MATLAB Week 3

17 November 2009

Outline

- Graphics
  - Basic plotting
  - Editing plots from GUI
  - Editing plots from m-file
  - Advanced plotting commands

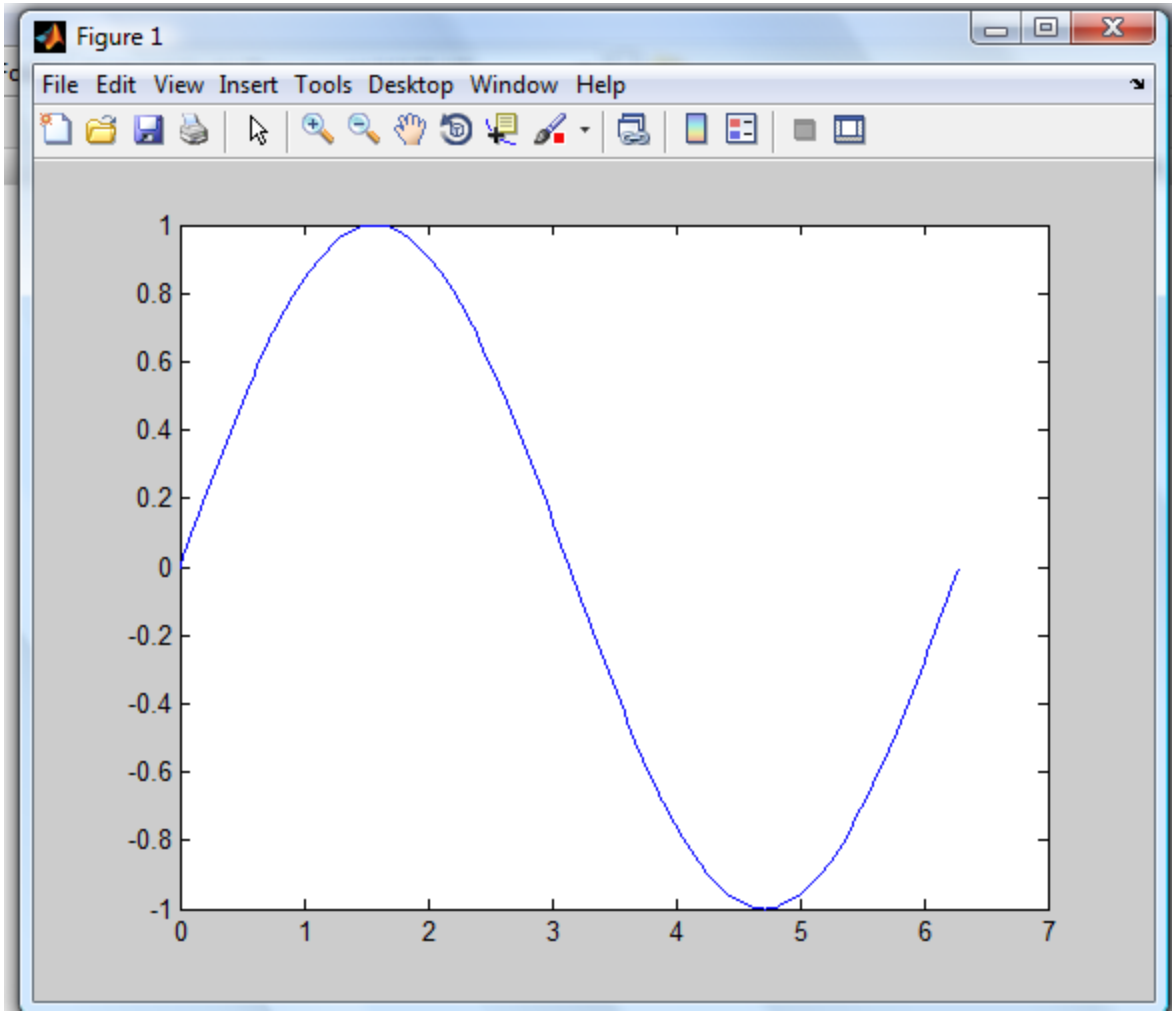
Basic Plotting

- **plot(x,y)**
  - Basic MATLAB plotting command
  - Creates figure window if not already created
  - Autoscales each axis to fit data range
  - Can add axes object properties after x,y
- **figure(n)**
  - Will create a new figure window
  - MATLAB will use current figure window by default
  - Plot command will overwrite figure if called repeatedly given same current figure window

