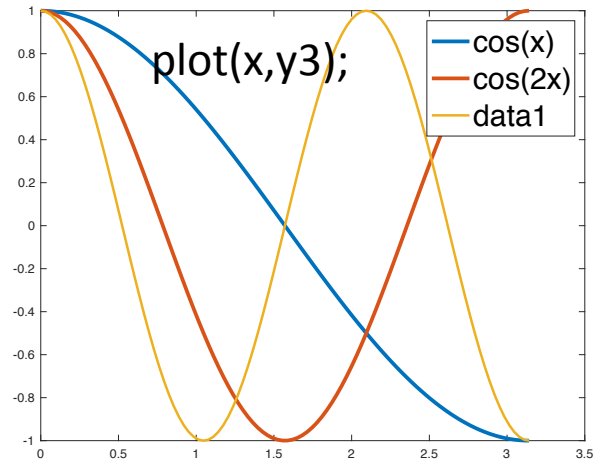


Examples

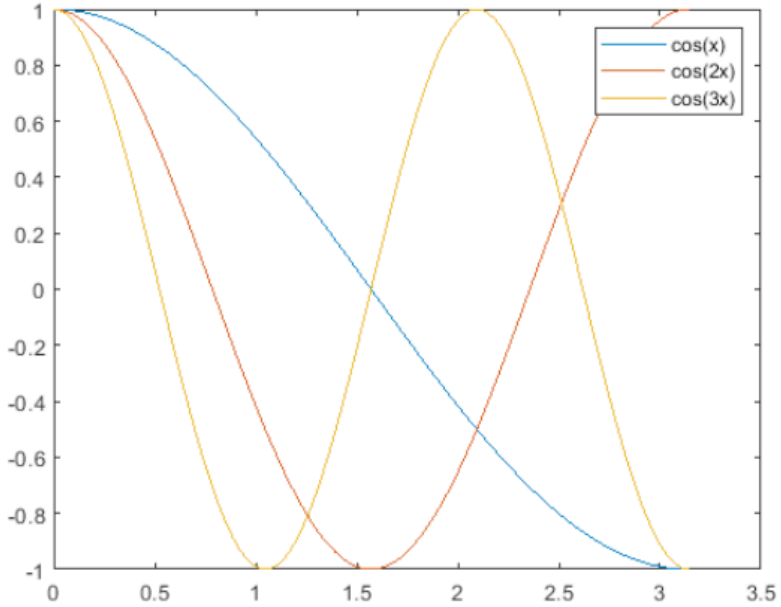
Add Legend to Current Axes

```
x = linspace(0,pi);
y1 = cos(x);
plot(x,y1)

hold on
y2 = cos(2*x);
plot(x,y2)

legend('cos(x)','cos(2x)')
```





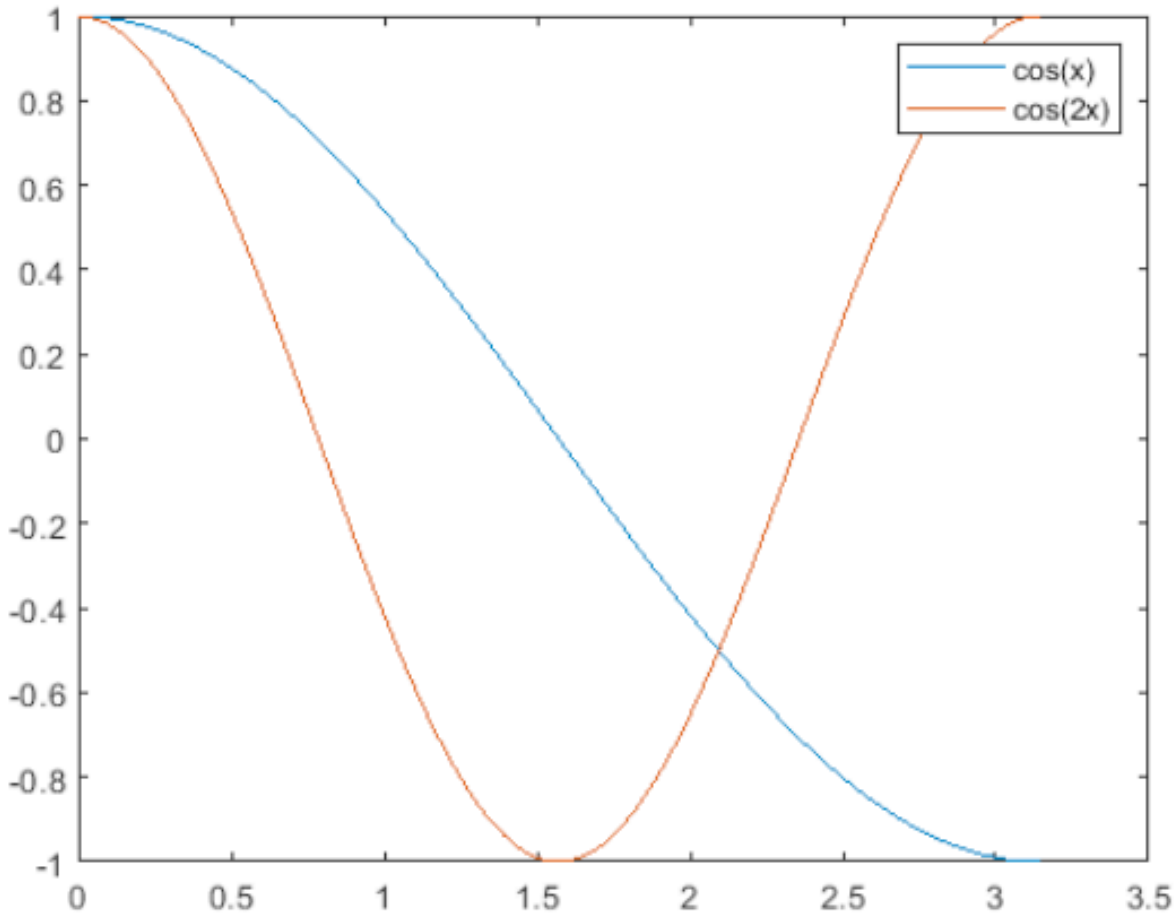
y3 = cos(3*x);
plot(x,y3,'DisplayName','cos(3x)') hold off



- If you add or delete a data series from the axes, the legend updates accordingly.
- Control the label for the new data series by setting the DisplayName property as a name-value pair during creation.
- If you do not specify a label, then the legend uses a label of the form 'dataN'.

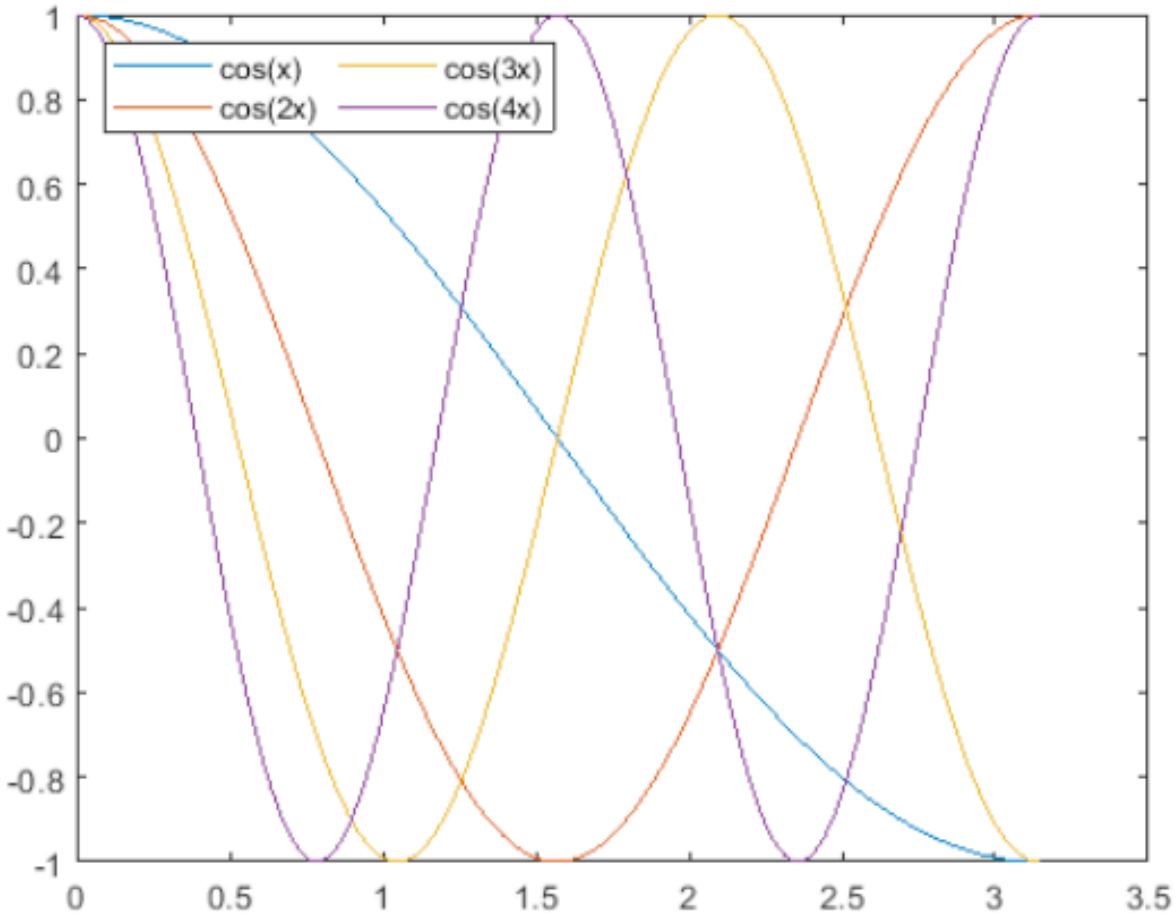
Specify Legend Labels During Plotting Commands

```
>> x = linspace(0,pi);
y1 = cos(x);
plot(x,y1,'DisplayName','cos(x)')

hold on
y2 = cos(2*x);
plot(x,y2,'DisplayName','cos(2x)')
hold off
>> legend
```



Legend Location and Number of Columns

```
>> x = linspace(0,pi);
y1 = cos(x);
plot(x,y1)

hold on
y2 = cos(2*x);
plot(x,y2)

y3 = cos(3*x);
plot(x,y3)

y4 = cos(4*x);
plot(x,y4)
hold off
>> legend({'cos(x)','cos(2x)','cos(3x)','cos(4x)'},'Location','northwest','NumColumns',2);
or
>> legend('cos(x)','cos(2x)','cos(3x)','cos(4x)','Location','northwest','NumColumns',2)
```

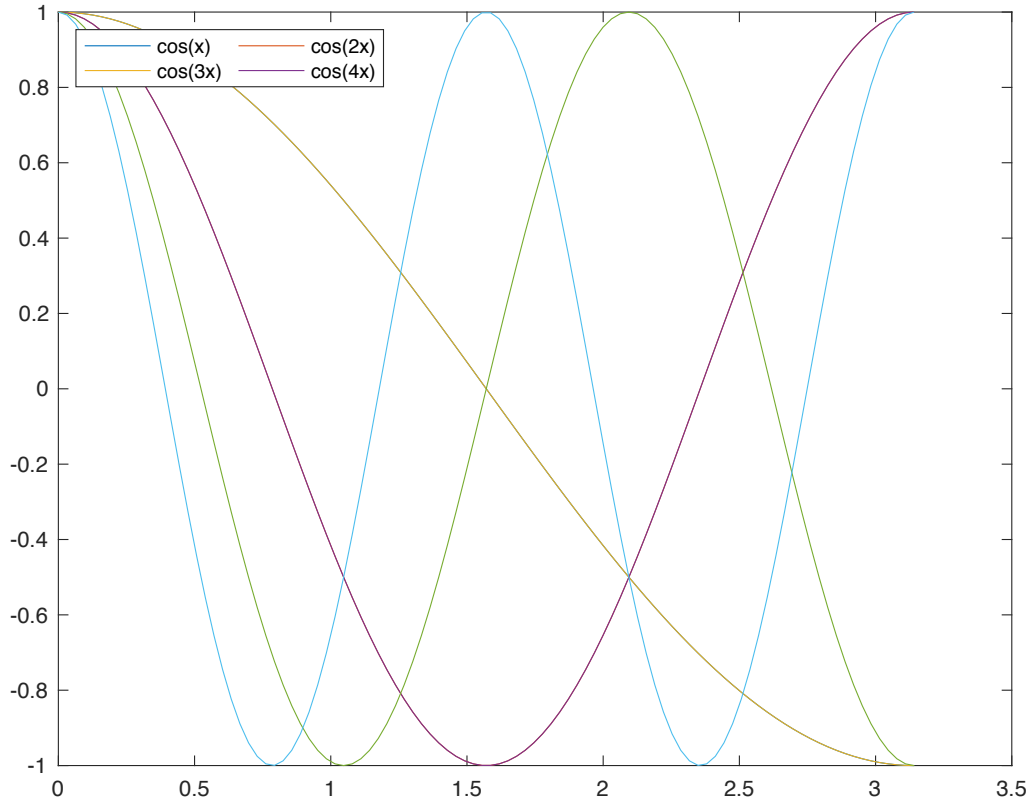


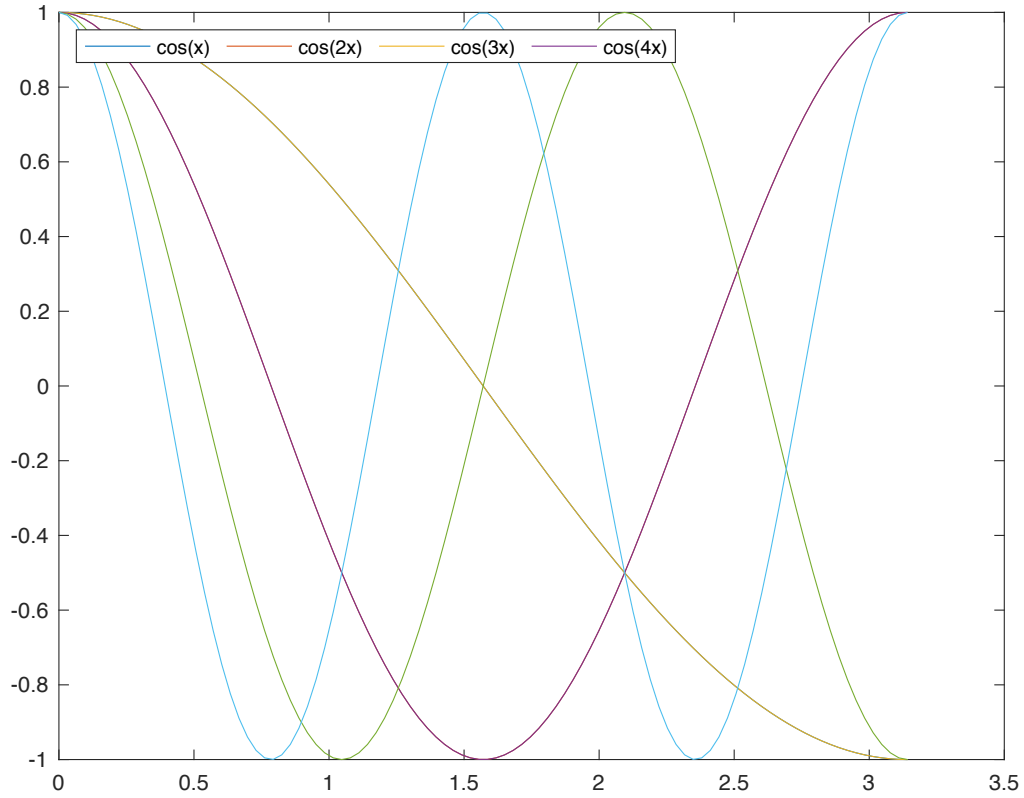
Setting properties

- By default, the legend orders the items from top to bottom along each column.

- To order the items from left to right along each row instead, set the Orientation property to 'horizontal'.

```
legend('cos(x)','cos(2x)','cos(3x)','cos(
4x)','Location','northwest','NumColumns',
2,'orientation','horizontal')
```






```
legend('cos(x)','cos(2x)','cos(3x)',
'cos(4x)','Location','northwest','
orientation','horizontal')
```

Included Subset of Graphics Objects in Legend

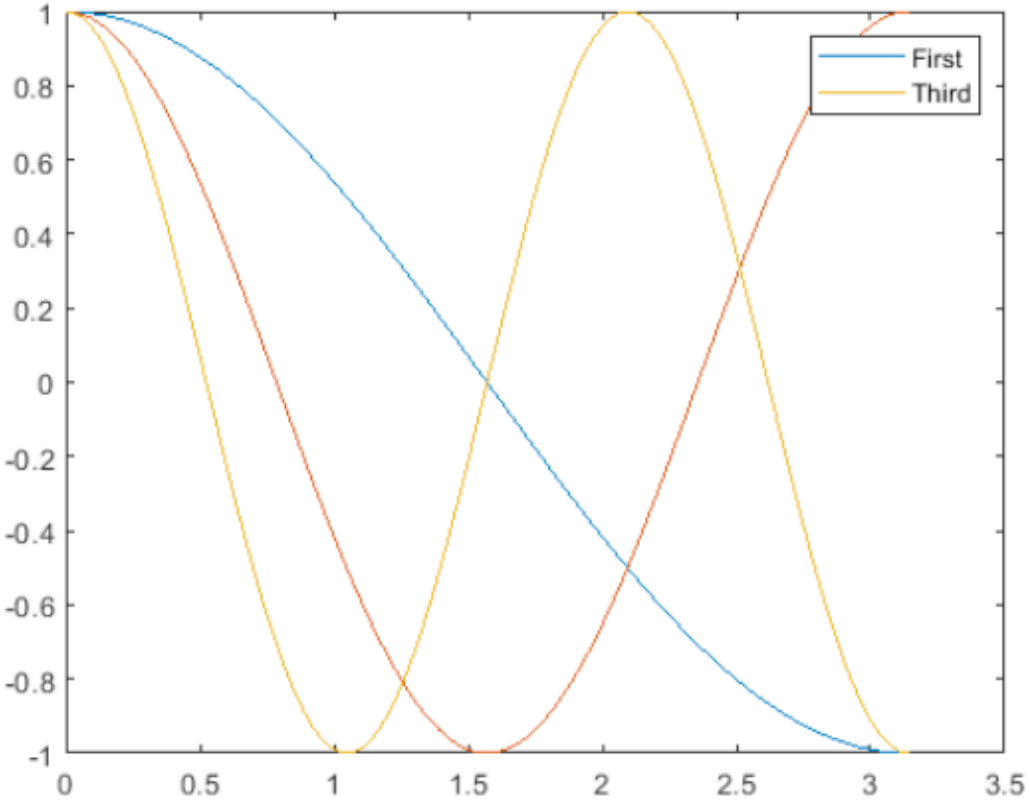
- If you do not want to include all of the plotted graphics objects in the legend, then you can specify the graphics objects that you want to include.

- Plot three lines and return the Line objects created.

- Create a legend that includes only two of the lines.

- Specify the first input argument as a vector of the Line objects to include.

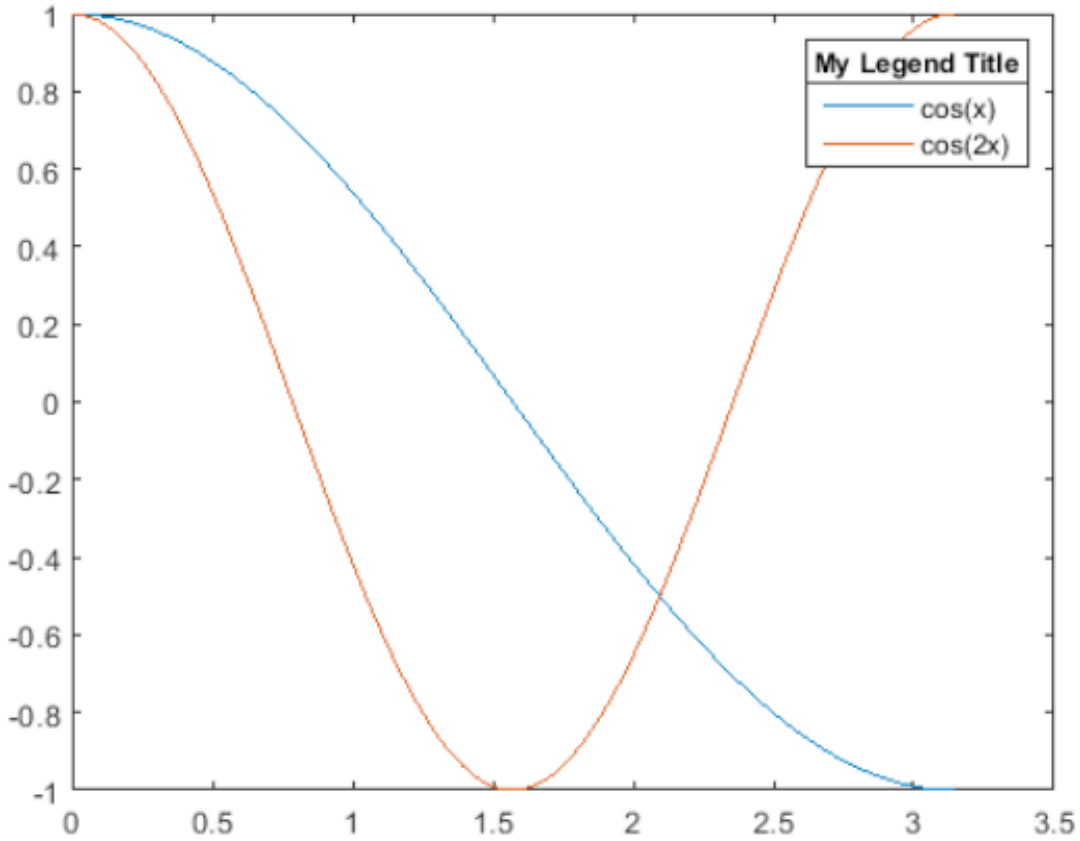
```
x = linspace(0,pi);
y1 = cos(x);
p1 = plot(x,y1);

hold on
y2 = cos(2*x);
p2 = plot(x,y2);

y3 = cos(3*x);
p3 = plot(x,y3);
hold off
legend([p1 p3],{'First','Third'})
```