Basic Plotting

- Create a simple plot from the command line

```
Command Window
>> t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```



Basic Plotting

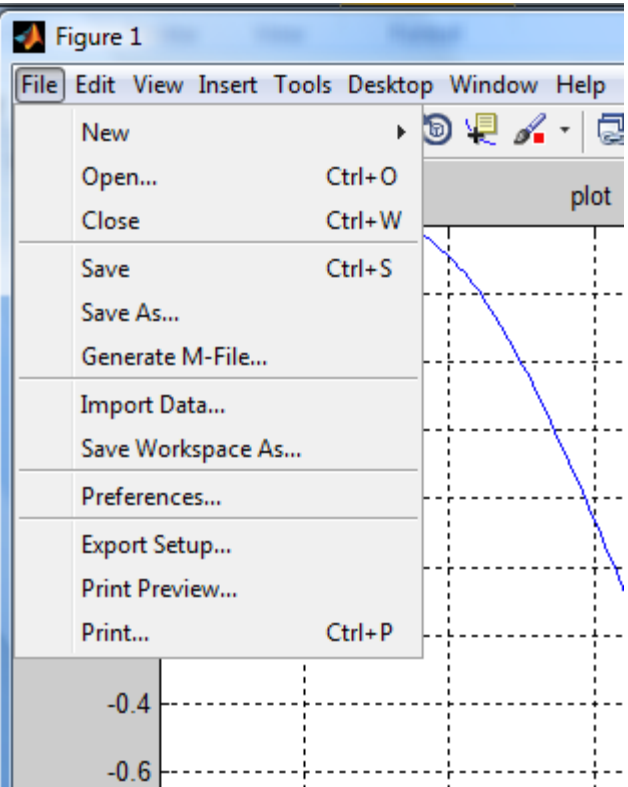
- MATLAB figures consist of objects
  - Figure, axes, lineseries (data), colorbar, etc
  - Can edit the properties of each object to modify appearance of figure
- Plot commands will generate figure and axis handles to use to modify properties in m-files
  - Discuss more later today

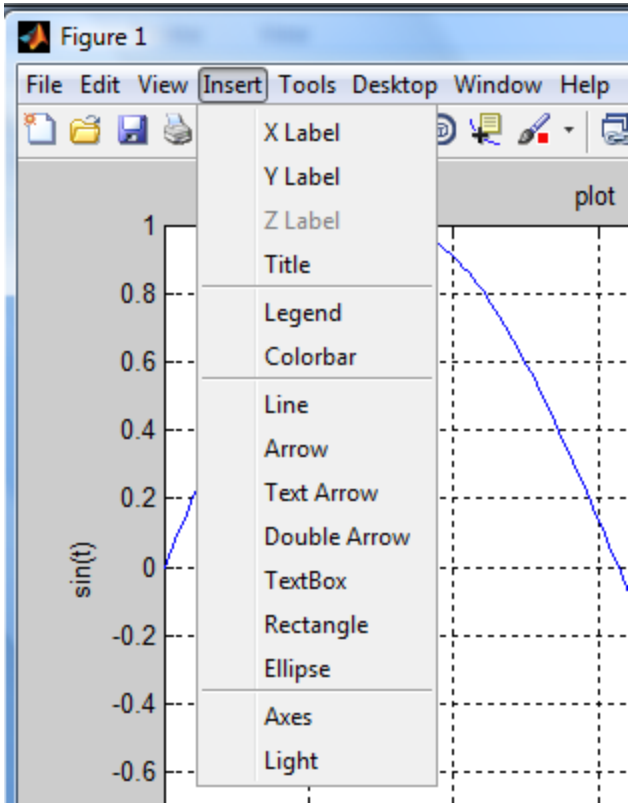
Figure Window

- All editing of figure can be done manually in figure window
- Can insert title, axis labels, legend, change axis scaling, tick marks and labels, etc
  - Essentially anything you can do from function calls you can do by hand in the figure window
    - It just takes longer and you have to do it to every figure every time it is created

Figure Window

- Four important menus on figure window
  - File, edit, insert and tools drop down menus





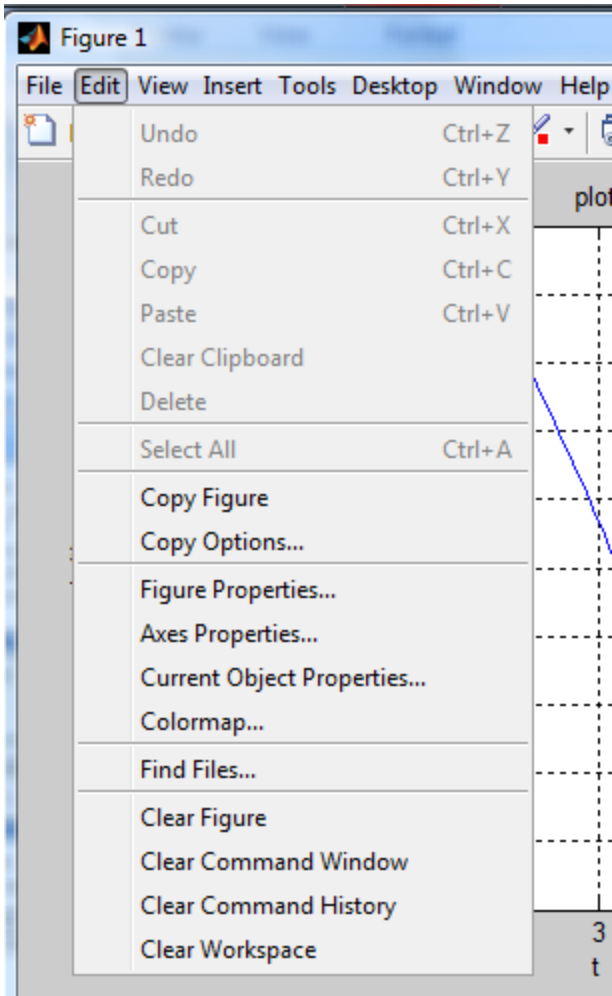


Figure Window

- File menu
  - Save, open, close, print figure
    - Can save as many different image formats: png, jpg, eps, tif, etc.
- Edit menu
  - Edit axis properties, figure properties
- Insert menu
  - Insert title, legend, axis labels
- Tools menu
  - Change view of plot, add basic regression lines

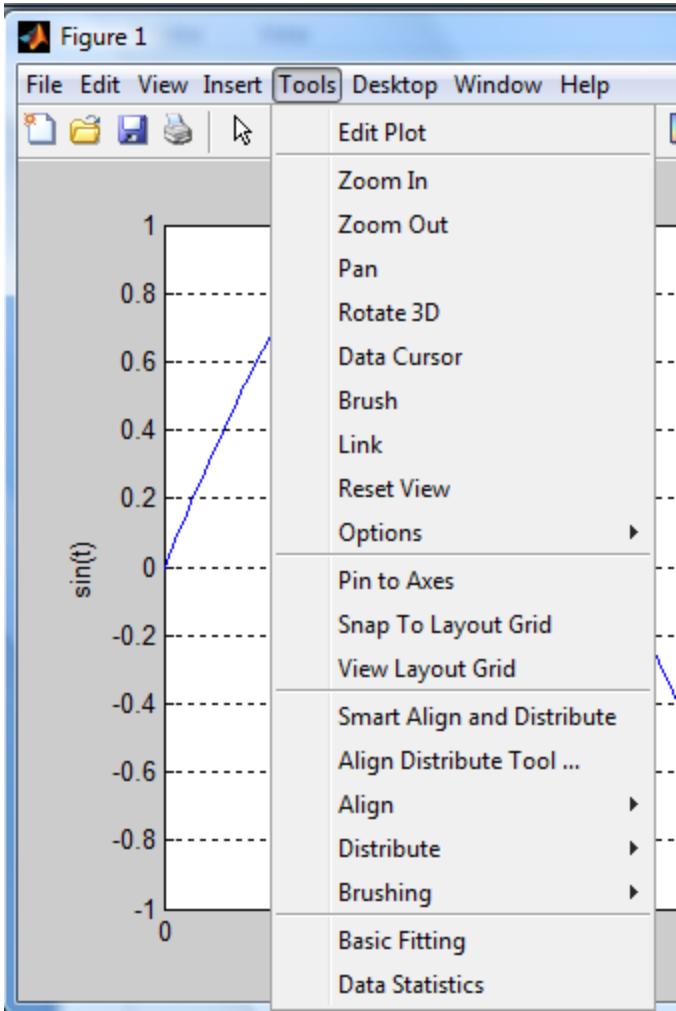


Figure Window

- Top toolbar
  - From left to right:

    New figure, open file, save figure, print figure, edit figure, zoom in , zoom out, pan, rotate 3D, data cursor, brush/select data, link plot, insert colorbar, insert legend, hide plot tools, show plot tools and dock figure

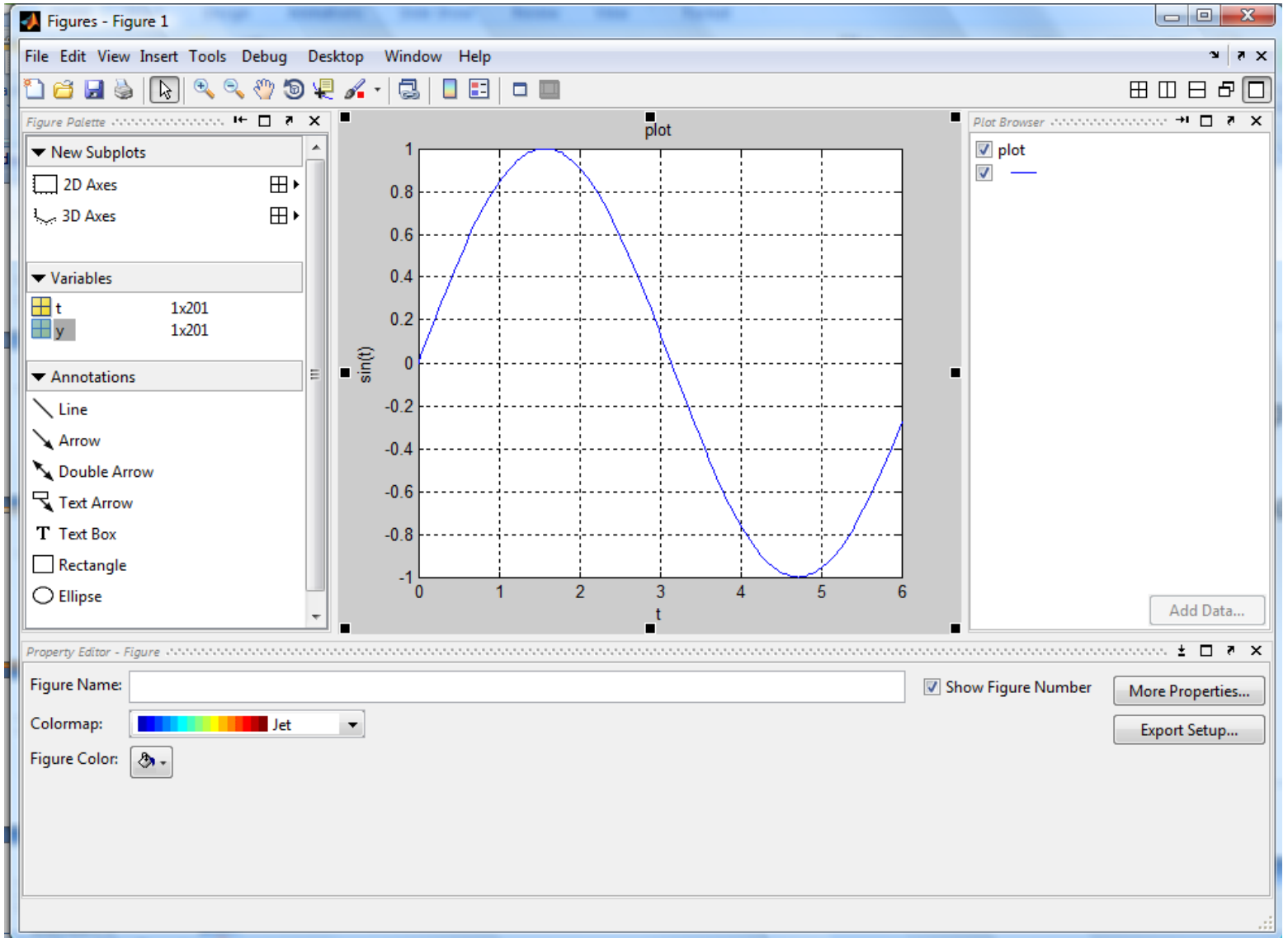
Figure Window

- Edit plot icon
  - Probably most important
  - When selected it allows you to move placement of title, axis labels, legend, colorbar, any other text or items on plot
  - Allows you to select objects of the figure to then edit properties of under the edit menu, edit current object properties option

Figure Window

- Plot tools
  - Clicking on plot tools icon in toolbar opens up plot tools window
  - Can also be done with **plottools** from command line or m-file
  - Easy way to insert text boxes, arrows, other things

Plot Tools Window



Basic Plotting

- **grid** command will turn on x, y-axis grid lines
- **axis([xmin xmax ymin ymax])**
  - Command to set axis limits
  - **axis square** will set the axis limits such that the plot is square
  - **axis equal** will set scale factor and tick marks to be equal on each axis
  - **axis auto** will return axis scaling to auto, which is default

Basic Plotting

- **xlabel('text')**
- **ylabel('text')**
- **title('text')**
  - These three commands will set the xlabel, ylabel and title of the plot for you
  - The grid, axis and labeling commands all need to be performed **after** the plot command