Add Title to Legend

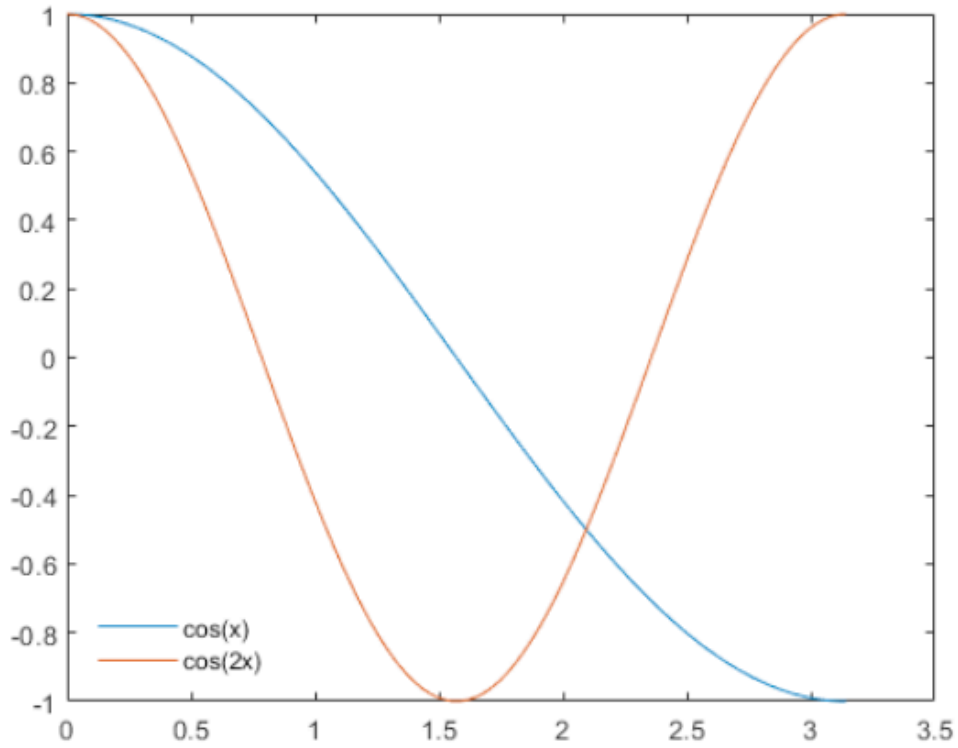
Plot two lines and create a legend. Then, add a title to the legend.

```
x = linspace(0,pi);
y1 = cos(x);
plot(x,y1)

hold on
y2 = cos(2*x);
plot(x,y2)
hold off

lgd = legend('cos(x)','cos(2x)');
title(lgd,'My Legend Title')
```



Remove Legend Background

```
x = linspace(0,pi);
y1 = cos(x);
plot(x,y1)

hold on
y2 = cos(2*x);
plot(x,y2)
hold off

legend({'cos(x)','cos(2x)'},'Location','southwest')
legend('boxoff')
```



Exercise: Learn how to modify legend appearance

Axis control, zoom in and zoom out

Once a plot is generated, you can change the axes limits with the axis command.

```
axis([xmin xmax ymin ymax])
```

```
axis('equal') or axis equal
axis('square')or axis square
axis('normal') or axis normal
axis('axis') or axis axis
axis('off') or axis off
```

```
axis( [-5 10 2 22] ) ;  sets the x-axis from -5 to 10, y-axis from 2 to 22,
axy = [-5 10 2 22] ;          axis(axy) ;       same as above, and
ax = [-5 10]; ay=[2 22];      axis([ax ay]); same as above.
```

- The axis command must come after the plot command to have the desired effect.

Semi-control of axes

It is possible to control only part of the axes limits and let MATLAB set the other limits automatically.

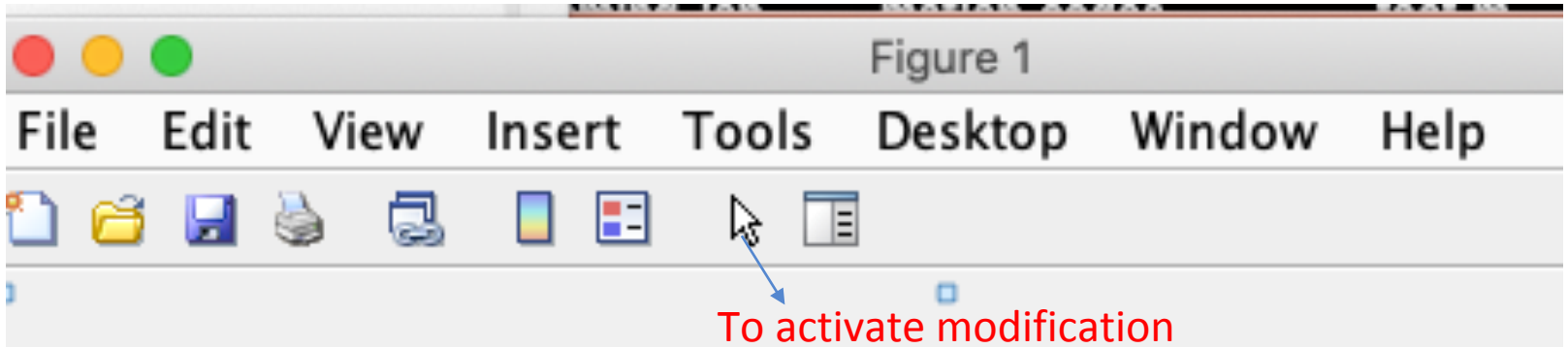
This is achieved by spedfying the desired limits in the axis command along with inf as the values of the limits that you would like to be set automatically.

For example,

```
axis ( [-5 10 -inf inf] )
axis ( [-5 inf -inf 22] )
```

Modifying plots with the plot editor

- To activate interactive tool for modification, go to the figure window and click on the left-leaning arrow in the menu bar.



- Now we can select and double-click (or right-click) on any object in the current plot to edit it.

- Double- clicking on the selected object brings up a property editor window where we can select and modify the current properties of the object.

- Using other tools from tools tab, we can modify and annotate figures.

Overlay plots

Method 1: Using the plot command to generate overlay plots

- If the entire set of data is available, plot command with multiple arguments may be used to generate an overlay plot.

- For example plot(x1,y1, x2,y2, ' : ' , x3,y3, ‘o’)

- Note that the vectors (xi, yi) must have the same length pairwise.

- If the length of all vectors is the same, then it is convenient to make a matrix of x vectors and a matrix of y vectors and then use the two matrices as the argument of the plot command.

- For example X=[x1 x2 x3] ; Y=[y1 y2 y3] ; plot(X,Y) produces a plot with three lines drawn in different colors.

- When plot command is used with matrix arguments, each column of the second argument matrix is plotted against the corresponding column of the first argument matrix.

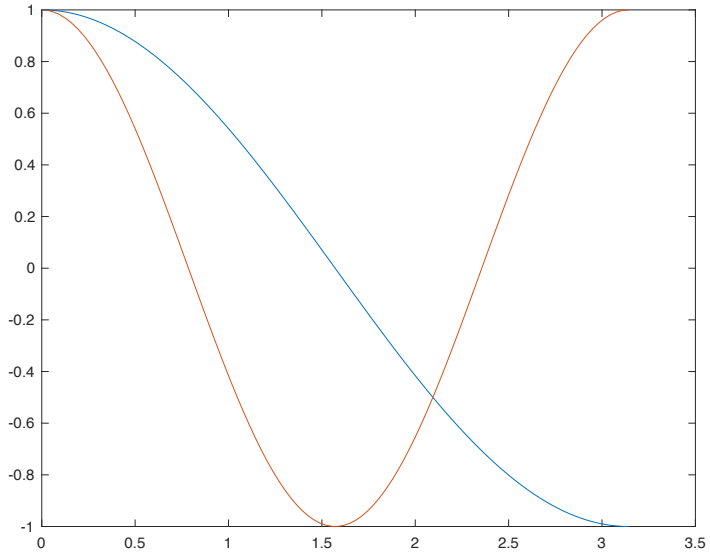
Method 2: Using the hold command to generate overlay plots

Another way of making overlay plots is with the hold command.

Invoking hold on at any point during a session freezes the current plot in the graphics window.

All subsequent plots generated by the plot command are simply added to the existing plot .

```
x = linspace(0,pi);
y1 = cos(x);
plot(x,y1)

hold on
y2 = cos(2*x);
plot(x,y2)
hold off
```



The hold command is useful for overlay plots when the entire data set to be plotted is not available at the same time, for example plotting within for loop.

Method 3: Using the line command to generate overlay plots

- line(x,y) plots a line in the current axes using the data in vectors x and y. If either x or y, or both are matrices, then line draws multiple lines.

- Unlike the plot function, line adds the line to the current axes without deleting other graphics objects or resetting axes properties.

line

Create primitive line

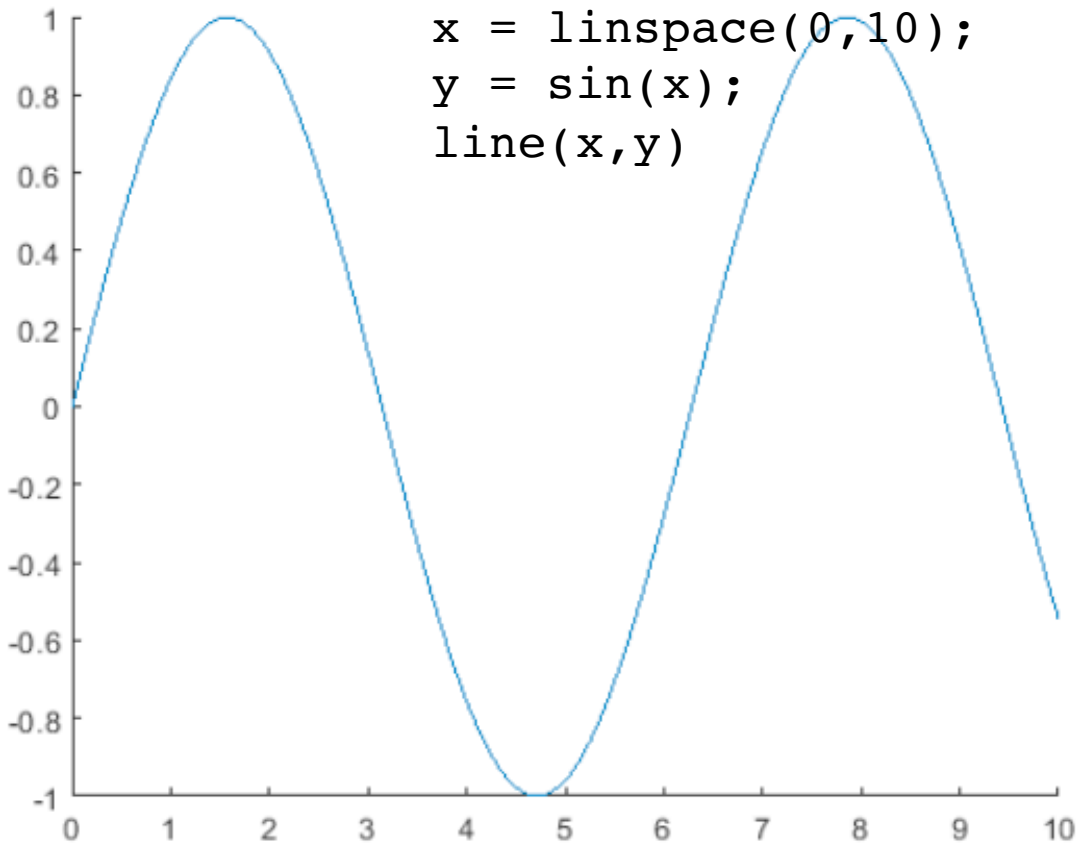
Syntax

```
line(x,y)
line(x,y,z)
line
```

```
line(__,Name,Value)
line(ax,__)
```

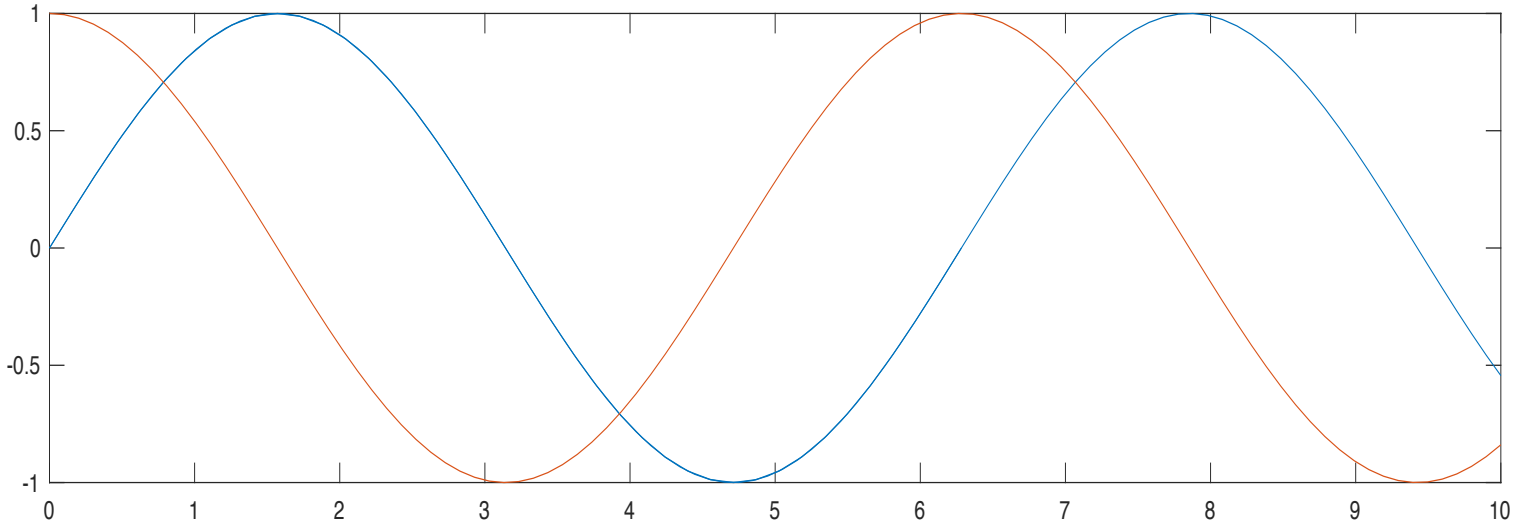
```
pl = line(__)
```

```
x = linspace(0,10);
y = sin(x);
line(x,y)
```



Plot Multiple Lines Using Matrix Data

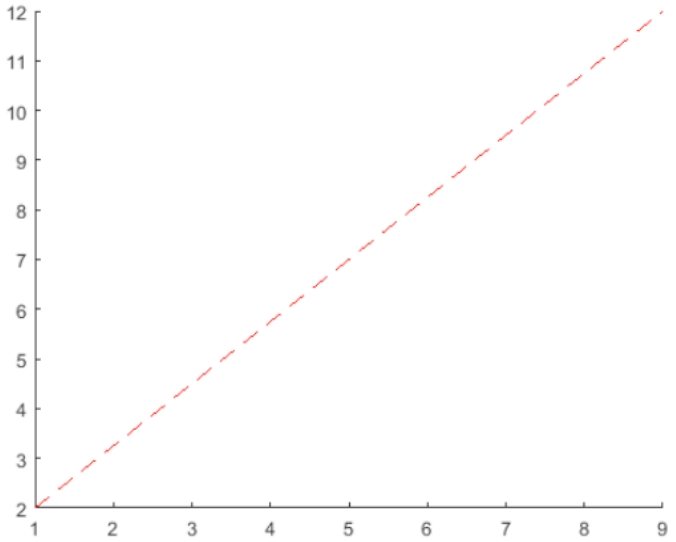
```
x = linspace(0,10)';
y = [sin(x) cos(x)];
line(x,y)
```



Specify Line Properties

Draw a red, dashed line between the points (1,2) and (9,12).
Set the Color and LineStyle properties as name-value pairs.

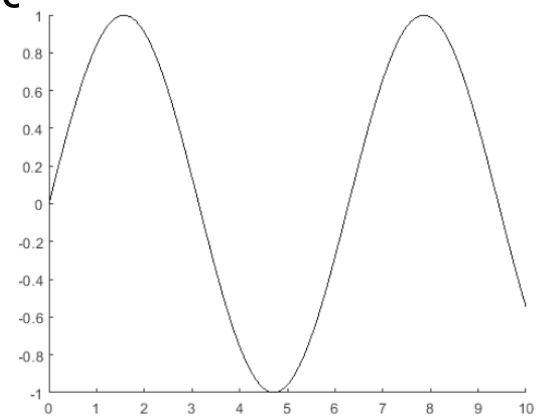
```
x = [1 9]; y = [2 12];
line(x,y,'Color','red','LineStyle','--')
```



Plot Line Using Low-Level Syntax

Create x and y as vectors. Then plot y versus x using the low-level version of the line function

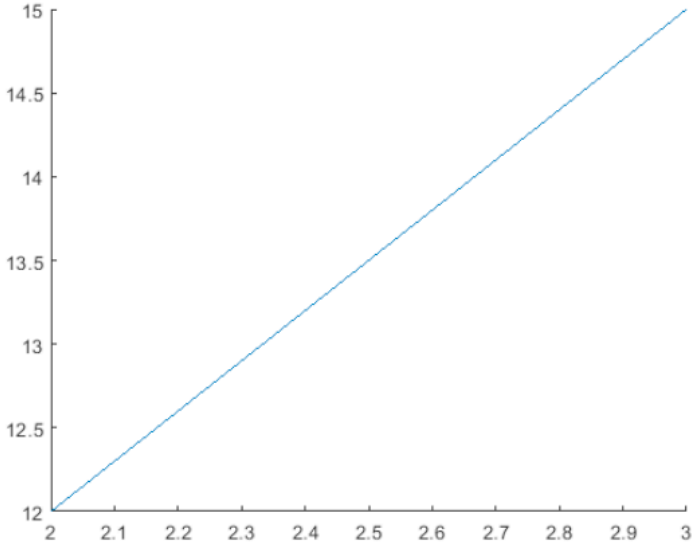
```
x = linspace(0,10);
y = sin(x);
line('XData',x,'YData',y)
```



Change Line Properties After Creation

First, draw a line from the point (3,15) to (2,12) and return the Line object. Then change the line to a green, dashed line. Use dot notation to set properties.

```
x = [3 2];
y = [15 12];
pl = line(x,y);
```



```
pl.Color = 'green';
pl.LineStyle = '--';
```

Adding a reference line to the current axes

Syntax

```
refline(m,b)
refline(coeffs)
refline
refline(ax, __ )
hline = refline( __ )
```

Description

`refline(m,b)` adds a reference line with slope `m` and intercept `b` to the current axes.

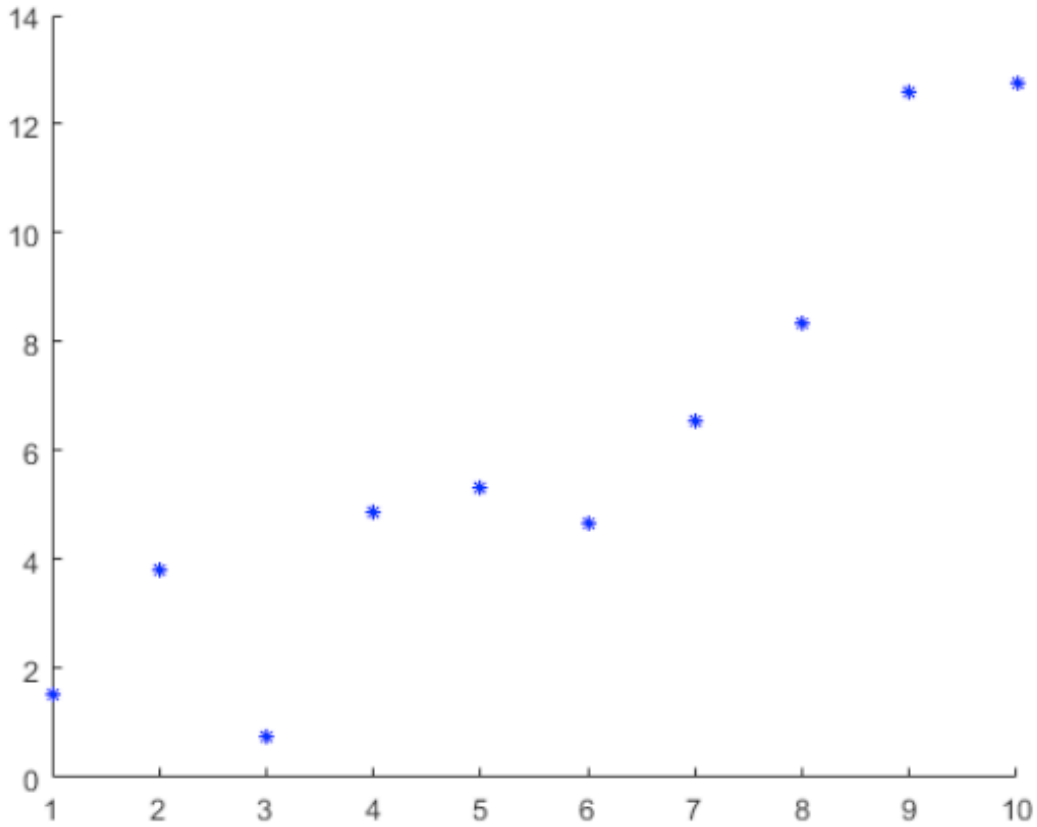
`refline(coeffs)` adds the line defined by the elements of the vector `coeffs` to the figure.

`refline` with no input arguments is equivalent to `lsline`.

`refline(ax, __ )` adds a reference line to the plot in the axis specified by `ax`, using any of the input arguments in the previous syntaxes.

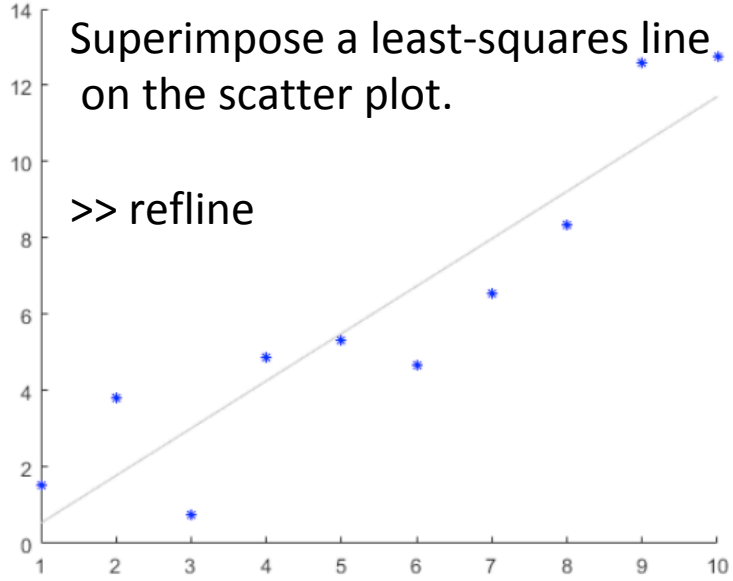
`hline = refline( __ )` returns the reference line object `hline` using any of the input arguments in the previous syntaxes. Use `hline` to modify properties of a specific reference line after you create it. For a list of properties, see Line Properties.

Add Reference Line at the Mean

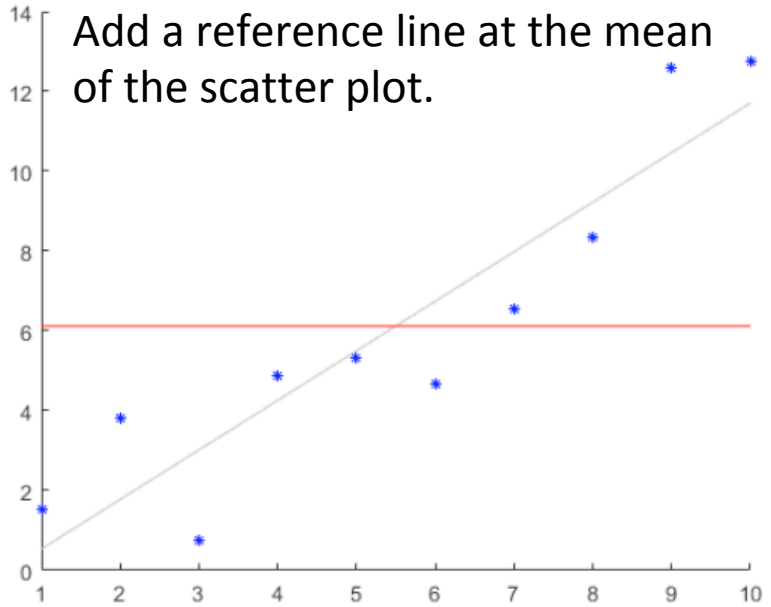


Superimpose a least-squares line on the scatter plot.

>> refline



Add a reference line at the mean of the scatter plot.



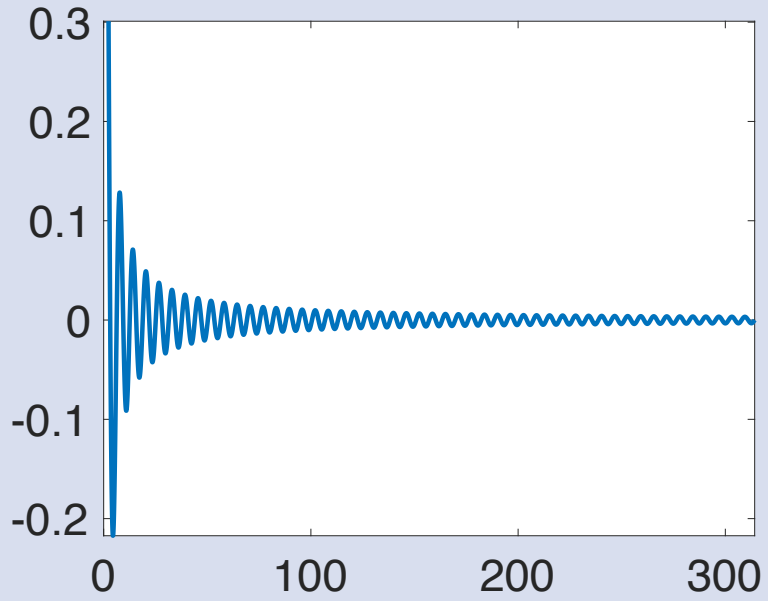

```
mu = mean(y);
hline = refline([0 mu]);
hline.Color = 'r';
```

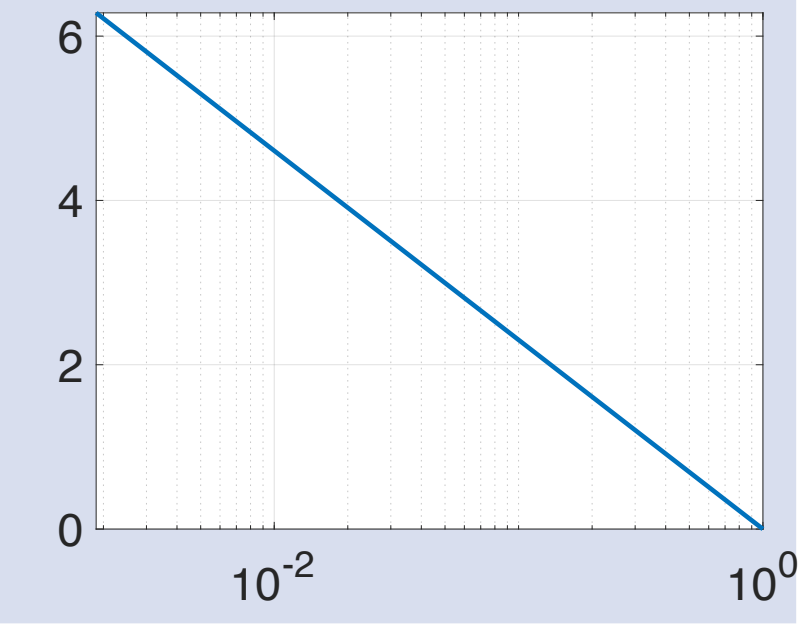
Lesson 21: Specialized 2-D plots

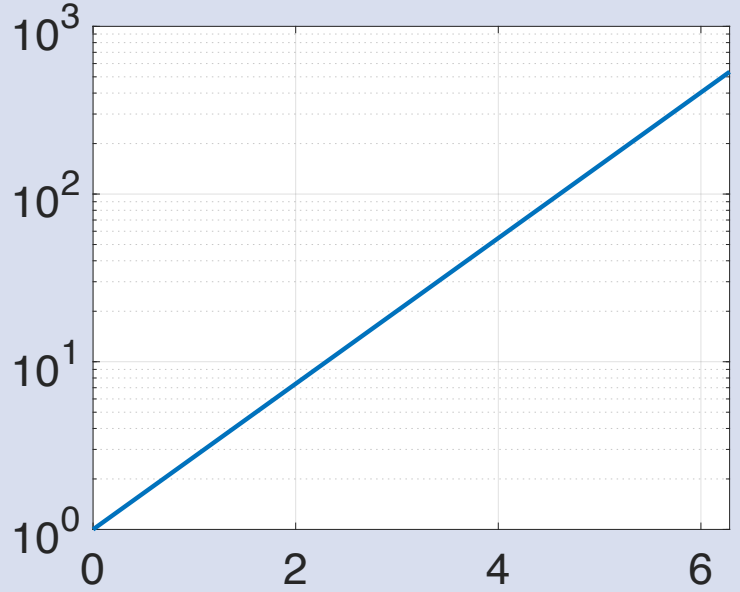
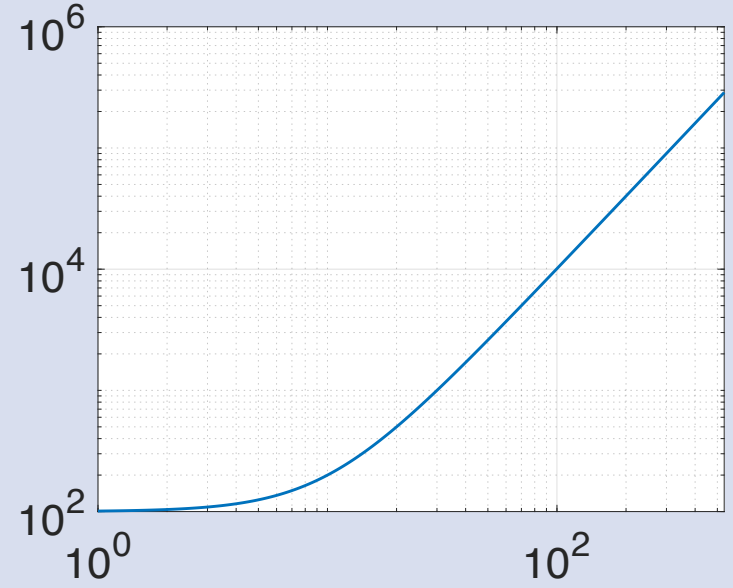
Learning objectives

To learn how to generate two dimensional plots in MATLAB.

Special 2d plots (alternatives of ‘plot’ command)

fplot *plots a function of a single variable*	fplot(@(x) sin(x)./x,[0 100*pi])	

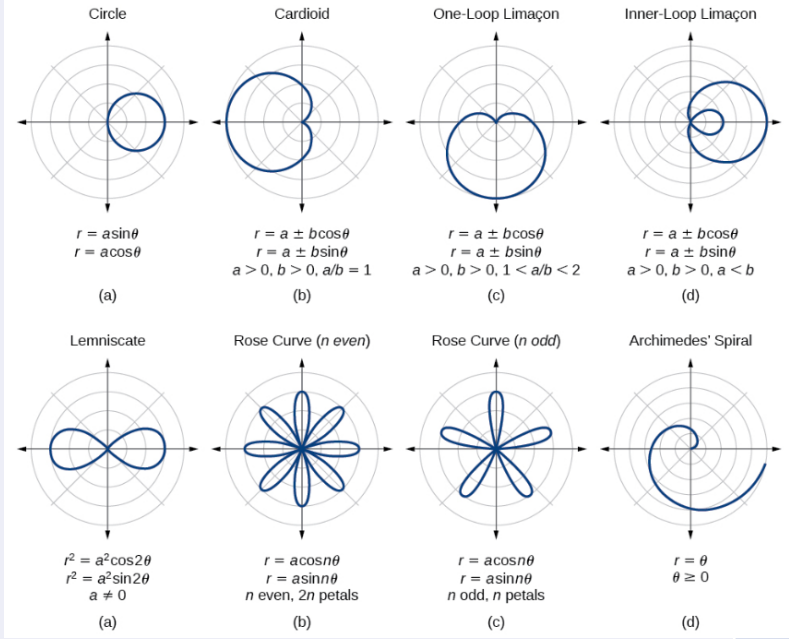
semilogx
 	
makes semilog plot with log scale on the x-axis
	t = linspace(0,2*pi,200);
	x = exp(-t);
	y = t;
	semilogx(x,y)
	grid on
		

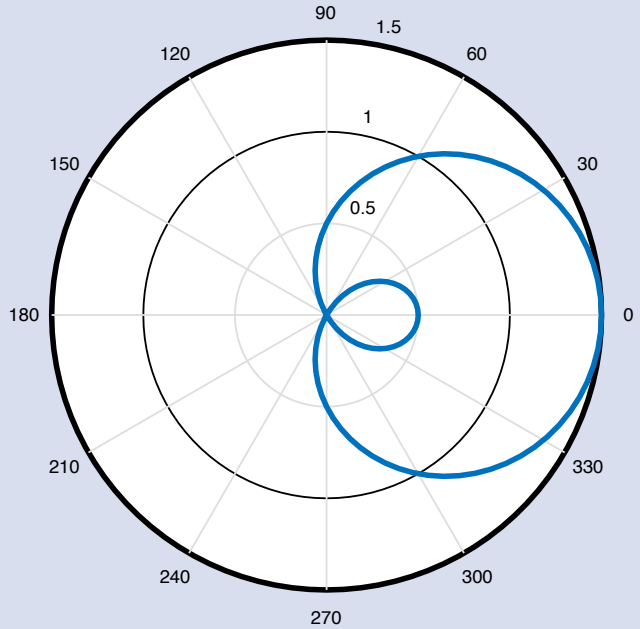
semilogy *makes semilog plot with log scale on the y-axis*	t = linspace(0,2*pi,200) ; semilogy(t,exp (t)) grid	
loglog *creates plot with log scale on both the x-axis and the y-axis*	t = linspace(0,2*pi,200); loglog(exp(t),100+exp(2*t)) grid on	

polar

plots curves in polar coordinates

```
t= 2*pi*[0:.01:1];
r = 0.5 + cos(t);
polar(t,r)
```

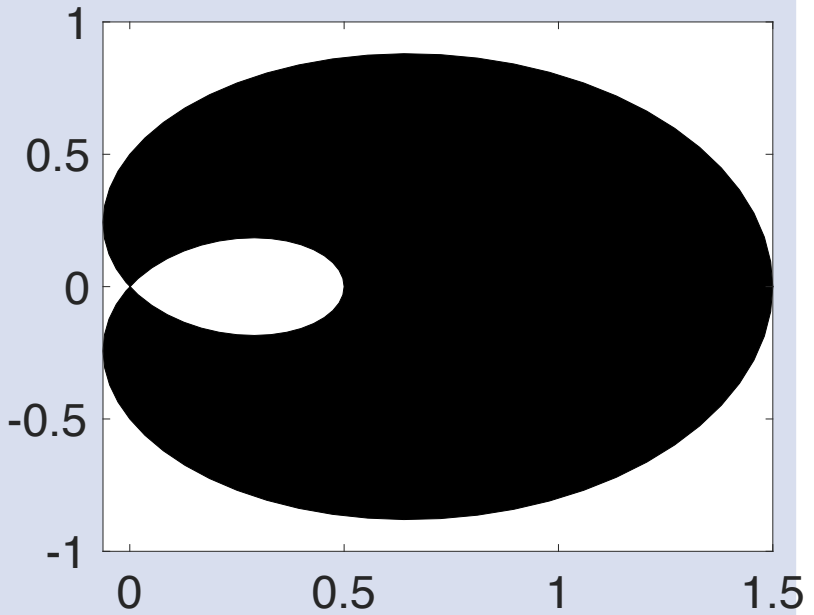


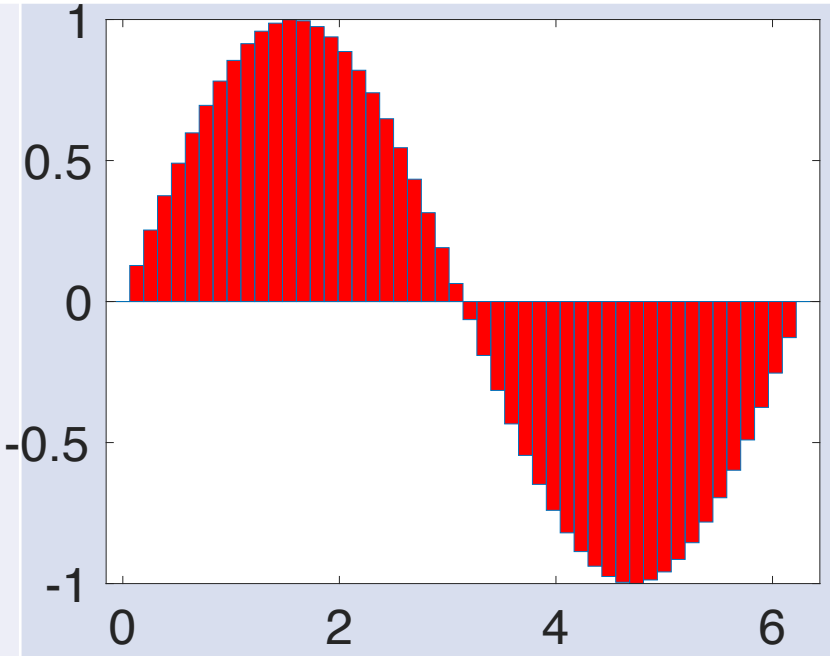


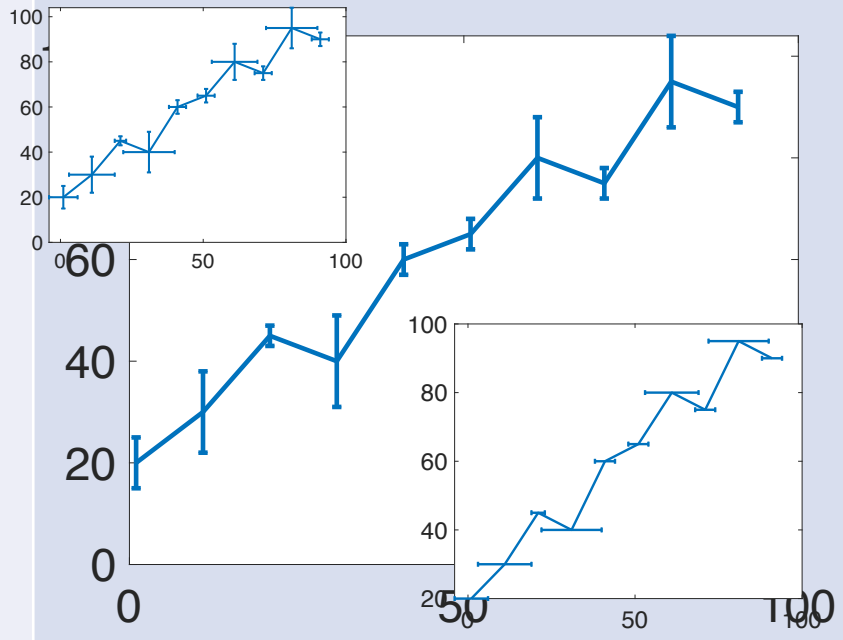
fill

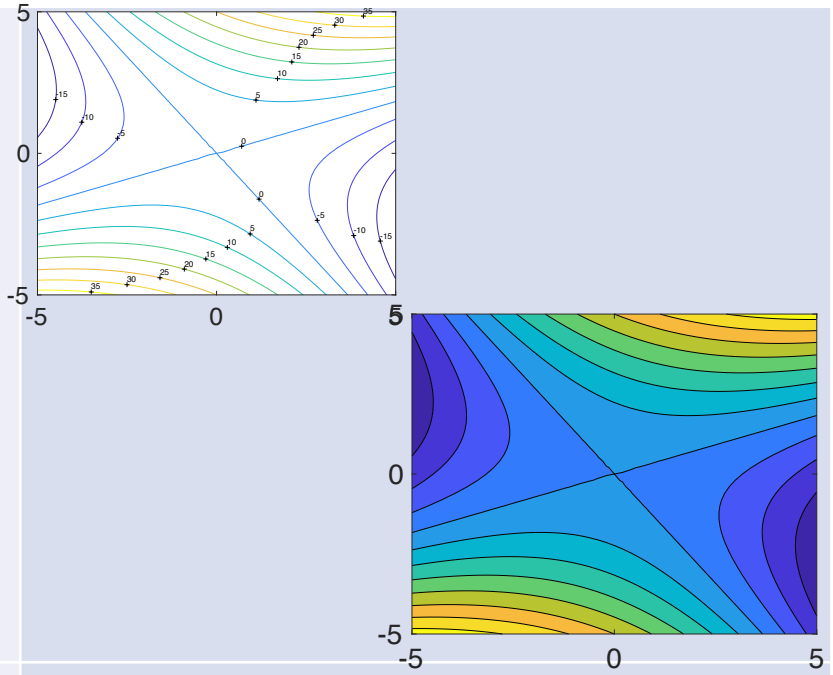
draws filled polygons of specified color

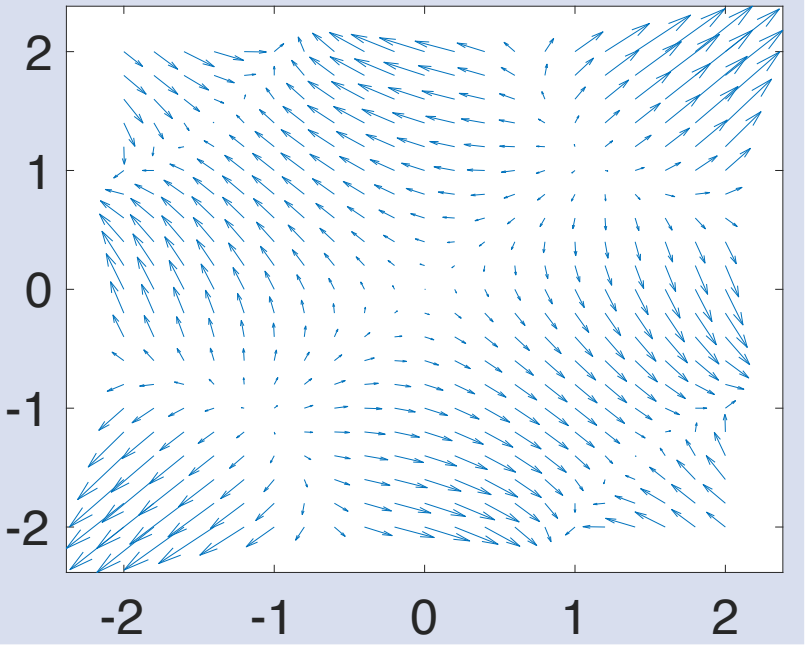
```
t= 2*pi*[0:.01:1];
r = 0.5 + cos(t);
x = r.*cos(t);
y = r.*sin(t);
fill(x,y,'k')
```