```
>> grid
>> axis([0 6 -1 1])
>>
```

```
>> xlabel('t')
>> ylabel('sin(t)')
>> title('plot')
```

Basic Plotting

- **h = plot(x,y)**
  - **h** is a vector of handles to lineseries objects
  - This allows you to use h to edit the appearance of the data being plotted (i.e. line color, line style, line width)
- **h = figure(n)**
  - **h** is a handle to figure **n**
  - This allows you to modify properties of the figure object
    - Useful if you are creating many figures and want to modify them at different points in the program

Modifying Plots

- **set(h,'*PropertyName*',PropertyValue,...)**
  - The **set** command takes object handle **h** and sets property values for give property names for that object handle
    - If **h** is an array, the property values will be set for all object handles in **h**
- **a = get(h,'*PropertyName*')**
  - **get** returns the value of a given property

Modifying Plots

- **gcf** stands for get current figure handle
  - Will return handle of current figure
- **gca** stands for get current axis handle
  - Will return handle of current axis
- These two commands are very useful when used in conjunction with the **set** and **get** commands
- Allows you to edit the properties of the current figure or axis without having the handle specified in a variable

Modifying Plots

- MATLAB has many properties for various objects
  - We will only go over some of the basics
  - MATLAB help is great for discovering the rest
- Line color
  - Specified by either an RGB triplet from 0 to 1, or short or long name

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

Modifying Plots

- Line width
  - Specified by an integer, default is 0.5 points
  - Each point is 1/72 of an inch
- Line style

Line Style Specifiers

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Modifying Plots

- Line markers
  - Can choose to add markers at each data point
  - Can have only markers, no line

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point (see note below)
x	Cross
'square' or s	Square
'diamond' or d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram' or p	Five-pointed star (pentagram)
'hexagram' or h	Six-pointed star (hexagram)

Modifying Plots

- Example syntax
- Edit line style, line width and line color in plot command (with error)

```
>> plot(t,y,'linewidth',2,'linecolor','k','linestyle', ':')
??? Error using ==> plot
Invalid property found.
Object Name :  line
Property Name :  'linecolor'.

>> plot(t,y,'linewidth',2,'color','k','linestyle', ':')
>>
```

- Edit same things using lineseries handle

```
>> h = plot(t,y);
>> set(h, 'linewidth',1','color','g','linestyle','-.')
>>
```

Modifying Plots

- You can also quickly specify line style, marker type and color in the plot command

```
>> h = plot(t,y,'b-x');
>> h = plot(t,y,'bx');
>> h = plot(t,y,'bo');
>>
```

- These all set the line color to blue
  - The first sets the line style to a solid line with x at every data point
  - Second sets line style to none with an x at every data point
  - Third is the same as second, execpt marker is o instead of x

Modifying Plots

- Setting axis tick marks and labels
  - Use the xtick, ytick, ztick and xticklabel, yticklabel, zticklabel property names
  - Can specify one or both or none to let MATLAB auto select tick interval and labels