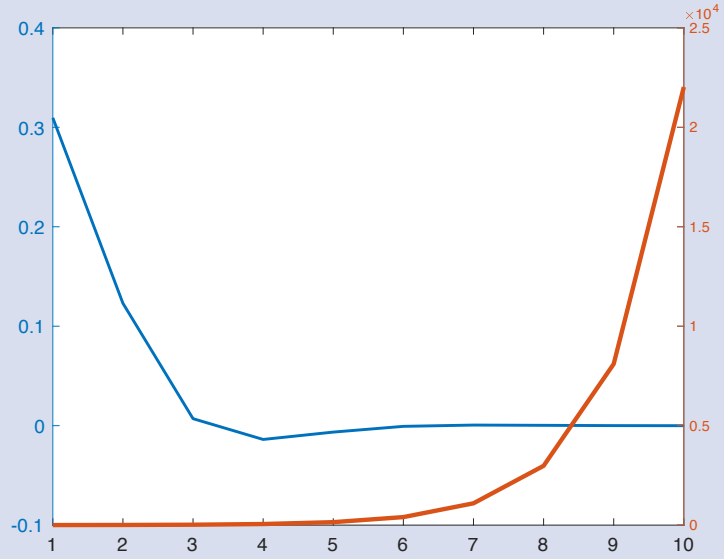


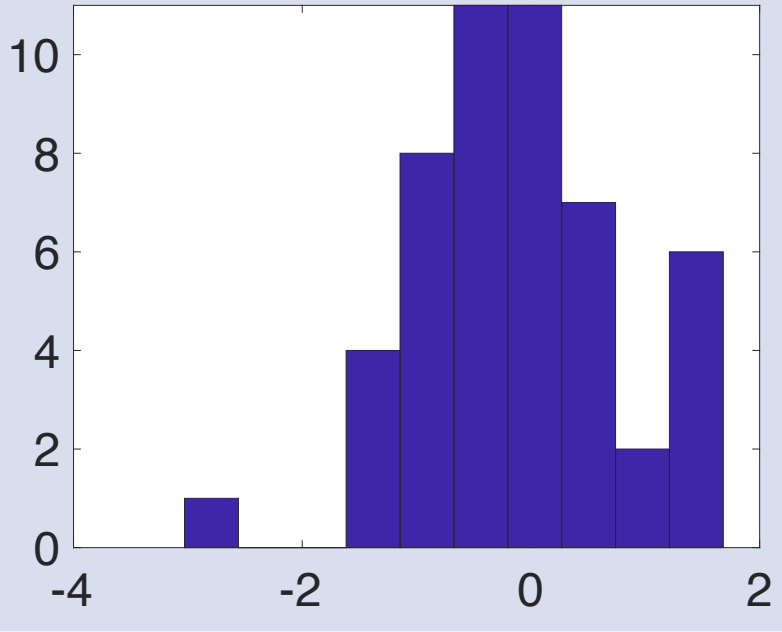
bar *creates a bar graph*	t=linspace(0,2*pi,50); bar(t,sin(t))	

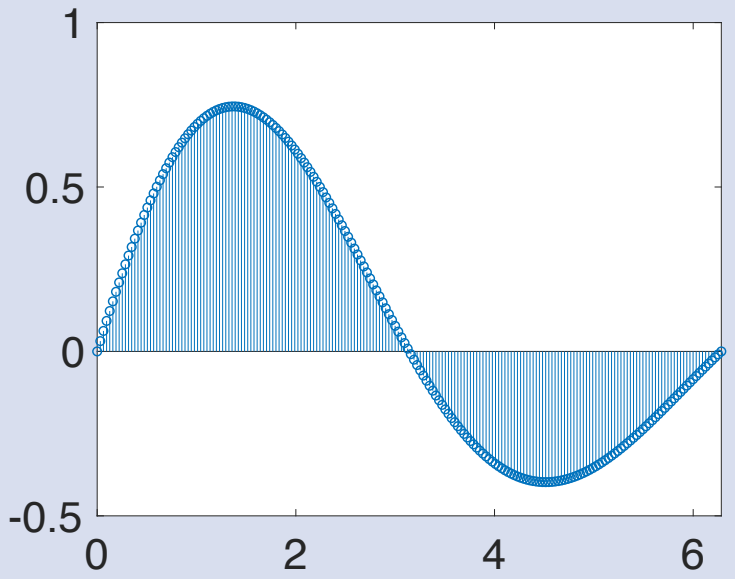
errorbar
 	
plots a graph and puts error bars
	Error bars show the confidence level of data or the deviation along a curve.
	
	x = 1:10:100;
	y = [20 30 45 40 60 65 80 75 95 90];
	err = [5 8 2 9 3 3 8 3 9 3];
	errorbar(x,y,err)
	errorbar(x,y,err,'horizontal’)
	errorbar(x,y,err,’both')
		

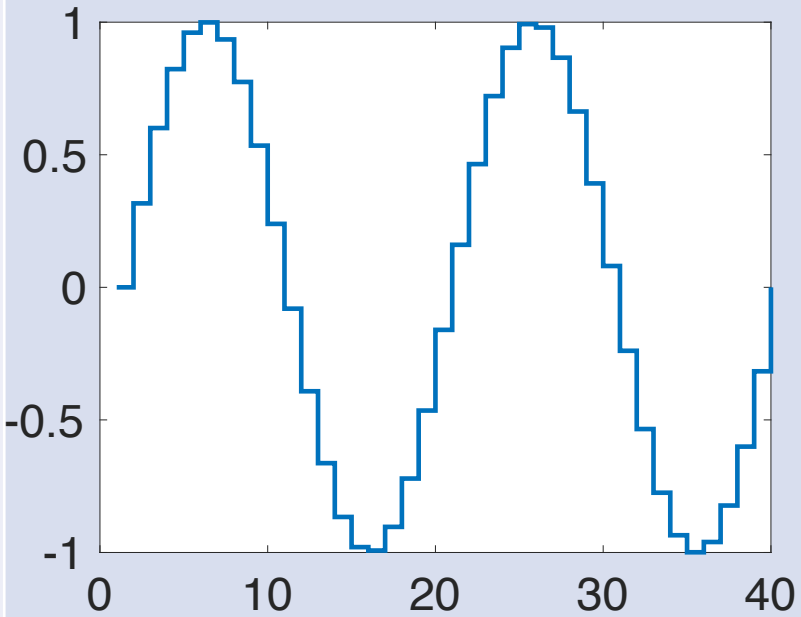
contour & contourf
 	
makes contour plots
	r = -5:.2:5;
	[X,Y] = meshgrid(r,r);
	Z = -.5*X.^2 + X.*Y + Y.^2;
	cs = contour(X,Y,Z);
	clabel(cs)
	
	cs = contourf(X,Y,Z);
		

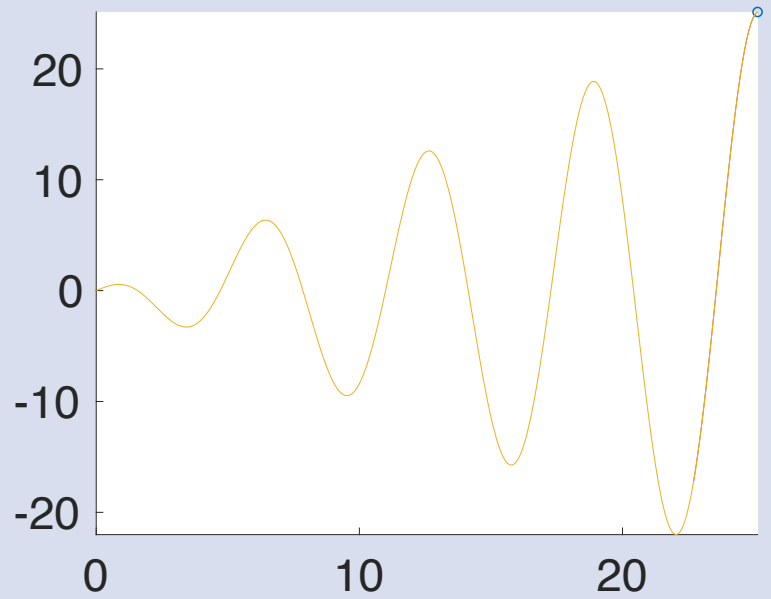
quiver
 	
plots vector fields
	r=-2: .2:2;
	[X,Y]= meshgrid (r, r);
	Z=X.^2- 5*sin(X.*Y)+Y.^2;
	[dx,dy]=gradient(Z,.2,.2);
	quiver(X,Y,dx,dy,2);
		

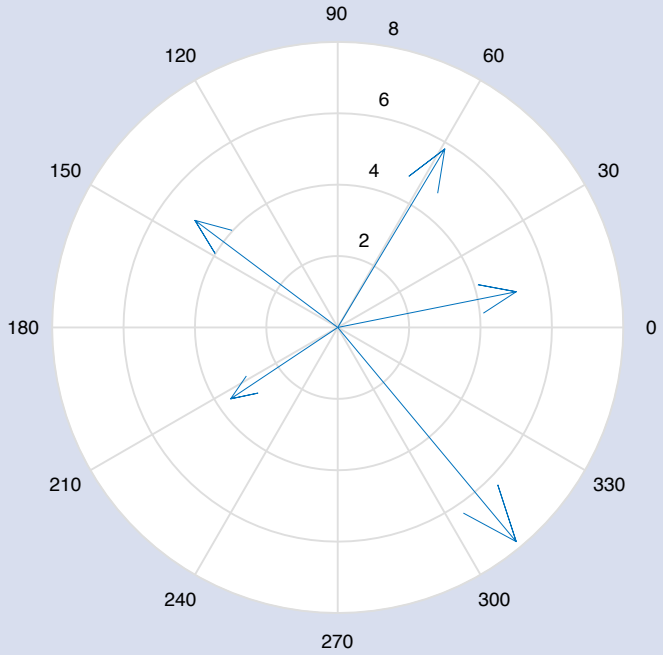
ploYYy
 	
makes a double y-axis plot
	x= 1:1:10;
	y1=exp(-x).*sin(x);
	y2=exp(x);
	plotyy(x, y1, x, y2);
		

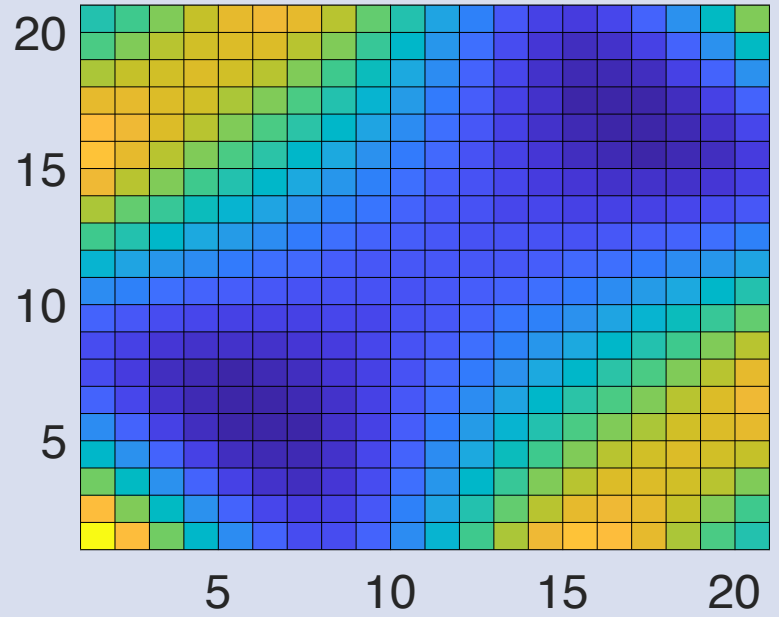
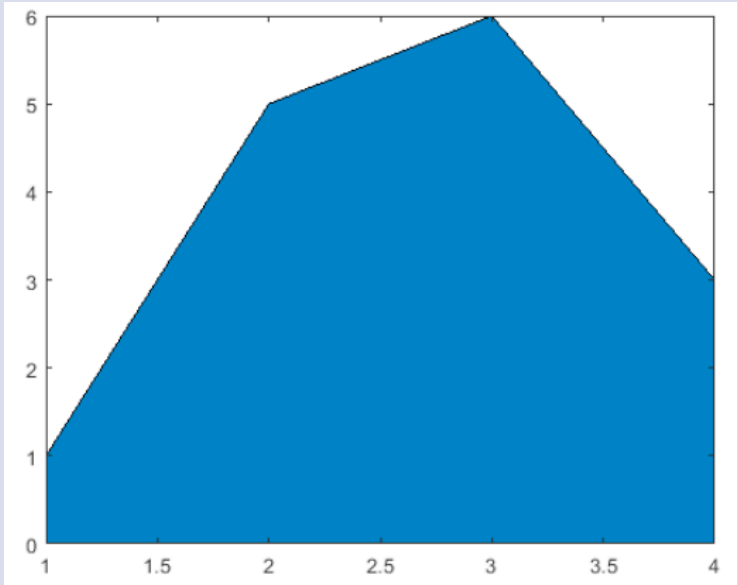
histogram *makes histograms*	y =randn(50,1); histogram(y)	

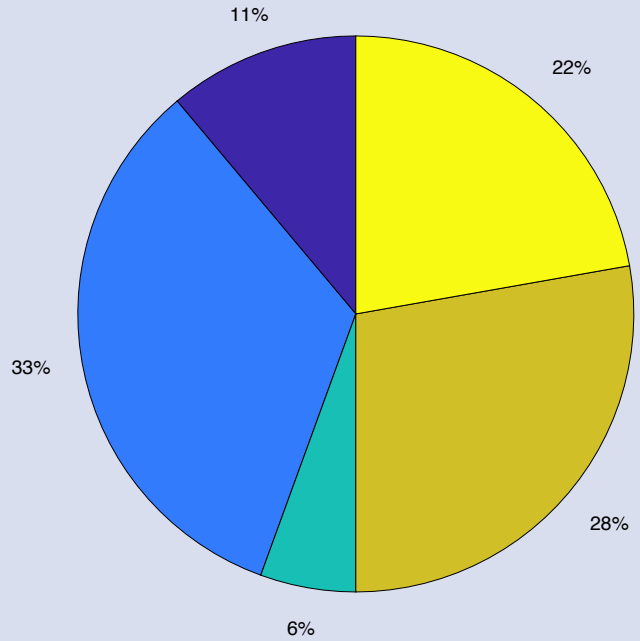
stem
 	
plots a stem graph.
	Discrete sequence of data values.
	The data values are indicated by circles terminating each stem.
	
	t = linspace(0,2*pi,200);
	f = exp(-0.2*t).*sin(t);
	stem(t,f)
		

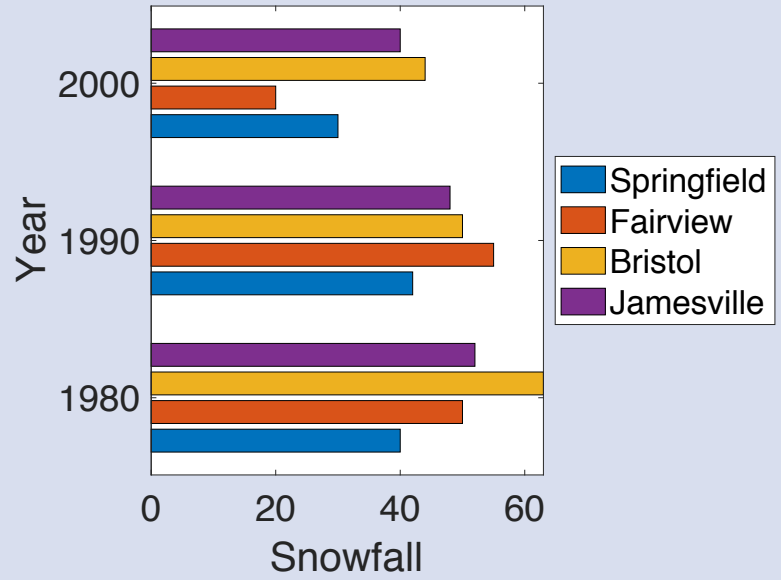
stairs *plots a stair graph*	X = linspace(0,4*pi,40); Y = sin(X); stairs(Y)	

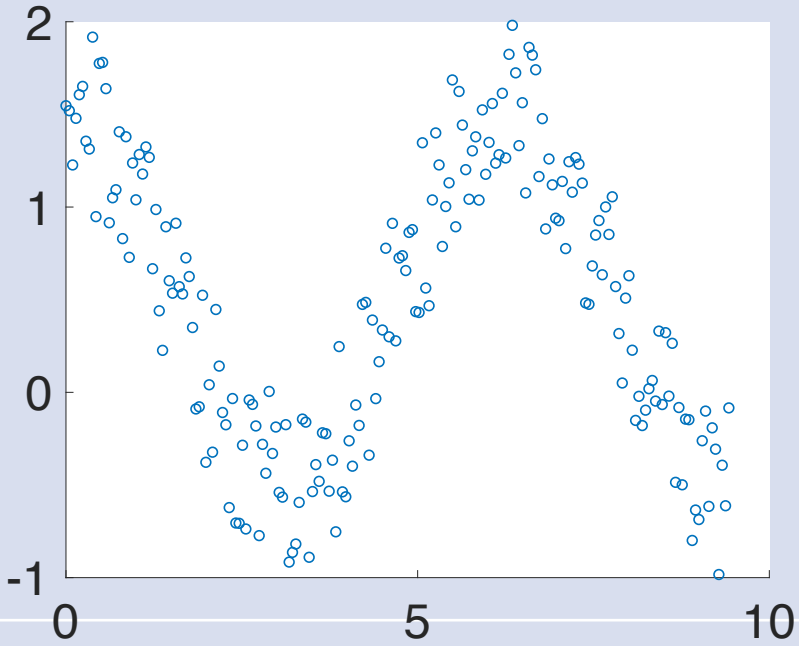
comet
 	
makes an animated 2-D plot
	x = linspace(0, 8*pi, 300);
	y = x .* cos(x);
	comet(x,y)
		

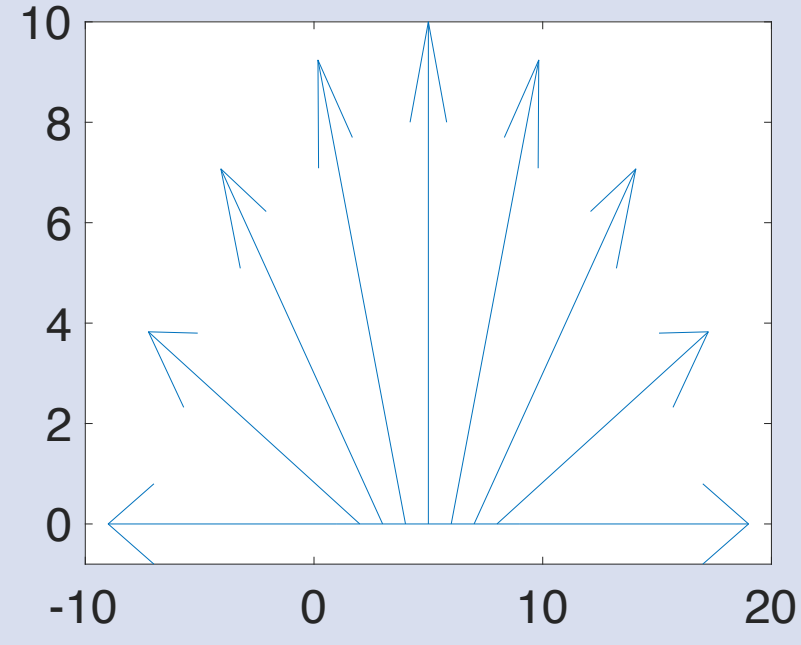
compass
 	
creates arrow graph for complex numbers
	u = [5 3 -4 -3 5];
	v = [1 5 3 -2 -6];
	compass(u,v)
		

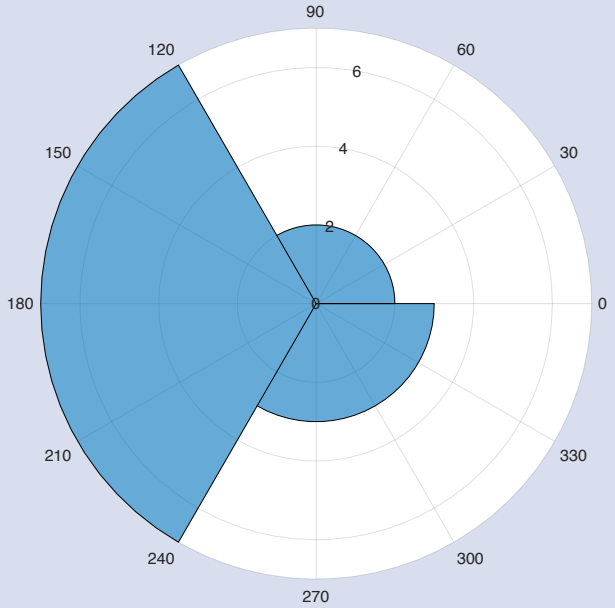
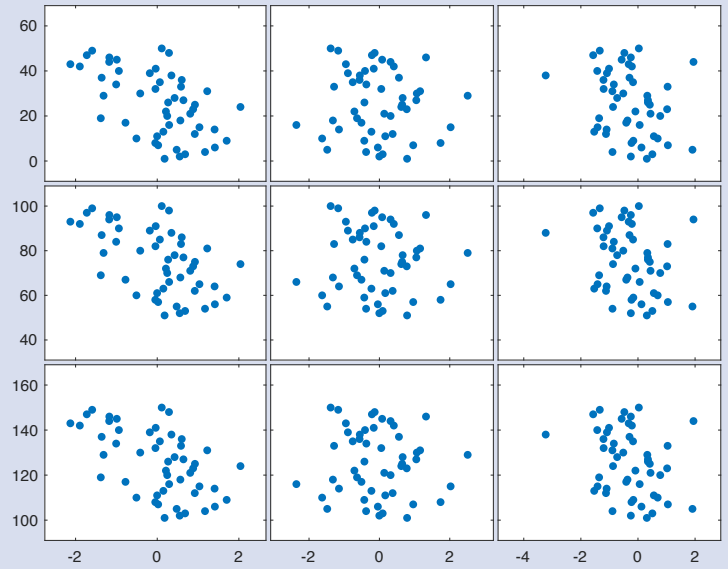
pcolor *makes pseudocolor plot of a matrix*	r = -2:.2:2; [X,Y] = meshgrid(r,r); Z = X.^2 - 5*sin(X.*Y)+Y.^2; pcolor(Z),	
area *creates a filled area plot*	area(X,Y) plots the values in Y against the x-coordinates X. The function then fills the areas between the curves based on the shape of Y. y = [1 5 6 3]; area(y)	

pie *creates a pie chart*	X = [1 3 0.5 2.5 2]; pie(X)	

barh
 	
creates a horizontal bar graph
	x = [1980 1990 2000];
	y = [40 50 63 52; 42 55 50 48; 30 20 44 40];
	barh(x,y)
	xlabel('Snowfall')
	ylabel('Year')
	legend({'Springfield','Fairview','Bristol','Jamesville'})
		

scatter
 	
creates a scatter plot
	scatter(x,y) creates a scatter plot with circles at the locations specified by the vectors x and y. This type of graph is also known as a bubble plot.
	
	x = linspace(0,3*pi,200);
	y = cos(x) + rand(1,200);
	scatter(x,y)
		

feather
 	
makes a feather plot
	feather(U,V) plots arrows originating from the *x*-axis. Specify the direction of arrows using the Cartesian components U and V, with U indicating the *x*-components and V indicating the *y*-components.
	t = -pi/2:pi/8:pi/2;
	u = 10*sin(t); v = 10*cos(t);
	feather(u,v)
	
	th = -pi/2:pi/16:0; r = 10*ones(size(th));
	[u,v] = pol2cart(th,r); feather(u,v)
		

Polarhistogram *histogram plot in polar coordinates*	polarhistogram(theta) creates a histogram plot in polar coordinates by sorting the values in theta into equally spaced bins. Specify the values in radians. theta = [0.1 1.1 5.4 3.4 2.3 4.5 3.2 3.4 5.6 2.3 2.1 3.5]; polarhistogram(theta)	
plotmatrix makes a scatter plot of a matrix	plotmatrix(X,Y) creates a matrix of subaxes containing scatter plots of the columns of X against the columns of Y. If X is *p*-by-*n* and Y is *p*-by-*m*, then plotmatrix produces an *n*-by-*m* matrix of subaxes. X = randn(50,3); Y = reshape(1:150,50,3); plotmatrix(X,Y) subplot in the ith row, jth column of the figure is a scatter plot of the ith column of Y against the jth column of X	

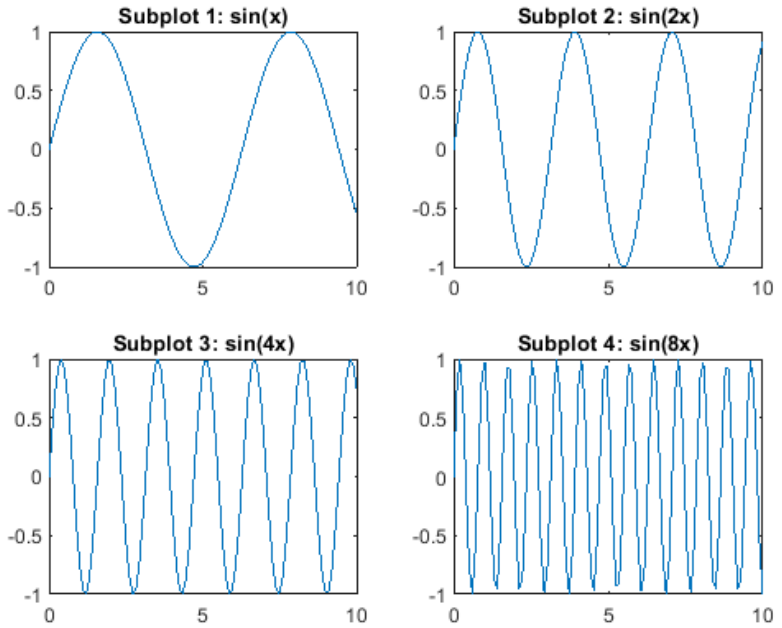
Using subplot for Multiple Graphs

If you want to make a few plots and place the plots side by side (not overlay), use the subplot command to design your layout. The subplot command requires three integer arguments:

subplot (m , n , p)

- Subplot divides the graphics window into m x n subwindows and puts the plot generated by the next plotting command into the pth subwindow, where the sub- windows are counted row-wise.

- Thus, the command subplot (2 , 2 , 3) , plot(x , y) divides the graphics window into four subwindows and plots y versus subwindow, which is the first subwindow in the second row.



Exercises

Use a single script for exercise 1 to3.

1 Plot the oscillations y1 = cost, y2 = cos3t and their sum y3 = y1 + y2, for $0 \le t \le 4\pi$, on the same axes. Here t is measured in seconds and y1, y2, y3 are measured in cm.

2 At time t seconds, $t \ge 0$, a moving point has coordinates x = sin2t, y = cos3t (metres), and so its speed is given by

$$speed(t) = \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} = \sqrt{4\cos^2 2t + 9\sin^2 3t}.$$

Plot the path taken by the point over one cycle $0 \le t \le 2\pi$, where t is regarded as a parameter, and also plot the speed against time.

3 repeat example 2 using comet plotting.

4 Draw phase portraits of $\dot{x_1} = -x_1 - 2x_2x_1^2 + x_2, \qquad \dot{x_2} = -x_1 - x_2$

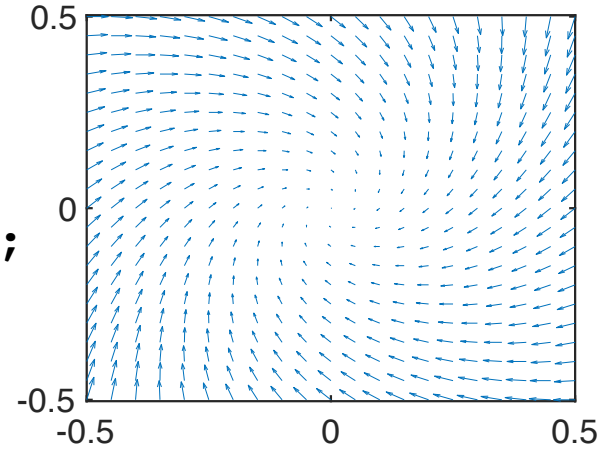
$$-0.5 \le x_1 \le 0.5, \; -0.5 \le x_2 \le 0.5$$

Solutions

```
clear all
t=linspace(0,4*pi,201);
y1=cos(t); y2=cos(3*t); y3=y1+y2;
figure(1)
plot(t,y1,'b--',t,y2,'g:',t,y3,'r')
axis([0 4*pi -2 2]) % specifies the axes
limits
legend('y1=cos(t)','y2=cos(3*t)','y3=y1+y
2')
hold on
plot([0 4*pi],[0 0],'k') % adds the t
axis in black
xlabel('time in seconds')
ylabel('displacement in cm')
title('oscillations')
hold off
```

```
clear all
t=linspace(0,2*pi,500);
x=sin(2*t);
y=cos(3*t);
figure(2)
plot(x,y)
axis equal % same scale (metres) on each axis
speed=sqrt(4*cos(2*t).^2+9*sin(3*t).^2); % note
the dots
figure(3)
axis([0 2*pi 0 4])
hold on
plot(t,speed)
hold off
```

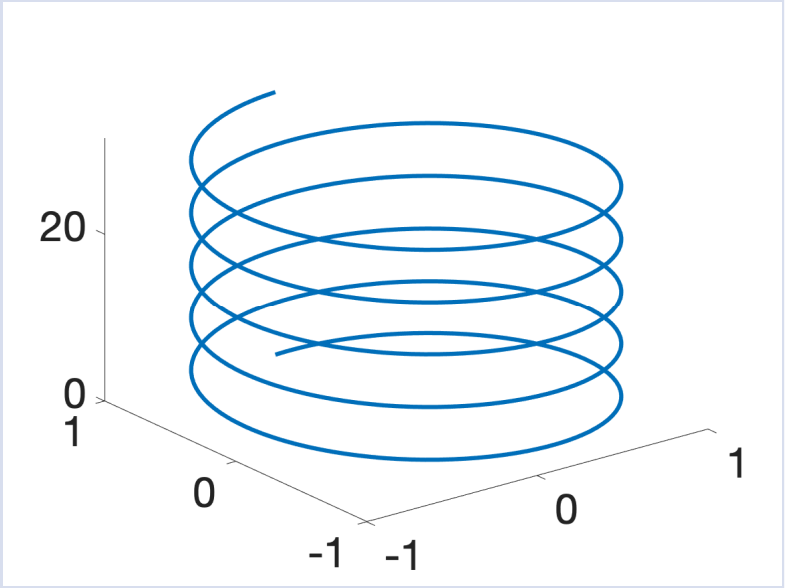
```
[x1, x2] = meshgrid(-.5:0.05:0.5, -.5:.05:.5);
x1dot = -x1 - 2 *x2 .*x1.^2+x2;
x2dot = -x1-x2;
quiver(x1,x2,x1dot, x2dot)
```

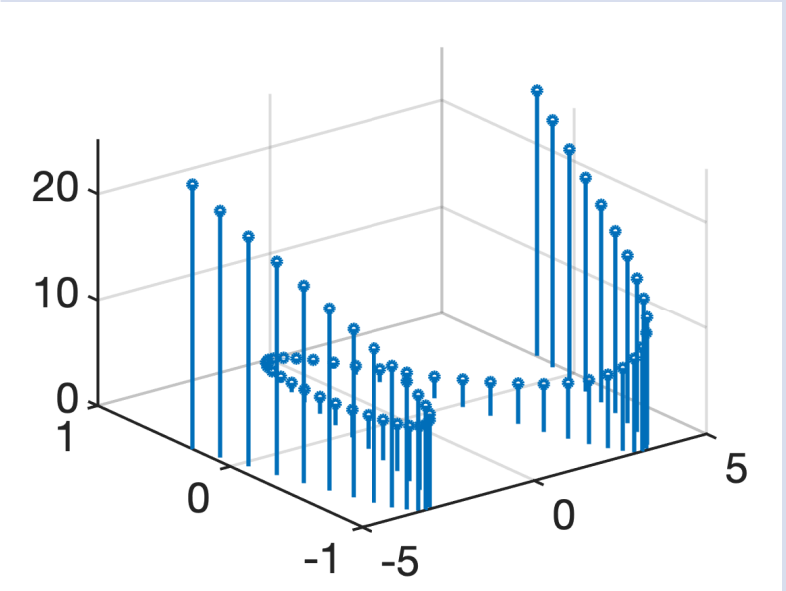


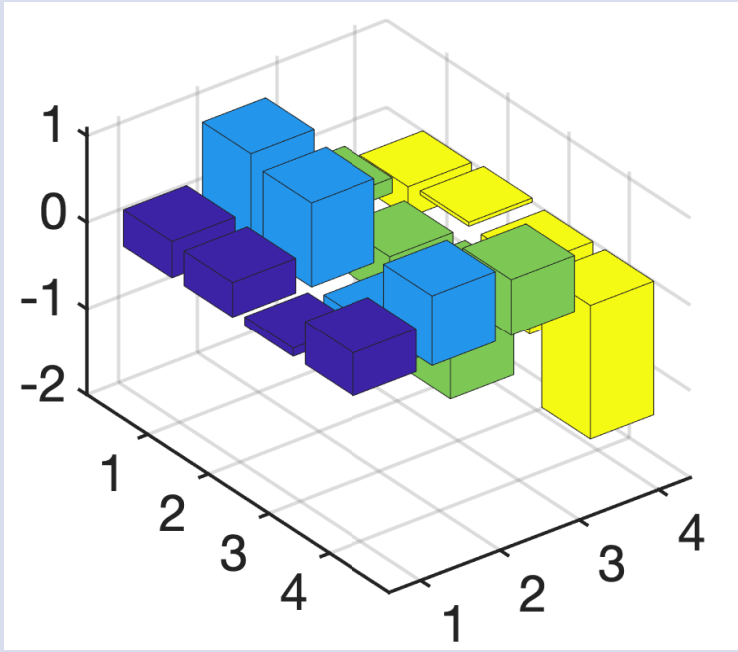
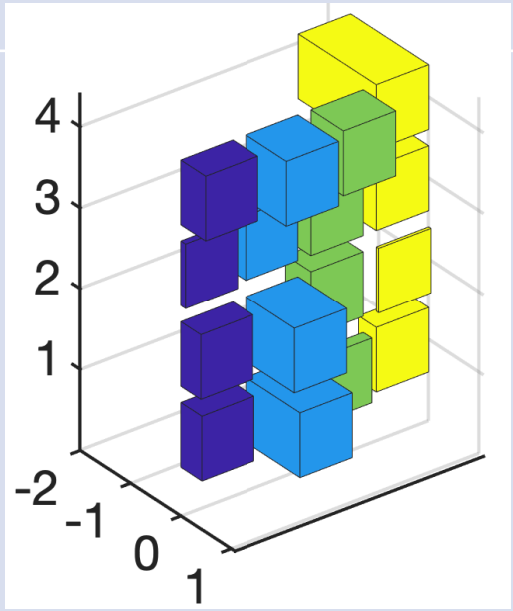
Lesson 22: 3-D plots

Learning objectives

To learn how to create three dimensional plots in MATLAB.

plot3
 	
creates a 3d curve in space
	x = sin(t);
	y = cos(t);
	z = 0:pi/50:10*pi;
	plot3(x,y,z)
		

stem3
 	
Create a discrete data plot using stem in 3d
	x = linspace(-5,5,60);
	y = cos(x);
	z = x.^2;
	stem3(x,y,z)
		

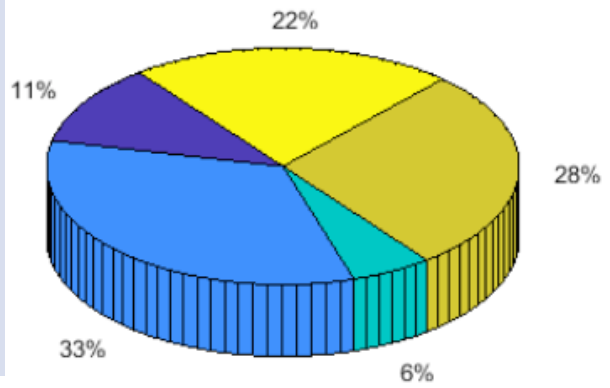
bar3 *creates a 3d bar plot*	z=randn(4); bar3(z) each element in Z corresponds to one bar. When Z is a vector, the *y*-axis scale ranges from 1 to length(Z). When Z is a matrix, the *y*-axis scale ranges from 1 to the number of rows in Z. bar3(Y,Z) draws a bar chart of the elements in Z at the locations specified in Y, where Y is a vector defining the *y* values for the vertical bars.	
bar3h *makes a horizontal bar plot in 3d*	z=randn(4); bar3h(z)	

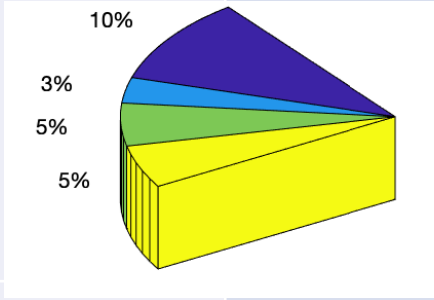
pie3

creates a 3d pie chart

```
x = [1,3,0.5,2.5,2];  pie3(x);
x = [0.1,.03,0.05,0.05]; pie3(x)
```

The values in X are normalized via X/sum(X) to determine the area of each slice of the pie. If sum(X)1, the values in X directly specify the area of the pie slices. MATLAB draws only a partial pie if sum(X)<1.



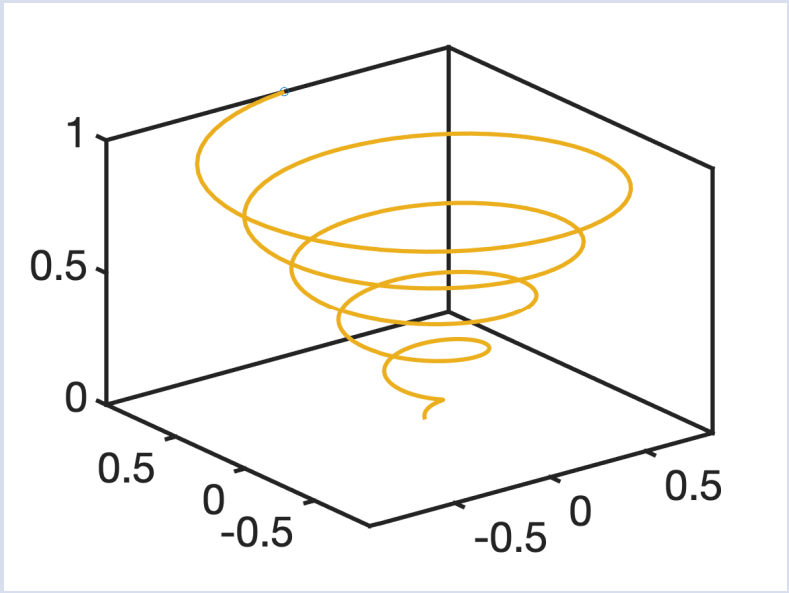


comet3

Creates animated 3d plot

```
t = 0:0.1:10*pi;
r = linspace (0, 1, numel (t));
z = linspace (0, 1, numel (t));
x=r.*sin(t);
y=r.*cos(t);
comet3(x,y,z)
 box on
```

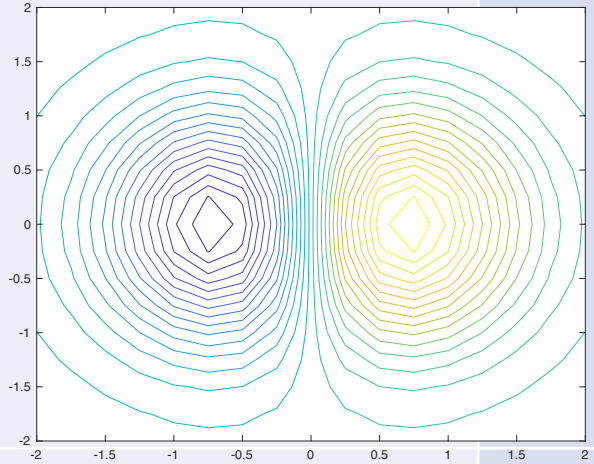
Try: comet3(r.*sin(t), r.*cos(t), z, 0.00001);

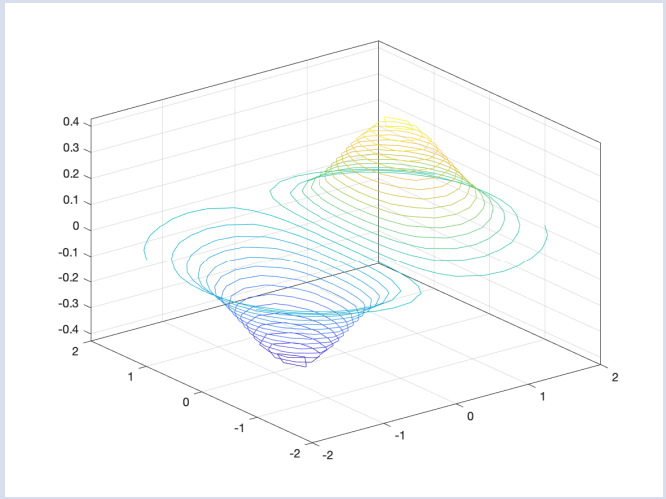


contour3

creates a 3d contour plot of a surface defined on a rectangular grid

```
[X,Y] = meshgrid([-2:.25:2]);
Z = X.*exp(-X.^2-Y.^2);
contour3(X,Y,Z,30)

contour(X,Y,Z,30)
```

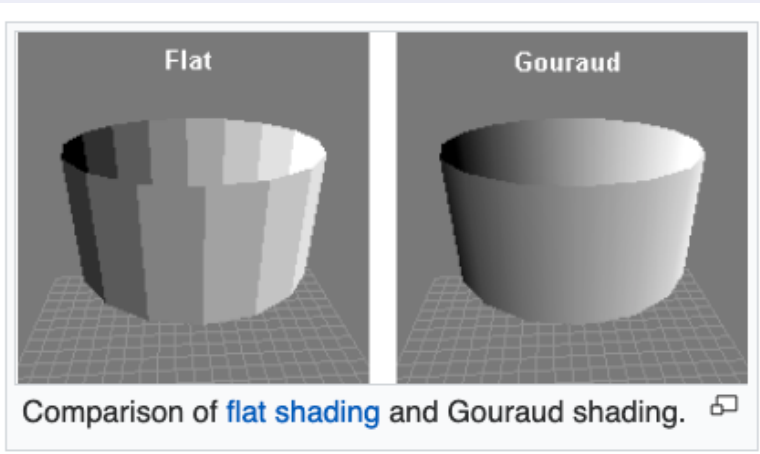




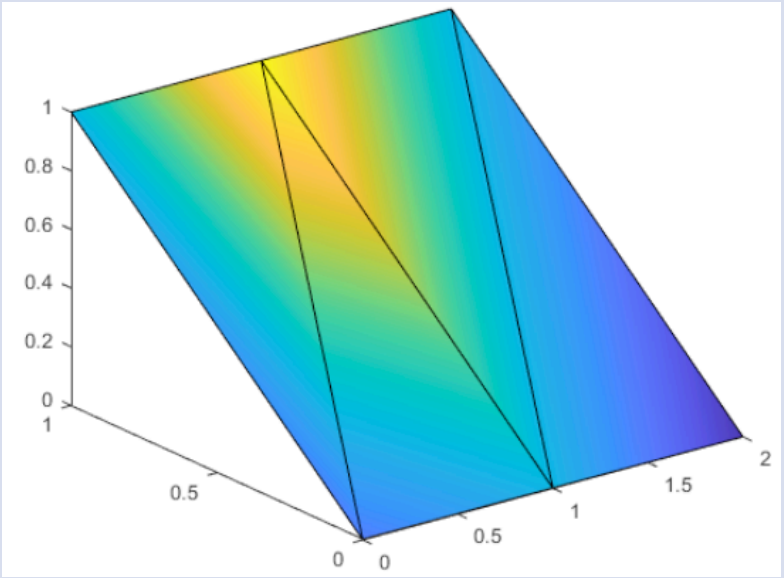
fill3

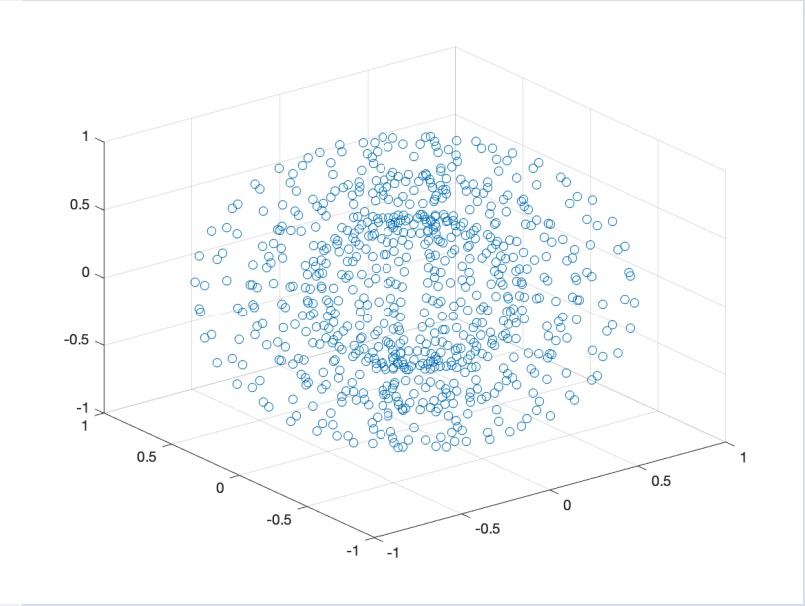
Create a flat-shaded and gouraud-shaded polygons.