```
>> set(gca, 'xtick', [1 3 5]);
```

  - Puts tick marks at x = 1,3,5
  - **set(gca, 'xtick', []);**
    - Will remove all tick marks

Modifying Plots

- Four different ways to set the tick labels
  - **set(gca,'XTickLabel',{'1';'10';'100'})**
    set(gca,'XTickLabel','1|10|100')
    set(gca,'XTickLabel',[1;10;100])
    set(gca,'XTickLabel',['1 ';'10 ';'100'])
  - MATLAB runs through the label array until it labels all tick locations
    - If label array is too small MATLAB wraps around and begins again

Modifying Plots

- Can get really in-depth modifying figure size, background color, font size, font color, font type, axis color, etc
- Example program posted on website with more examples of plot formatting changes

Bar Graphs

- MATLAB will plot horizontal and vertical bar graphs

Syntax

```
bar(Y)
bar(x,Y)
bar(...,width)
bar(...,'style')
bar(...,'bar_color')
bar(...,'PropertyName',PropertyValue,...)
bar(axes_handle,...)
barh(axes_handle,...)
h = bar(...)
barh(...)
h = barh(...)
hpatches = bar('v6',...)
hpatches = barh('v6',...)
```

Description

A bar graph displays the values in a vector or matrix as horizontal or vertical bars.

`bar(Y)` draws one bar for each element in `Y`. If `Y` is a matrix, `bar` groups the bars produced by the elements in each row. The *x*-axis scale ranges from 1 up to `length(Y)` when `Y` is a vector, and `1` to `size(Y,1)`, which is the number of rows, when `Y` is a matrix. The default is to scale the *x*-axis to the highest x-tick on the plot, (a multiple of 10, 100, etc.). If you want the *x*-axis scale to end exactly at the last bar, you can use the default, and then, for example, type

```
set(gca,'xlim',[1 length(Y)])
```

at the MATLAB prompt.

`bar(x,Y)` draws a bar for each element in `Y` at locations specified in `x`, where `x` is a vector defining the *x*-axis intervals for the vertical bars. The *x*-values can be nonmonotonic, but cannot contain duplicate values. If `Y` is a matrix, `bar` groups the elements of each row in `Y` at corresponding locations in `x`.

`bar(...,width)` sets the relative bar width and controls the separation of bars within a group. The default `width` is `0.8`, so if you do not specify `x`, the bars within a group have a slight separation. If `width` is `1`, the bars within a group touch one another. The value of `width` must be a scalar.

`bar(...,'style')` specifies the style of the bars. `'style'` is `'grouped'` or `'stacked'`. `'group'` is the default mode of display.

Pie Charts

- MATLAB can also make pie charts

Syntax

```
pie(X)
pie(X,explode)
pie(...,labels)
pie(axes_handle,...)
h = pie(...)
```

Description

`pie(X)` draws a pie chart using the data in `X`. Each element in `X` is represented as a slice in the pie chart.

`pie(X,explode)` offsets a slice from the pie. `explode` is a vector or matrix of zeros and nonzeros that correspond to `X`. A nonzero value offsets the corresponding slice from the center of the pie chart, so that `X(i,j)` is offset from the center if `explode(i,j)` is nonzero. `explode` must be the same size as `X`.

`pie(...,labels)` specifies text labels for the slices. The number of labels must equal the number of elements in `X`. For example,

```
pie(1:3,{'Taxes','Expenses','Profit'})
```

`pie(axes_handle,...)` plots into the axes with the handle `axes_handle` instead of into the current axes (gca).

`h = pie(...)` returns a vector of handles to patch and text graphics objects.

Remarks

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`.

Stem Plots

- And stem plots for discrete data

Syntax

```
stem(Y)
stem(X,Y)
stem(...,'fill')
stem(...,LineSpec)
stem(axes_handle,...)
h = stem(...)
hlines = stem('v6',...)
```

Description

A two-dimensional stem plot displays data as lines extending from a baseline along the *x*-axis. A circle (the default) or other marker whose *y*-position represents the data value terminates each stem.

`stem(Y)` plots the data sequence `Y` as stems that extend from equally spaced and automatically generated values along the *x*-axis. When `Y` is a matrix, `stem` plots all elements in a row against the same *x* value.

`stem(X,Y)` plots `X` versus the columns of `Y`. `X` and `Y` must be vectors or matrices of the same size. Additionally, `X` can be a row or a column vector and `Y` a matrix with `length(X)` rows.

`stem(...,'fill')` specifies whether to color the circle at the end of the stem.

`stem(...,LineSpec)` specifies the line style, marker symbol, and color for the stem and top marker (the baseline is not affected). See LineSpec for more information.

`stem(axes_handle,...)` plots into the axes object with the handle `axes_handle` instead of into the current axes object (gca).

`h = stem(...)` returns a vector of stemseries object handles in `h`, one handle per column of data in `Y`.

Multiple Lineseries

- The plot command can plot up to n lineseries at one time
  - You can also specify line style, color and marker symbol for each

```
>> h = plot(t,y,t,y2,t,y3,t,y4);
>> h = plot(t,y,'b-',t,y2,'r-.',t,y3,'g:',t,y4,'k--');
>>
```

  - In this case h would be an array of length 4
    - Typical array notation would be used to edit a given lineseries

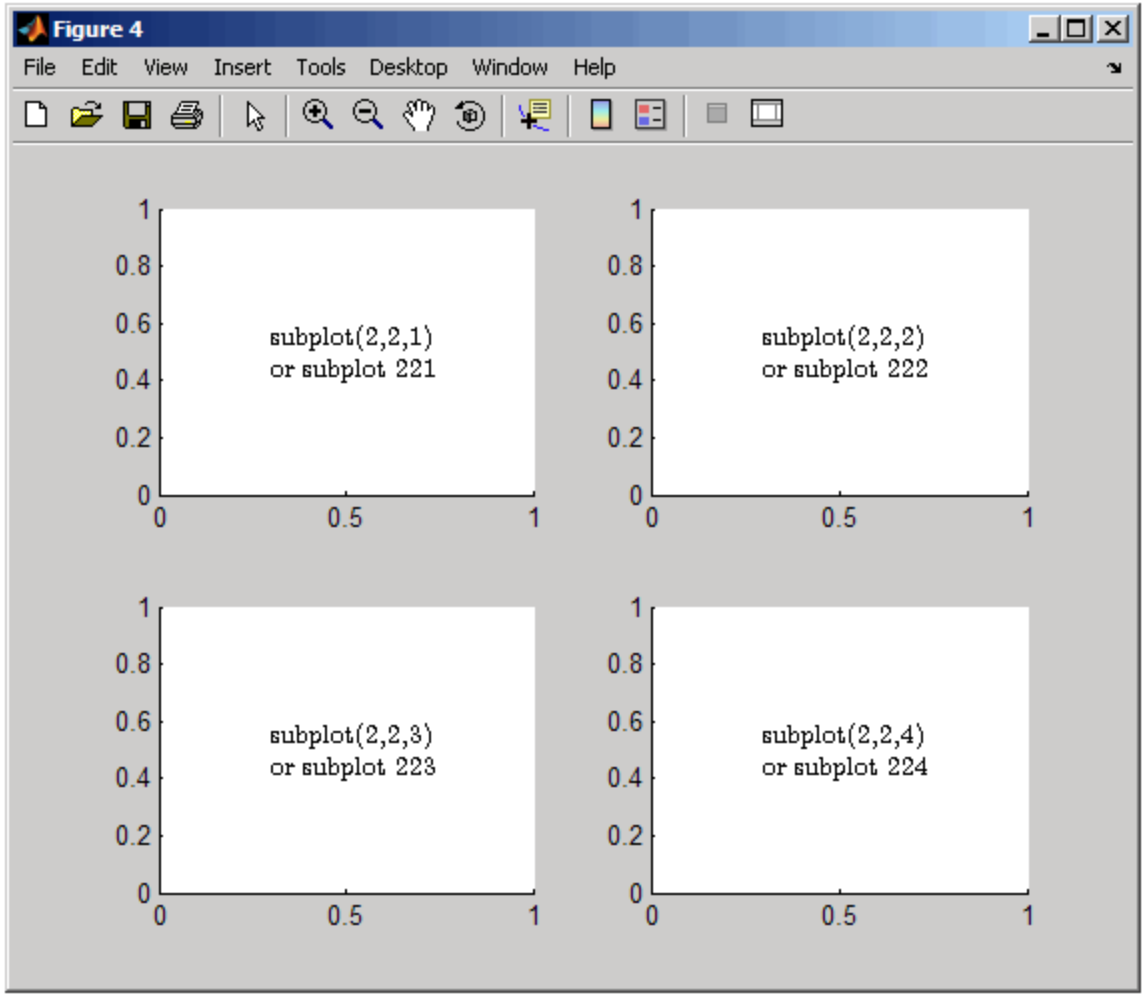
Subplots

- **h = subplot(m,n,p) or subplot(mnp)**
  h = subplot(m,n,p,'replace')
  - Three typical syntax uses for subplot command
  - Subplot will generate m by n subplots on a figure object
  - **p** specifies which subplot to create out of the **m*n** total
  - ‘replace’ will overwrite any subplot in that current position

Subplots

Subplots in Quadrants

The following illustration shows four subplot regions and indicates the command used to create each.

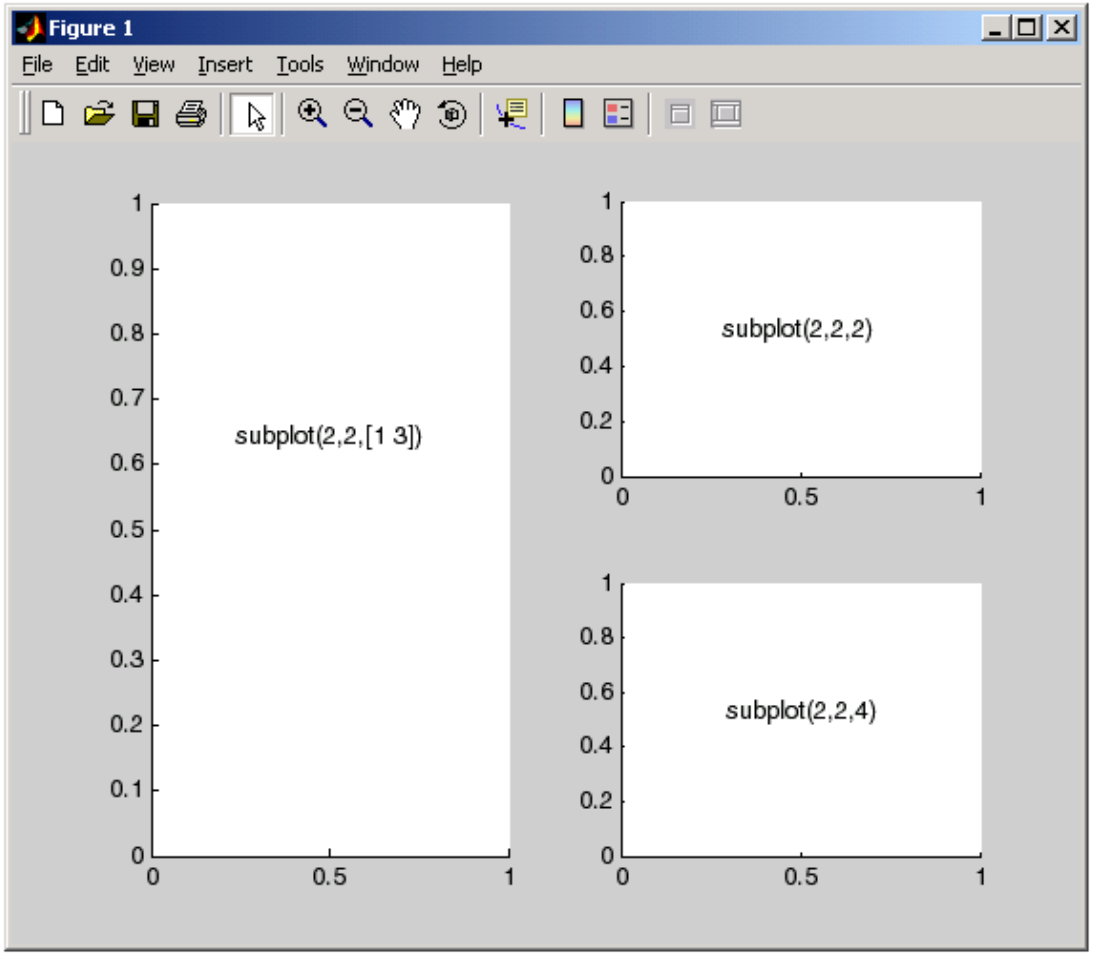


Subplots

Assymetrical Subplots

The following combinations produce asymmetrical arrangements of subplots.