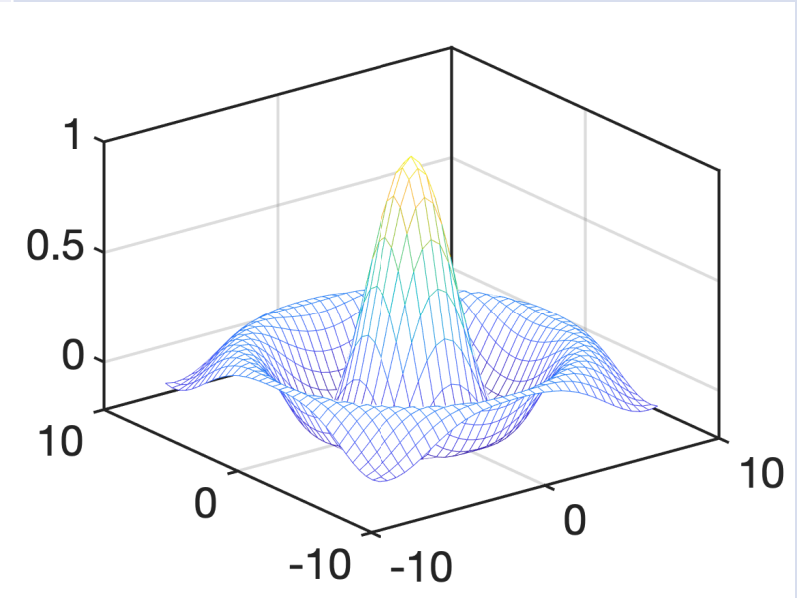
```
X = [0 1 1 2; 1 1 2 2; 0 0 1 1];
Y = [1 1 1 1; 1 0 1 0; 0 0 0 0];
Z = [1 1 1 1; 1 0 1 0; 0 0 0 0];
C = [0.5000 1.0000
    1.0000 0.5000 0.
    0.3330 0.3330 0.
>> fill3(X,Y,Z,C)
```

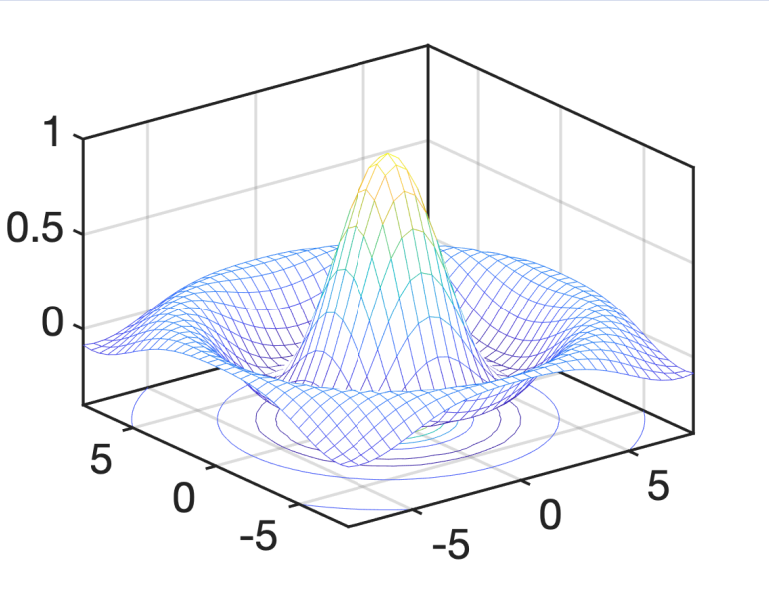
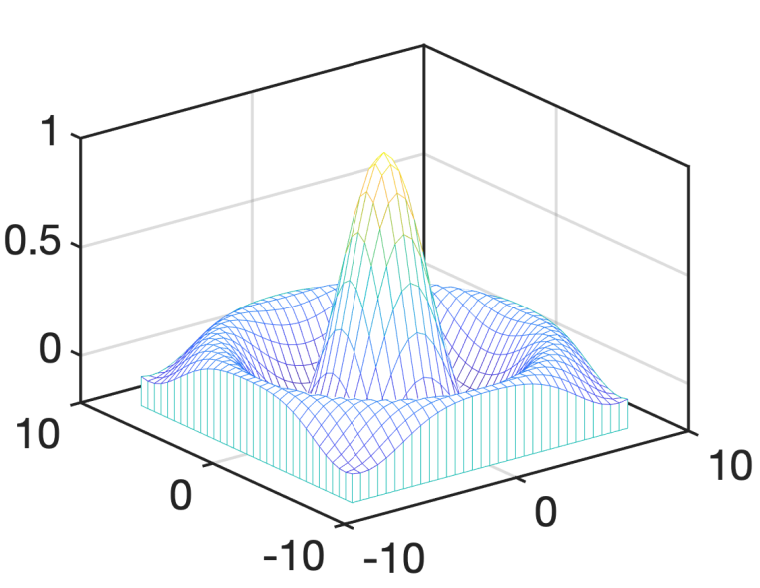


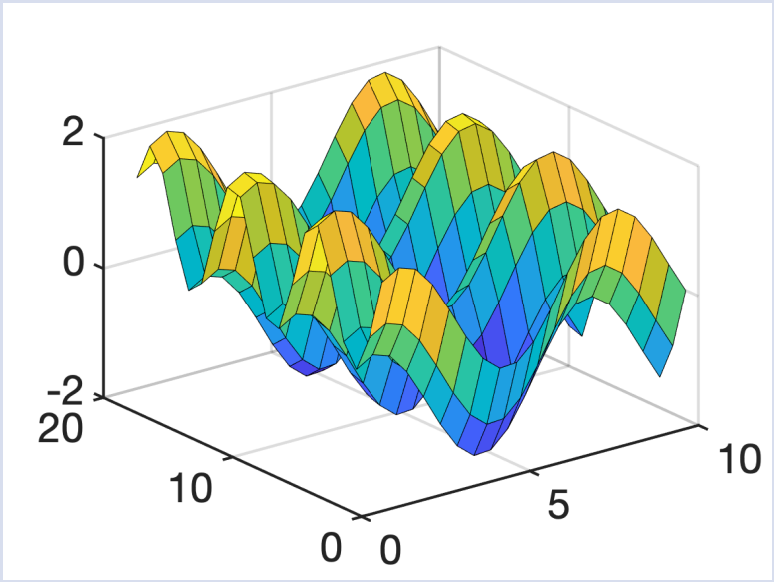
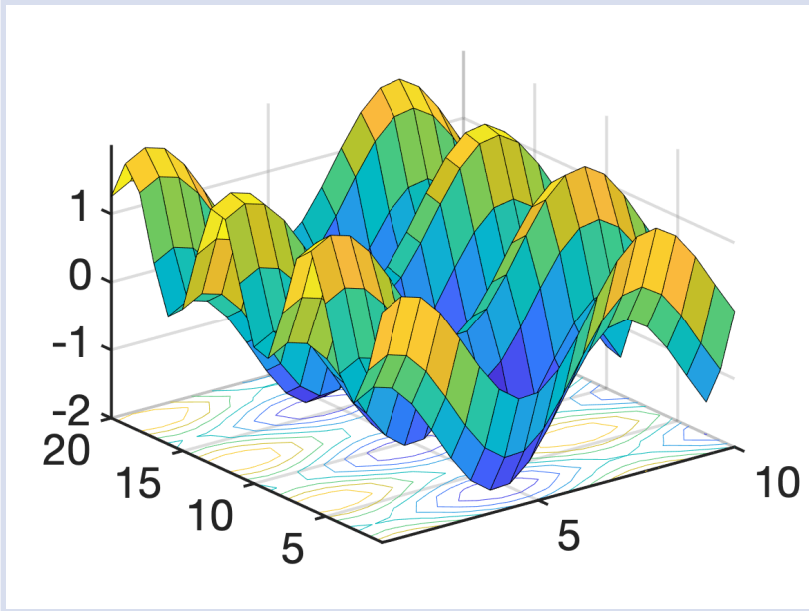
Comparison of flat shading and Gouraud shading.

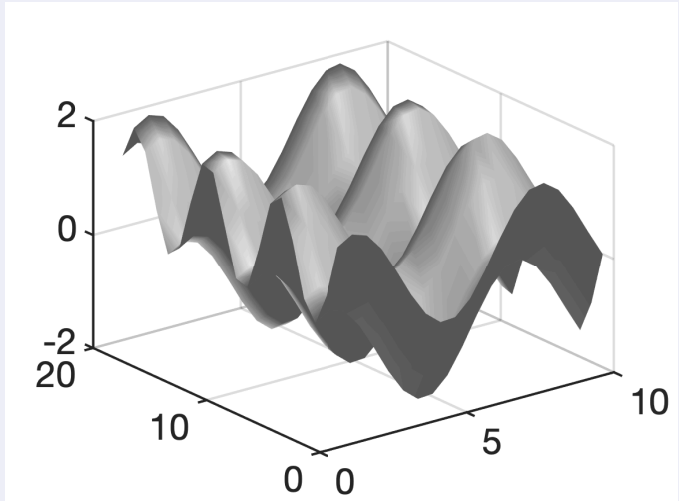
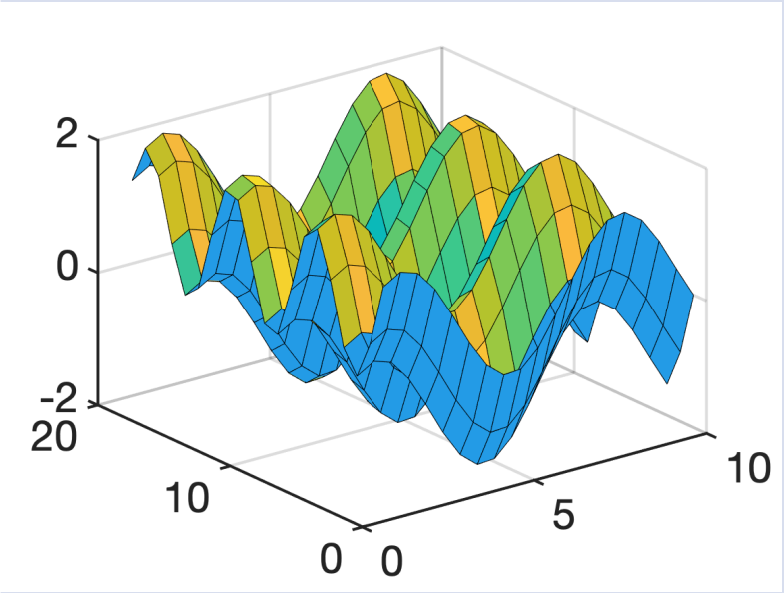
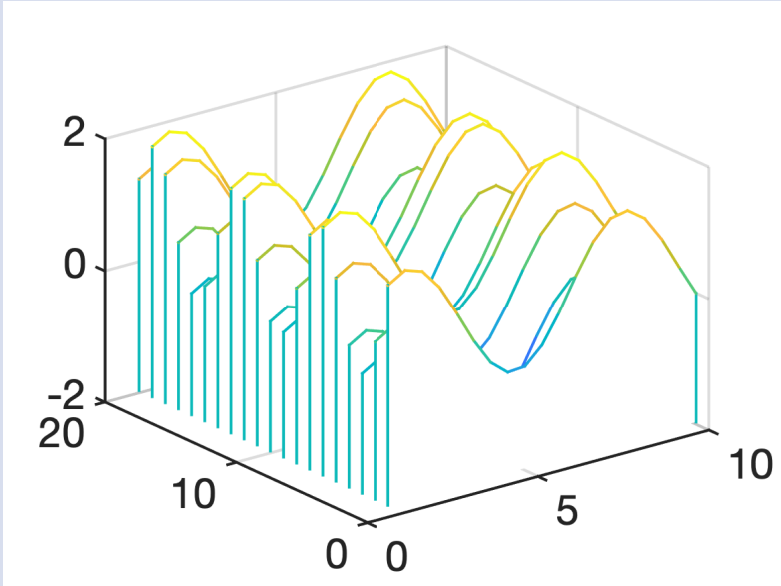


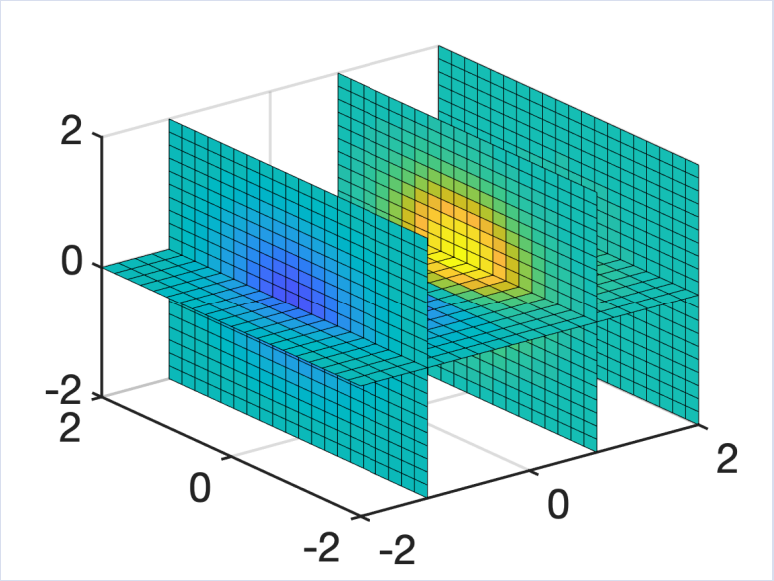
scatter3
 	
creates a 3d scatter plot
	[X,Y,Z] = sphere(16);
	x = [0.5*X(:); 0.75*X(:); X(:)];
	y = [0.5*Y(:); 0.75*Y(:); Y(:)];
	z = [0.5*Z(:); 0.75*Z(:); Z(:)];
	scatter3(x,y,z)
		

mesh *Create* a mesh plot	[X,Y] = meshgrid(-8:.5:8); R = sqrt(X.^2 + Y.^2) + eps; Z = sin(R)./R; mesh(X,Y,Z)	

meshc *creates a mesh plot* with a contour plot underneath	[X,Y] = meshgrid(-8:.5:8); R = sqrt(X.^2 + Y.^2) + eps; Z = sin(R)./R; meshc(X,Y,Z)	
meshz *creates a mesh plot* with a curtain around it	[X,Y] = meshgrid(-8:.5:8); R = sqrt(X.^2 + Y.^2) + eps; Z = sin(R)./R; meshz(X,Y,Z)	

surf *creates a 3d* surface plot	[X,Y] = meshgrid(1:0.5:10,1:20); Z = sin(X) + cos(Y); surf(X,Y,Z)	
surfc *creates a surface plot* with a contour plot underneath	[X,Y] = meshgrid(1:0.5:10,1:20); Z = sin(X) + cos(Y); surfc(X,Y,Z)	

surfl *creates a* surface plot with highlights from a light source	[X,Y] = meshgrid(1:0.5:10,1:20); Z = sin(X) + cos(Y); surfc(X,Y,Z) shading interp colormap(gray);	 
waterfall *Create a* waterfall plot, which is a mesh plot with a partial curtain along the *y* dimension	[X,Y] = meshgrid(1:0.5:10,1:20); Z = sin(X) + cos(Y); waterfall(X,Y,Z)	

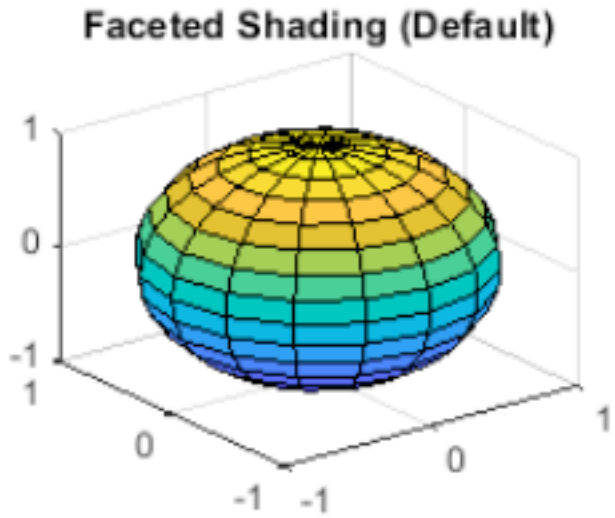
slice
 	
draws slices for the volumetric data V
	[X,Y,Z] = meshgrid(-2:.2:2);
	V = X.*exp(-X.^2-Y.^2-Z.^2);
	
	xslice = [-1.2,0.8,2];
	yslice = [];
	zslice = 0;
	slice(X,Y,Z,V,xslice,yslice,zslice)
	
	Do not create any slice planes that are orthogonal to the *y*-axis by specifying an empty array.
		

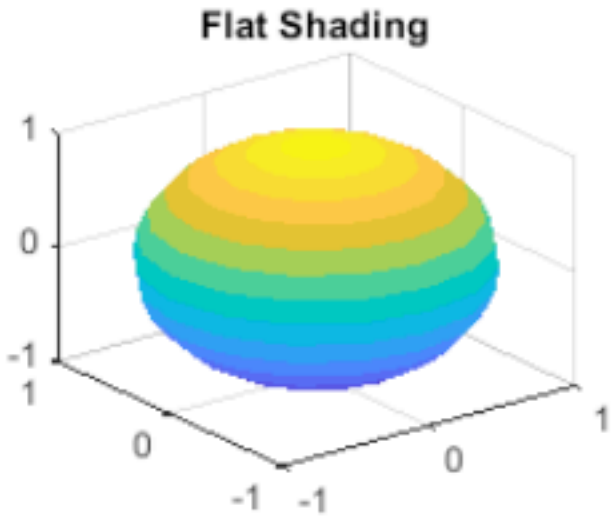
shading

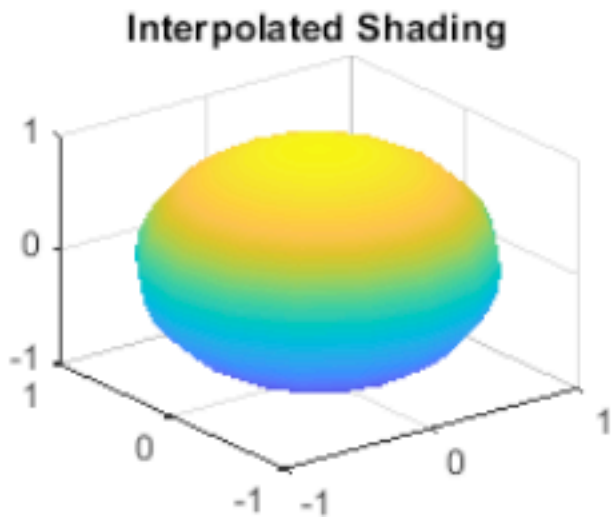
Set color shading properties

Syntax

```
shading flat
shading faceted
shading interp
shading(axes_handle,...)
```





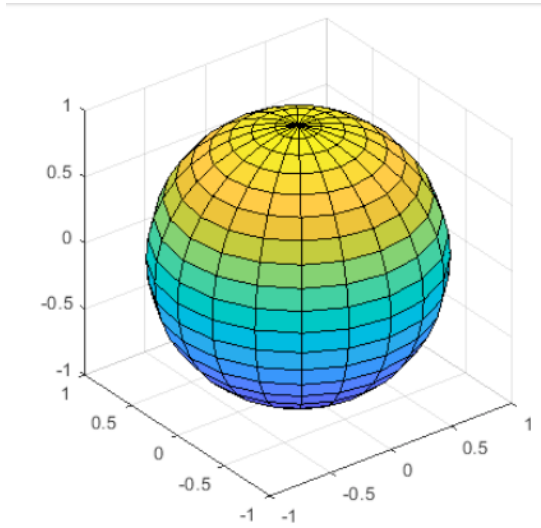


lighting

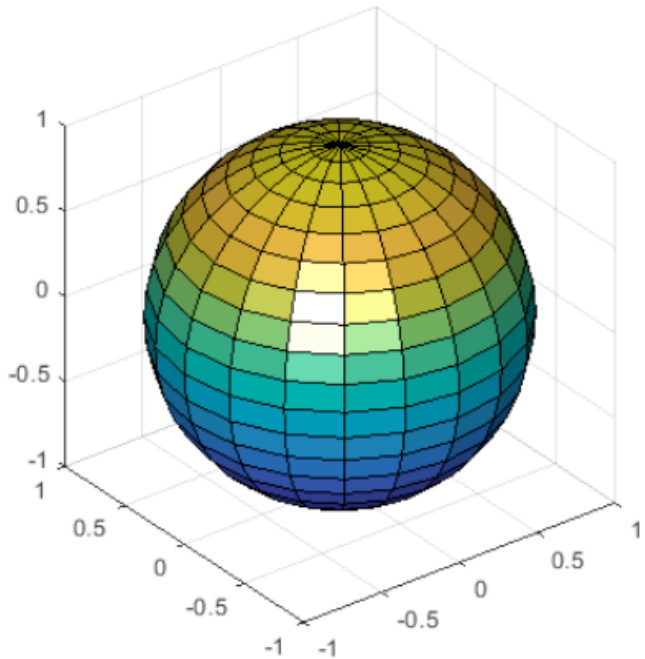
Specify lighting algorithm

Syntax

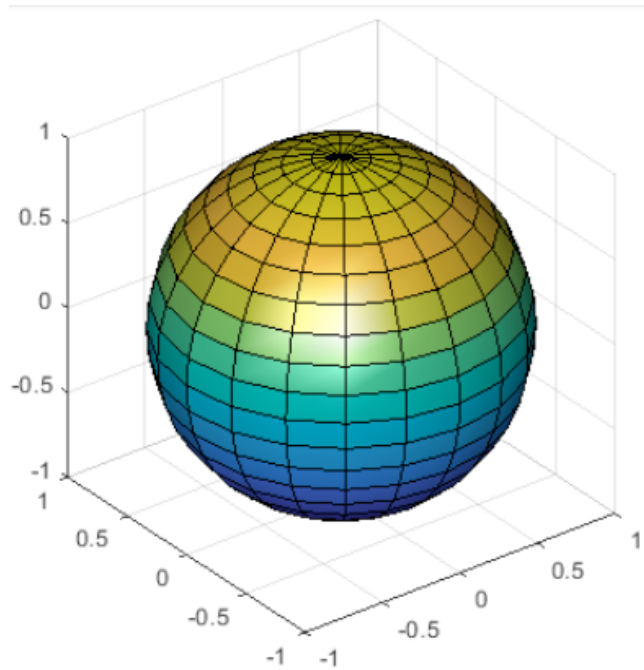
```
lighting flat
lighting gouraud
lighting none
lighting(ax,...)
```



lightangle(gca,-45,30)



lighting gouraud



Exercises

Make a three-dimensional graph of the parametric curve

$$x(t) = \frac{\cos(\omega t)}{\sqrt{1+a^2t^2}}, y(t) = \frac{\sin(\omega t)}{\sqrt{1+a^2t^2}}, z(t) = \frac{-at}{\sqrt{1+a^2t^2}},$$

for $a = 0.2$ and $\omega = 2$ and $-12 \leq t \leq 12$.

$$x(t) = a\cos(\omega t), \qquad y(t) = a\sin(\omega t), \qquad z(t) = bt,$$

with $a = 1$, $b = 0.1$ and $\omega = 2$, for $0 \leq t \leq 12\pi$.