```
subplot(2,2,[1 3])
subplot(2,2,2)
subplot(2,2,4)
```



Multiple Y-Axes

Syntax

```
plotyy(X1,Y1,X2,Y2)
plotyy(X1,Y1,X2,Y2,function)
plotyy(X1,Y1,X2,Y2,'function1','function2')
[AX,H1,H2] = plotyy(...)
```

Description

`plotyy(X1,Y1,X2,Y2)` plots `X1` versus `Y1` with *y*-axis labeling on the left and plots `X2` versus `Y2` with *y*-axis labeling on the right.

`plotyy(X1,Y1,X2,Y2,function)` uses the specified plotting function to produce the graph.

`function` can be either a function handle or a string specifying `plot`, `semilogx`, `semilogy`, `loglog`, `stem`, or any MATLAB function that accepts the syntax

```
h = function(x,y)
```

For example,

```
plotyy(x1,y1,x2,y2,@loglog) % function handle
plotyy(x1,y1,x2,y2,'loglog') % string
```

Function handles enable you to access user-defined subfunctions and can provide other advantages. See `@` for more information on using function handles.

`plotyy(X1,Y1,X2,Y2,'function1','function2')` uses `function1(X1,Y1)` to plot the data for the left axis and `function2(X2,Y2)` to plot the data for the right axis.

`[AX,H1,H2] = plotyy(...)` returns the handles of the two axes created in `AX` and the handles of the graphics objects from each plot in `H1` and `H2`. `AX(1)` is the left axes and `AX(2)` is the right axes.

Multiple X and Y-Axes

- Can go more low level than plotyy
  - Create plots with multiple x and y axes
  - Use **line** function to create individual lineseries objects
  - **line** will also create the figure object for you if you haven't created one yourself
- Nice example in MATLAB help
  - Search Using Multiple X- and Y-Axes

Histograms

- MATLAB will produce histograms using the **hist** or **histc** functions
- Provide MATLAB a vector and it will automatically bin data for you into 10 bins
- Can specify number of bins and let MATLAB determine bin size
- Can specify actual bin centers (**hist**) or bin ending points (**histc**)

Histograms

Syntax

```
n = hist(Y)
n = hist(Y,x)
n = hist(Y,nbins)
[n,xout] = hist(...)
hist(...)
hist(axes_handle,...)
```

Description

A histogram shows the distribution of data values.

`n = hist(Y)` bins the elements in vector `Y` into 10 equally spaced containers and returns the number of elements in each container as a row vector. If `Y` is an `m`-by-`p` matrix, `hist` treats the columns of `Y` as vectors and returns a 10-by-`p` matrix `n`. Each column of `n` contains the results for the corresponding column of `Y`. No elements of `Y` can be complex or of type `integer`.

`n = hist(Y,x)` where `x` is a vector, returns the distribution of `Y` among `length(x)` bins with centers specified by `x`. For example, if `x` is a 5-element vector, `hist` distributes the elements of `Y` into five bins centered on the *x*-axis at the elements in `x`, none of which can be complex. Note: use `histc` if it is more natural to specify bin edges instead of centers.

`n = hist(Y,nbins)` where `nbins` is a scalar, uses `nbins` number of bins.

`[n,xout] = hist(...)` returns vectors `n` and `xout` containing the frequency counts and the bin locations. You can use `bar(xout,n)` to plot the histogram.

`hist(...)` without output arguments produces a histogram plot of the output described above. `hist` distributes the bins along the *x*-axis between the minimum and maximum values of `Y`.

`hist(axes_handle,...)` plots into the axes with handle `axes_handle` instead of the current axes (`gca`).

- **histc** syntax very similar

Contour Plots

- Use **contour** and **contourf** to make contour and filled contour plots respectively

Syntax