Exercises

1. Read and reproduce the line plots in the Matlab Primer starting on page 1-17. Do not do the 3D plots now.
2. Generate a grid of equally spaced x values over the interval 0 <= x <= 5 and make line plots of the functions x, x^3, e^x all on the same graph (using hold on and hold off). After you plot the first function, you should dock the Figure window. (Try docking it and undocking it so that you know how to.)
3. Re-plot the three functions, this time with both lines and points. Try using a grid with only 6 points in x and a grid with 101 points in x.
4. Again make line plots of the functions x, x^3, e^x over the interval 0 <= x <= 5, but this time on
    1 on semilog axes (logarithm on the y-axis)
    2 on log-log axes

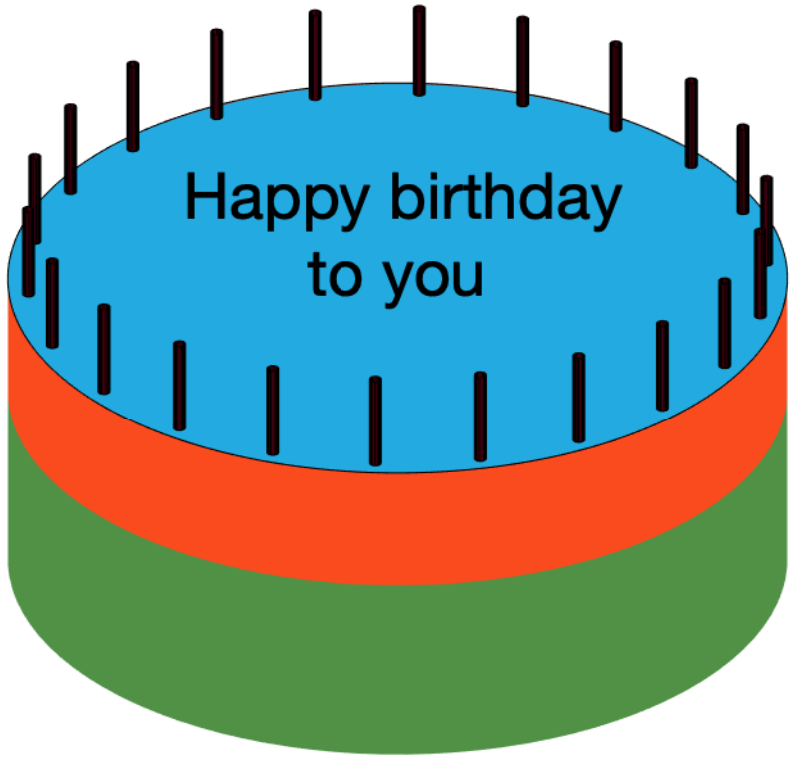
On the log-log plot, two of the curves are straight lines. What are their slopes and why?

5. Generate a vector of 100 uniformly distributed random numbers. (Hint, you need to use the Matlab function rand() with TWO arguments. By default rand() generates a matrix of random numbers. Read the Help.) Now plot a histogram of the distribution. (Use Help to learn how to plot a histogram.) Do the same for 1000 and 10,000 uniformly distributed random numbers. Do this for different number of bins.
6. Repeat question 5 for normally distributed random numbers. First use random numbers from the standard distribution, then random numbers from a normal distribution with mean 4 and variance 9. There are Matlab functions that will compute the mean and variance of a vector. Use Help to find these and compute the mean and variance of your distributions. Plot vertical lines at the mean and at the mean plus and minus one standard deviation.

http://homepages.warwick.ac.uk/staff/D.Barkley/Teaching/Matlab/Tutorial/plottingexercises.html

Fun with matlab

Birthday cake




```
[x_y y_y z_y] = cylinder( 1 , 100 );
[x_r y_r z_r] = cylinder( 1 , 100 );
surf(x_y,y_y,z_y / 3 + 0.5 , 'facecolor' ,
[ 252 / 255  77  / 255   34  / 255 ],
'linestyle' , 'none' );  hold on
surf(x_r,y_r,z_r / 2 , 'facecolor' ,[ 81 / 255
149 / 255  72 / 255 ], 'linestyle' , 'none' );
z_y_max = max (z_y / 3 + 0.5 );
z_y_max = z_y_max( 1 );
x_top = cos(linspace( 0 , 2 * pi, 1000 ));
y_top = sin(linspace( 0 , 2 * pi, 1000 ));
z_top = ones(size(x_top)) * z_y_max;

fill3(x_top,y_top,z_top, 'y' ,  'facecolor' ,
[ 38 / 255  173 / 255  228 / 255 ]);
for  i = 1 : 22
    [x_can y_can z_can] = cylinder( 0.015 ,
22 );
z_can = z_can / 4 + .8333 ;
x_can = x_can + 0.95 * cos(i * pi / 22 * 2 );
y_can = y_can + 0.95 * sin(i * pi / 22 * 2 );
    surf(x_can,y_can,z_can, 'facecolor' ,[ 250 /
255  2 / 255   60 / 255 ]); axis equal

end
```

https://www.programmersought.com/article/88662245440/

Lesson 23: Vector field and volumetric plots

Learning objectives

To learn how to create vector field and volumetric plots in MATLAB.

- One of the most crucial needs of visualization in scientific computation is for data that is essentially volumetric, i.e., defined over a 3-D space.
- For example, we may have temperature or pressure defined over each (x, y ,z) triple in a bounded 3-D space.
- How do we display this data graphically?
- If we have a function z = f(x, y) defined over a finite region of the xy-plane, we can display as a 3D surface plot.
- But we have f(x, y, z)! So, we need a 4-D hypersurface.

- We display volumetric data by slicing it along several planes in 3-D and plotting the data on those planes, either with graded color maps or with contours.

- One such visualization is making vector fields in 3D.
- Visualization of vector fields include

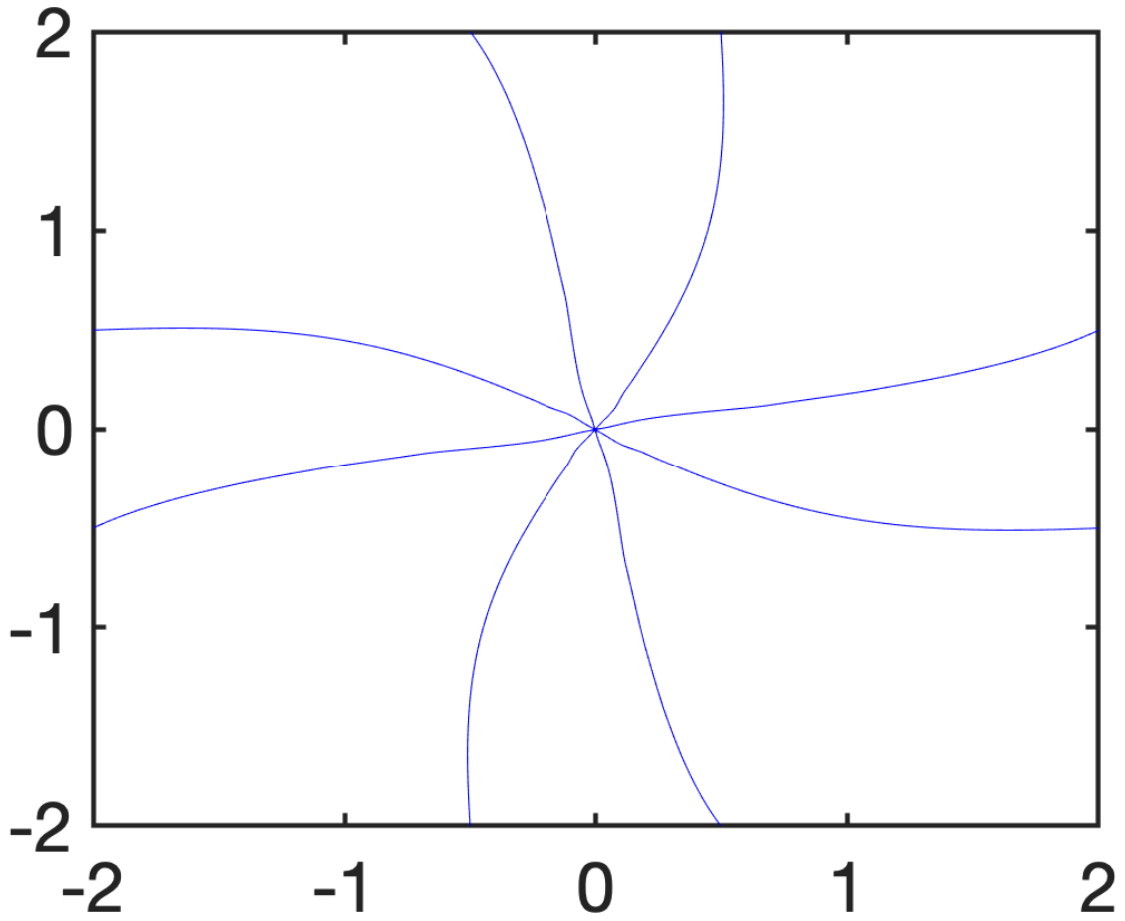
  - ✓ quiver,
  - ✓ quiver3,
  - ✓ stream2,
  - ✓ stream3,
  - ✓ streamline,
  - ✓ streamtube,
  - ✓ streamribbon,
  - ✓ streamslice,
  - ✓ streamparticles,
  - ✓ coneplot,
  - ✓ divergence,
  - ✓ curl

$$\begin{aligned} \dot{x} &= y + x - x(x^2 + y^2) \quad \text{and} \\ \dot{y} &= -x + y - y(x^2 + y^2). \end{aligned}$$

These two ODEs define a vector field ($u \equiv \dot{x}$ and $v \equiv \dot{y}$). Let us use `streaml`: draw a few solution trajectories starting from various points in the xy-plane (conditions in the phase plane). The general syntax of `streamline` is

```
streamline(x,y,z, u,v,w, x0,y0,z0)
```

```
v = linspace(-2,2,50) ;
[X,Y] = meshgrid(v) ;
U= Y + X - X.*(X.^-2 + Y.^-2);
V= Y-X-Y.*(X.^-2+Y.^-2);

x0=[-2 -2 -2 -2 -.5 -.5 .5 .5
2 2 2 -.01 -.01 .01 .01];

y0= [-2 -.5 .5 2 -2 2 -2 2 -2
5 .5 2 -.01 .01 -.01 .01];

streamline(X,Y,U,V,x0,y0)
```



Volumetric plots

Visualization functions for volumetric plots include

- slice,
- slicecontour,
- isosurface,
- isonormal,
- isocaps,
- isocolors,
- subvolume,
- reducevolume,
- smooth3,
- reducepath,

Lesson 23: Handle Graphics

Learning objectives

To learn how to change the properties of plots using comment line with a handle

Handle Graphics

If you want extra-detailed control of your graph appearance or want to do animation (be-yond comet plots), you might want to learn Han-dle Graphics.

This is NOT a topic for beginners.

E.g. A Graphics object: **Line**

Properties

- line style,
- color,
- thickness,
- visibility

It is possible to change any of its properties later

- Suppose you draw several lines with pencil on a paper.
- If you want to change one of the lines, you must first find the line you want to change and then change what you do not like about it.
- On the graphics screen, a line may be one among several graphics objects (e.g., axes, text).
- So how do you get hold of a line? Answer is by its handle

What is a handle?

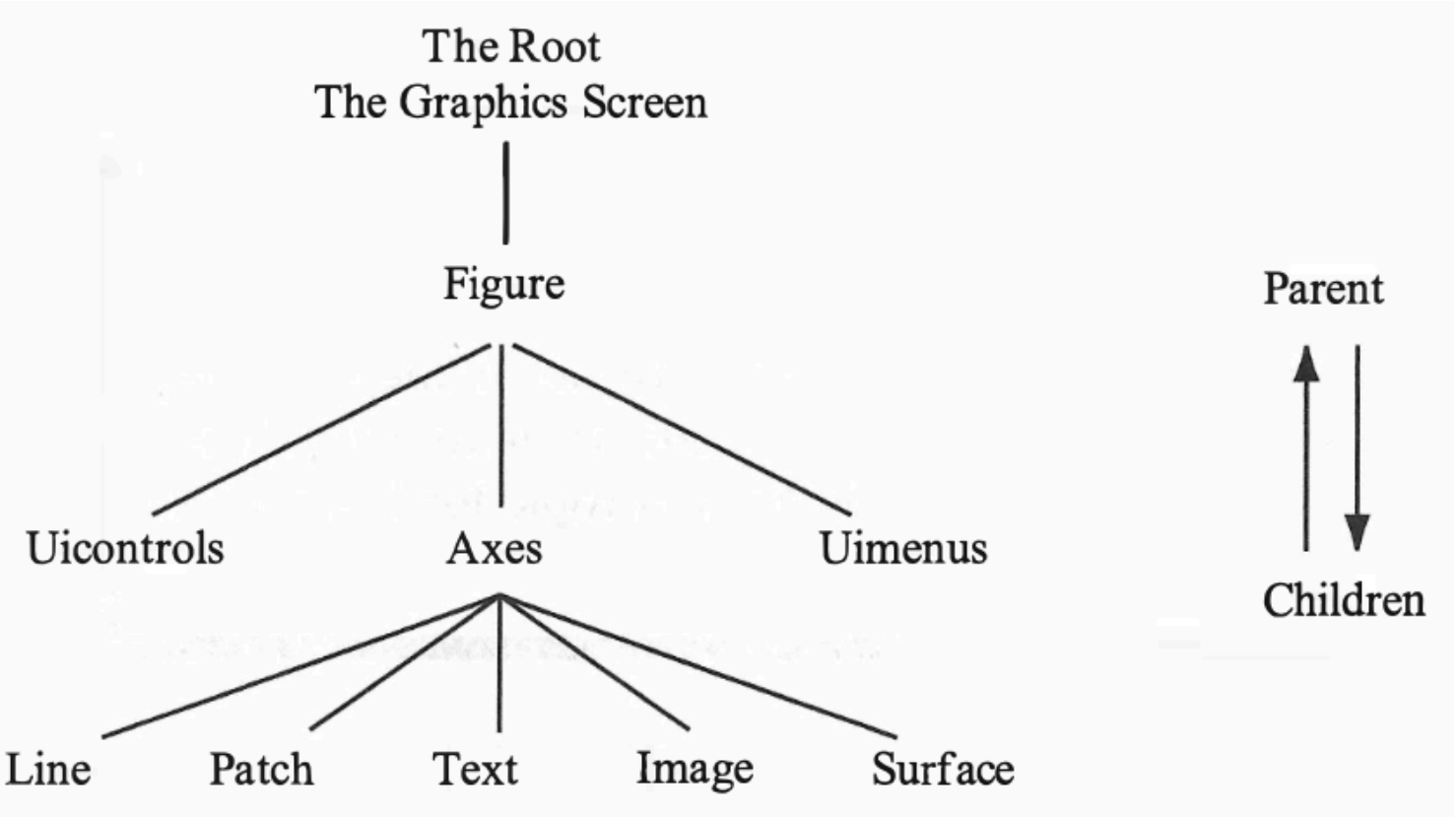
- MATLAB assigns a floating-point number to every object in the figure window (including invisible objects).
- It uses this number (handle of the object) as an address or name for the object in the figure.
- Once you get hold of the handle, you can access all properties of the object.
- Handle →object→list of properties
- In programming, this approach of defining objects and their properties is called object-oriented programming.
- Using this, you can access individual objects and their properties and change any property of an object without affecting other properties or objects.
- MATLAB's entire system of object-oriented graphics and its user controlability is **Handle Graphics**.

What we are going to learn is

- to know how to get the handles of graphics objects.
- to know how to use handles to get and change properties of the objects.

- Not all graphics objects are independent (for example, the appearance of a line depends on the current axes in use) , and a certain property of one may affect the properties of the others.
- to know how the objects are related is therefore important.

Graphics objects follow a hierarchy of parent-child relationship.



- It shows which objects will be affected if you change a default property value at a particular level.
- It tells you at which level you can query for handles of which objects.

Getting object handles (→ unique identifiers associated with each graphics object)

Two ways:

1. By creating handles explicitly at the **object-creation-level commands** (that is, you can make a plot and get its handle at the same time):

   hl = plot(x,y, 'r-') returns the handle of the line to hl, and hxl = xlabel (' Angle ‘)

   returns the handle of the x-label to **hxl**.

2. By using explicit handle-returning functions:

gcf gets the handle of the current figure.

Example: hfig = gcf;

returns the handle of the current figure in hfig.

gca gets the handle of the current axes

Example: haxes = gca;

returns the handle of the current axes in haxes.

gco gets the handle of the current object.

get(h) to get a list of all property names and their current values for an object with handle h

Handles of other objects, in turn, can be obtained with the get command.

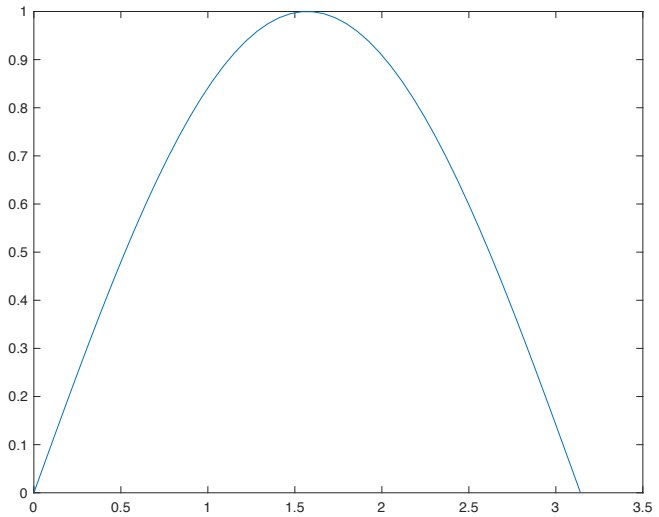
For example,

hlines=get(gca,'children’) the handles of all children of the current axes in a column vector hlines.

```
>> whos
  Name       Size          Bytes  Class                                  Attributes

  hl         1x1               8  matlab.graphics.chart.primitive.Line
  hlines     1x1               8  matlab.graphics.chart.primitive.Line
  t          1x50            400  double
```

Method 1

```
t= linspace(0,pi,50);
hl =plot(t,sin(t))

hl =

  Line with properties:

          Color: [0 0.4470 0.7410]
      LineStyle: '-'
      LineWidth: 0.5000
         Marker: 'none'
     MarkerSize: 6
   MarkerFaceColor: 'none'
          XData: [1×50 double]
          YData: [1×50 double]
          ZData: [1×0 double]

  Show all properties
```



Method 2

```
>> plot(t,sin(t));
>> hlines=get(gca,'children')
hlines =

  Line with properties:

          Color: [0 0.4470 0.7410]
      LineStyle: '-'
      LineWidth: 0.5000
         Marker: 'none'
     MarkerSize: 6
   MarkerFaceColor: 'none'
          XData: [1×50 double]
          YData: [1×50 double]
          ZData: [1×0 double]

  Show all properties
```

Object properties

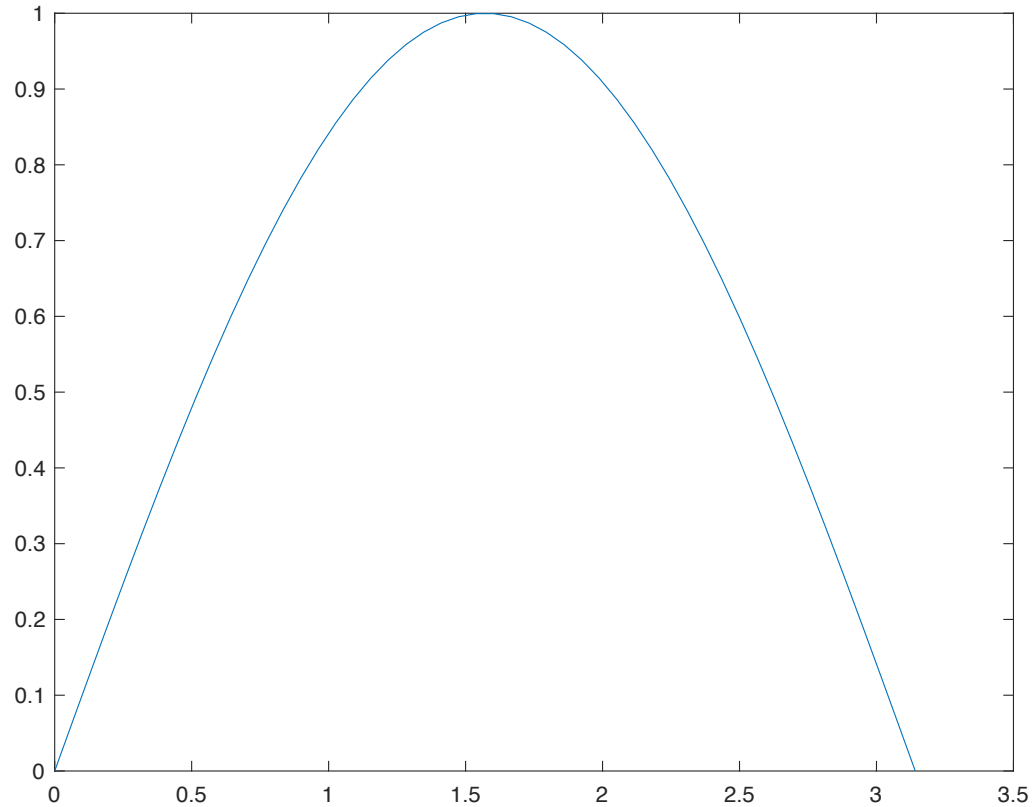
- Every graphics object on the screen has certain properties associated with it,
- E.g. properties of a line, type, parent, visible, color, linestyle, linewidth, xdata, ydata, etc.
- properties of a text object, such as xlabel or title, type, parent, visible, color, fontname, fontsize, fontweight, string, etc.
- Once the handle of an object is known, you can see the list of its properties and their current values with the command get(handle) .
- There are some properties common to all graphics objects. These properties are children, clipping, parent, type, userdata, and visible.

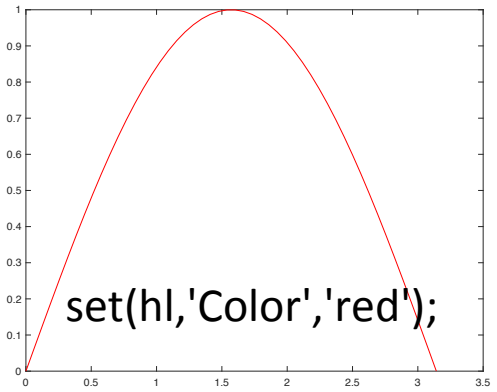
Setting property values

Any property can be changed by the command set(handle, 'PropertyName', 'PropertyVaLue') where PropertyValue may be a character string or a number.

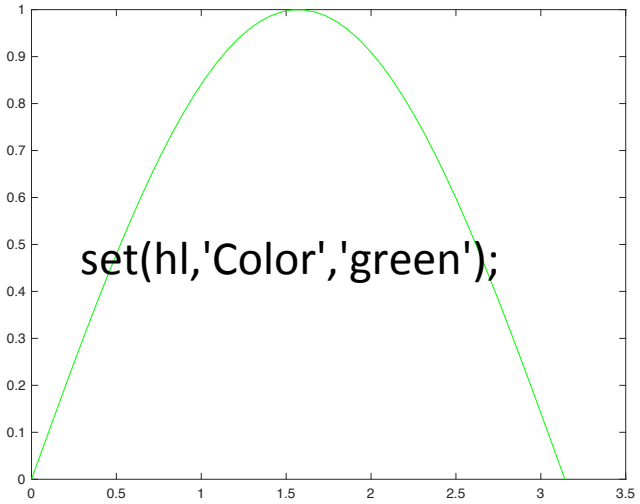
- If PropertyValue is a string, then it must be enclosed within single quotes.

```
t= linspace(0,pi,50);
hl =plot(t,sin(t))

hl =

  Line with properties:

            Color: [0 0.4470 0.7410]
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
       MarkerSize: 6
  MarkerFaceColor: 'none'
            XData: [1×50 double]
            YData: [1×50 double]
            ZData: [1×0 double]

  Show all properties
```



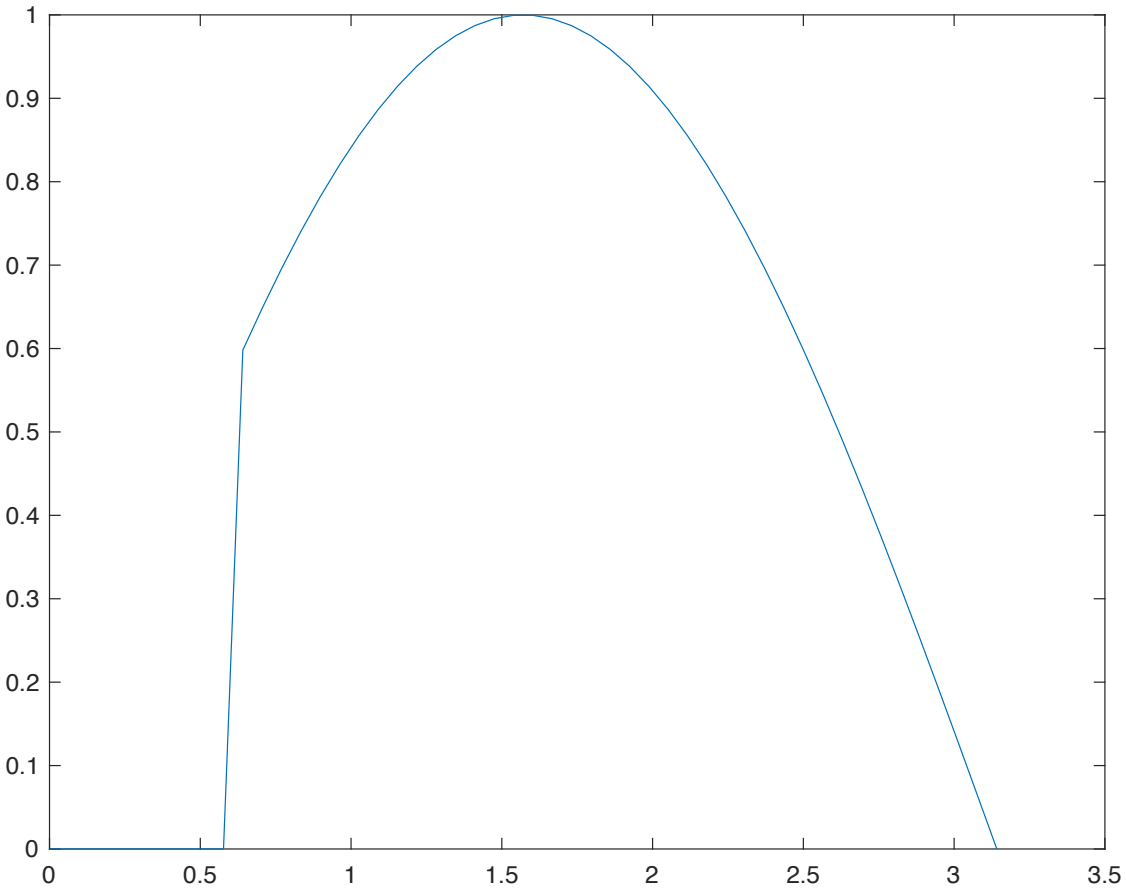


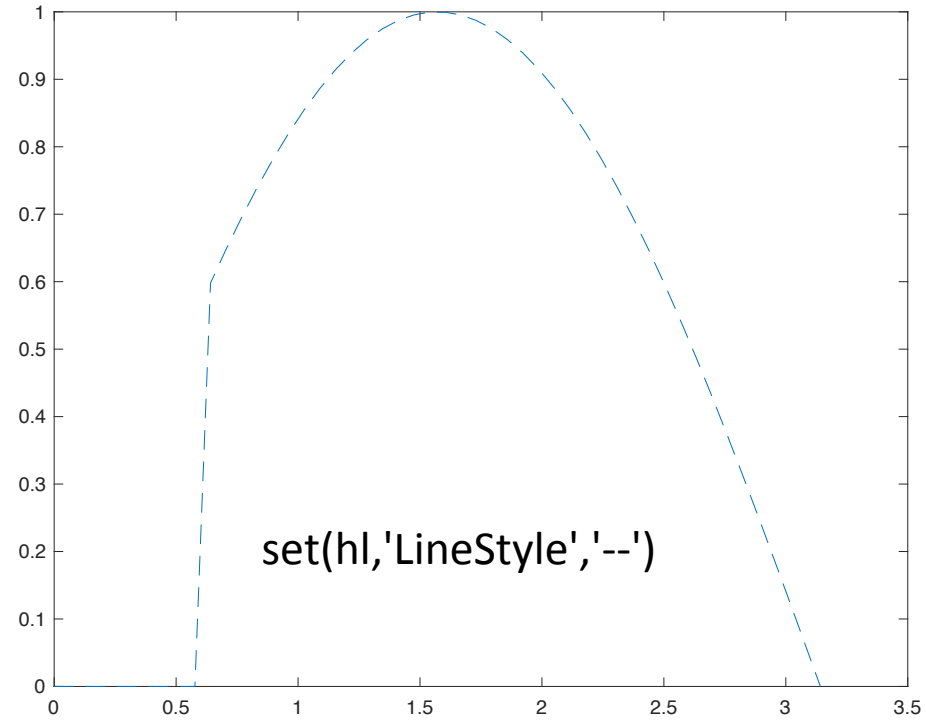

```
hl =

  Line with properties:

            Color: [1 0 0]
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
       MarkerSize: 6
    MarkerFaceColor: 'none'
            XData: [1×50 double]
            YData: [1×50 double]
            ZData: [1×0 double]
```




```
hl =

  Line with properties:

            Color: [0 1 0]
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
       MarkerSize: 6
    MarkerFaceColor: 'none'
            XData: [1×50 double]
            YData: [1×50 double]
            ZData: [1×0 double]

  Show all properties
```

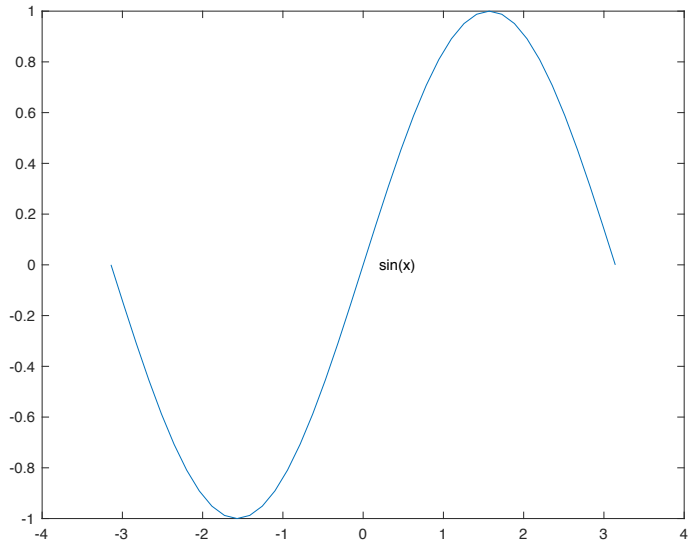
```
yvec=get(hl,'YData');
yvec(1:10)=0;
set(hl,'YData',yvec)
```





Changing text object

```
x = -pi:pi/20:pi;
y = sin(x);

f = figure;
p = plot(x,y);
txt1 = text(0.2,0,'sin(x)');
```

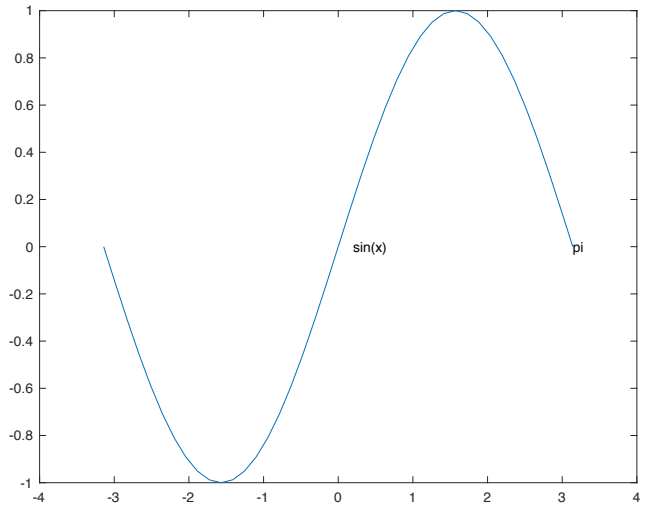


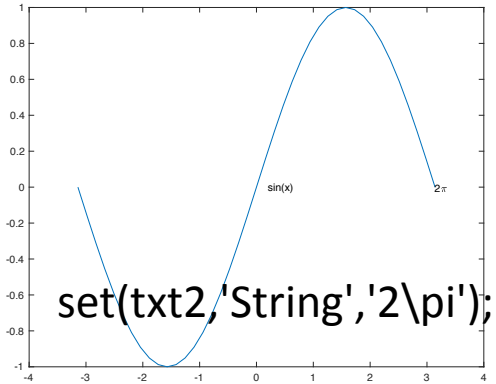
Line object

```
p =

  Line with properties:

            Color: [0 0.4470 0.7410]
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
       MarkerSize: 6
  MarkerFaceColor: 'none'
            XData: [1×41 double]
            YData: [1×41 double]
            ZData: [1×0 double]

  Show all properties
```

Text object

```
txt2 = text(x(end), y(end), 'pi')
txt2 =

  Text (pi) with properties:

              String: 'pi'
            FontSize: 10
          FontWeight: 'normal'
            FontName: 'Helvetica'
               Color: [0 0 0]
  HorizontalAlignment: 'left'
            Position: [3.1416 1.2246e-16 0]
               Units: 'data'

  Show all properties
```



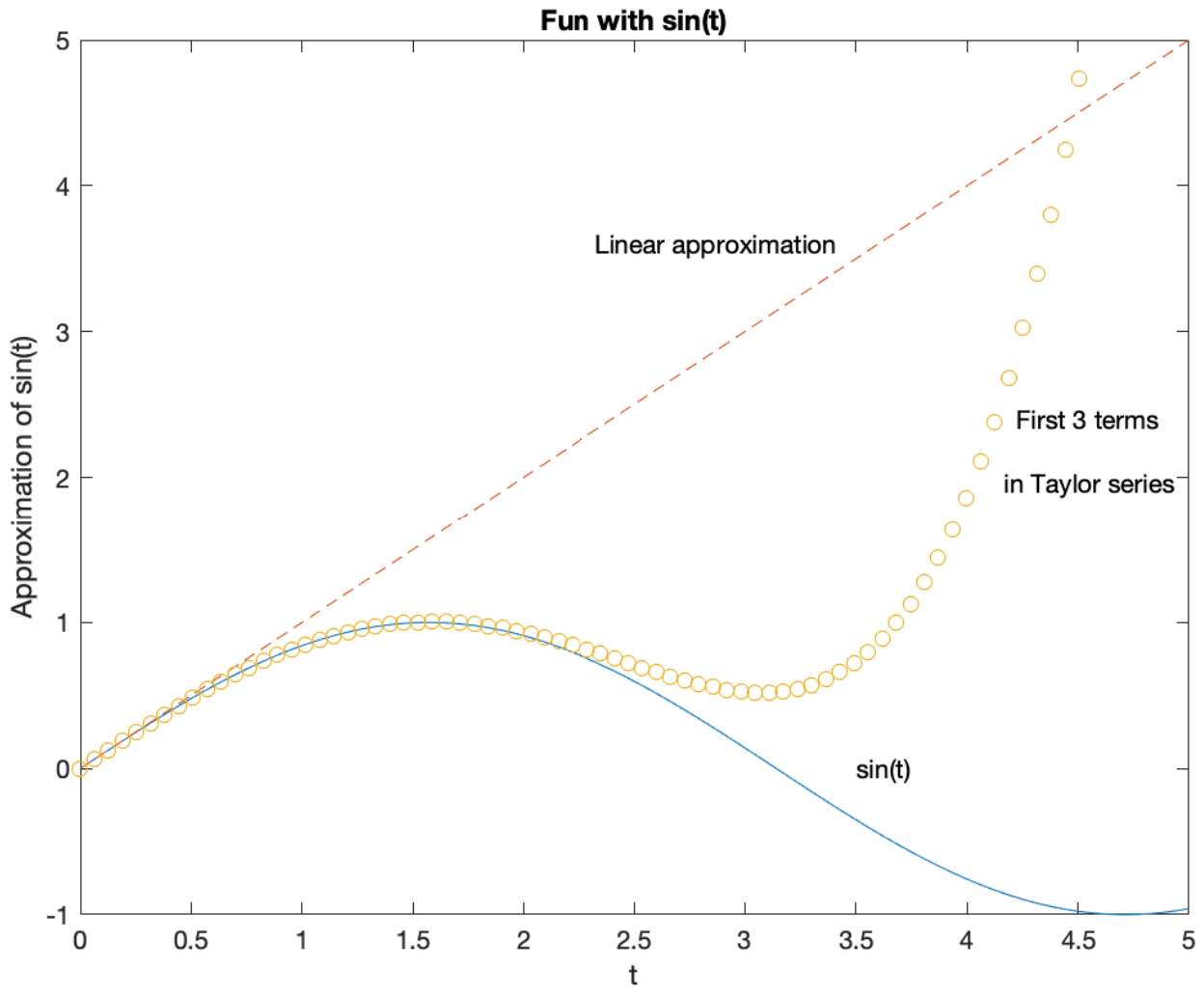



```
set(txt2,'String','2\pi');
```

Handle as a vector

Example:

```
t=linspace(0,2*pi,100);
y1= sin(t);
y2 = t;
 y3 =  t - t.^3/6 + t.^5/120;
 plot(t,y1,t,y2,'--',t,y3,'o');
 axis([0 5 -1 5]);
xlabel('t');
ylabel('Approximation of sin(t)')
title('Fun with sin(t)');
text(3.5,0,'sin(t)')
gtext('Linear approximation')
gtext('First 3 terms')
gtext('in Taylor series')
```



Get Handle

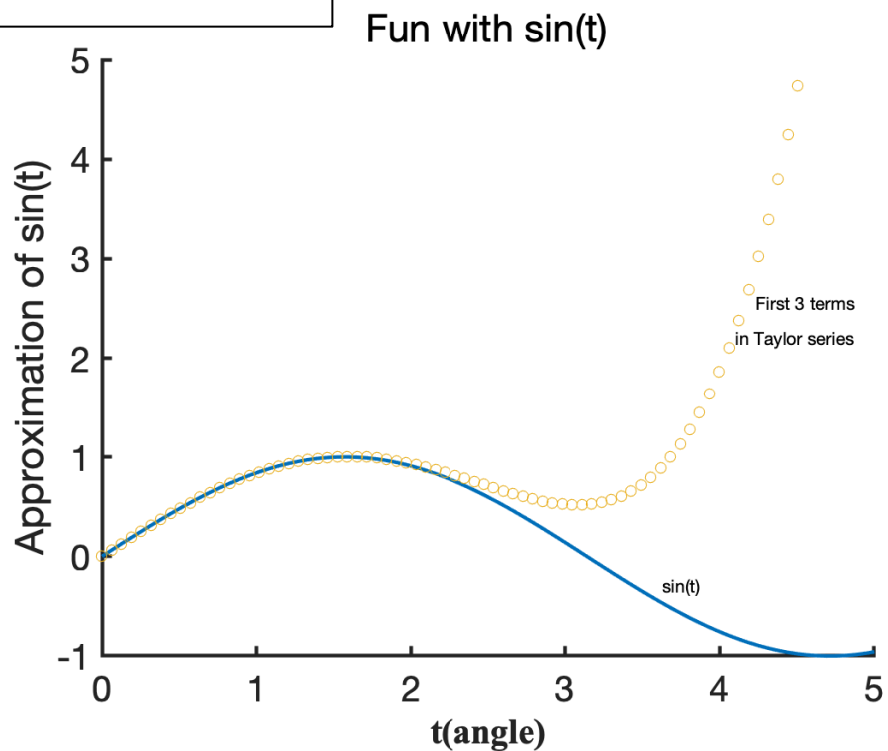
```
h=gca;
set(h,'box','on');
set(h,'box','off');
set(h,'linewidth',2,'fontsize',24);
hlines=get(h,'children');
```

vector

```
hlines =
  7×1 graphics array:
  Text    (in Taylor series)
  Text    (First 3 terms)
  Text    (Linear approximation)
  Text    (sin(t))
  Line
  Line
  Line
```

Set Handle

```
set(hlines(7),'linewidth',2);
set(hlines(6),'visible','off');
delete(hlines(3))
hx1=get(h,'xlabel');
set(hx1,'string','t(angle)')
set(hx1,'fontname','times')
set(hx1,'fontsize',20,'fontweight','bold')
ht=get(h,'title');
set(ht,'FontSize',30)
```



Understanding a vector of handles

```
hlines=get(gca,'children’);
hx1=get(gca,'xlabel’);
ht=get(gca,'title’);
```

The command, hlines=get (h , 'children') , gets the handles of all the children of the current axes (specified by handle h) in a column vector hlines.

The vector hlines have seven element---three lines and four texts.

how do we know which handle is for which line or which text?

get (hline(i) , ' type ') lists the type of the object whose handle is hline(i).

```
>> get(hlines(1),'type’)

ans =

    'text'

>> get(hlines(6),'type')

ans =

    'line'
```

```
>> get(hlines(5),'marker')

ans =

    'o'

>> set(hlines(5),'marker','s')
```

Deleting graphics objects

Any object in the graphics window can be deleted without disturbing the other objects with the command

delete(objHandLe)

where ObjHandle is the handle of the object.

E.g. set(hline(6) , 'visible' , 'off’) is the same as delete(hline (6)

Modifying plots with PropEdit

Alternatively, one can use the point-and-click graphics editor, PropEdit.

Simply type propedit to activate the editor and then edit the figure.

Summary

Command	Description
`gca`	Return handle of current axes
`gcf`	Return handle of current figure
`gco`	Return handle of current object
`get`	Query values of object's properties
`ishandle`	True if value is valid object handle
`set`	Set values of an object's properties

Saving and Printing Graphs

saveas

Save figure to specific file format

Syntax

```
saveas(fig,filename)
saveas(fig,filename,formattype)
```

```
x = [2 4 7 2 4 5 2 5 1 4];
bar(x);
saveas(gcf,'Barchart.png')
saveas(gcf,'Barchart','epsc')
```

Saving graphs to reusable files

Syntax

```
hgsave(filename)
hgsave(h,filename)
```

hgsave('Barchart.fig')

hgsave Barchart.fig

hgsave(h,'Barchart.fig')

sveas(gcf, 'Barchart' , 'fig')

savefig

Save figure and contents to FIG-file

Syntax

```
savefig(filename)
savefig(H,filename)
savefig(H,filename,'compact')
```

Create two plots and store the figure handles in array h. Save the figures to the file TwoFiguresFile.fig. Close the figures after saving them.

```
h(1) = figure;
z = peaks;
surf(z)

h(2) = figure;
plot(z)

savefig(h,'TwoFiguresFile.fig')
close(h)
```

To open the two figures, use the command:

```
figs = openfig('TwoFiguresFile.fig');
```

Save Figure Using 'compact' Option

Save a figure using the compact option:

```
h = figure
surf(peaks)
savefig(h,'PeaksFile.fig','compact')
```

To open the figure, use the command:

```
openfig('PeaksFile.fig');
```

Recreating the graph

- You can select Create M-code from the File menu of the graphics window.

- This action will create a function file, an M-file, that will require input data to recreate the graph.

- Thus, the generated M-code contains all commands necessary to create the graph but does not contain the data that was used for the graph.

- This particular way of saving a graph may be useful when you make a lot of changes to your graph using plot editor and you would like to save the final settings for creating similar plots with possibly different data.

Exercises

1 graphics placement with Handle Graphics
2. Fun with spirals
3. Editing contour plot