```
contour(Z)
contour(Z,n)
contour(Z,v)
contour(X,Y,Z)
contour(X,Y,Z,n)
contour(X,Y,Z,v)
contour(...,LineSpec)
contour(axes_handle,...)
[C,h] = contour(...)
[C,h] = contour('v6',...)
```

Description

A contour plot displays isolines of matrix `Z`. Label the contour lines using `clabel`.

`contour(Z)` draws a contour plot of matrix `Z`, where `Z` is interpreted as heights with respect to the *x*-*y* plane. `Z` must be at least a 2-by-2 matrix that contains at least two different values. The number of contour lines and the values of the contour lines are chosen automatically based on the minimum and maximum values of `Z`. The ranges of the *x*- and *y*-axis are `[1:n]` and `[1:m]`, where `[m,n] = size(Z)`.

`contour(Z,n)` draws a contour plot of matrix `Z` with `n` contour levels.

`contour(Z,v)` draws a contour plot of matrix `Z` with contour lines at the data values specified in the monotonically increasing vector `v`. The number of contour levels is equal to `length(v)`. To draw a single contour of level `i`, use `contour(Z,[i i])`. Specifying the vector `v` sets the `LevelListMode` to manual to allow user control over contour levels. See `contourgroup properties` for more information.

`contour(X,Y,Z)`, `contour(X,Y,Z,n)`, and `contour(X,Y,Z,v)` draw contour plots of `Z` using `X` and `Y` to determine the *x*- and *y*-axis limits. When `X` and `Y` are matrices, they must be the same size as `Z` and must be monotonically increasing.

`contour(...,LineSpec)` draws the contours using the line type and color specified by `LineSpec`. `contour` ignores marker symbols.

`contour(axes_handle,...)` plots into axes `gerkaxes_handle` instead of `gca`.

`[C,h] = contour(...)` returns a contour matrix, `C`, that contains the *x*, *y* coordinates and contour levels for contour lines derived by the low-level `contourc` function, and a handle, `h`, to a contourgroup object. The `clabel` function uses contour matrix `C` to label the contour lines. `ContourMatrix` is also a read-only Contourgroup property that you can obtain from the returned handle.

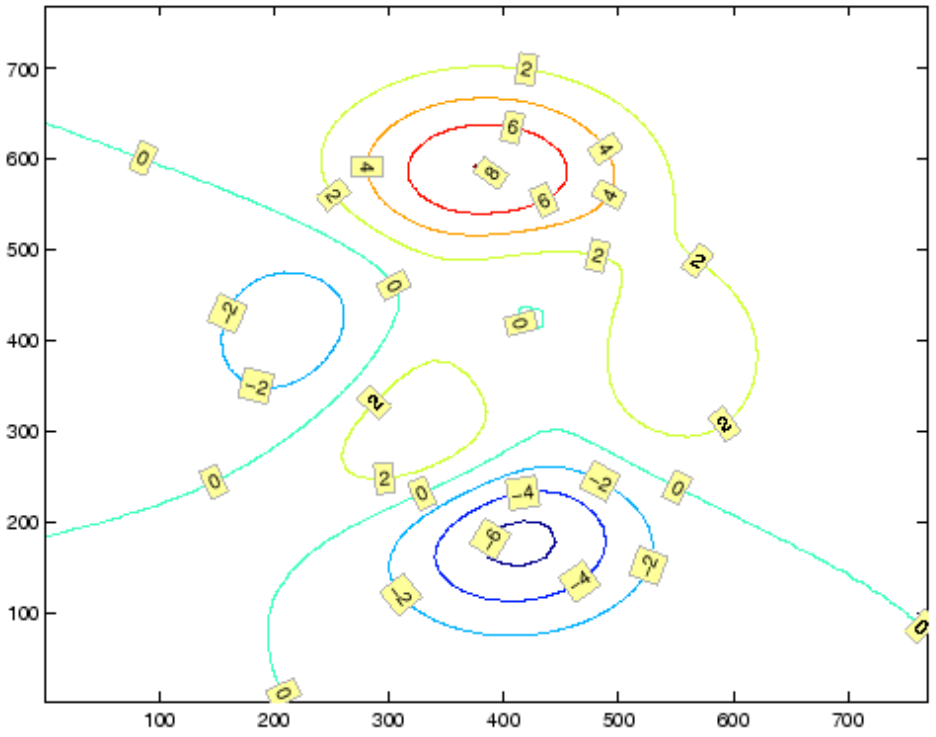
Contour Plots

- **contourf** syntax very similar to **contour**
- Can do interactive contour labeling
  - **clabel(C,h,'manual')**
  - Using this function call after a **contour** call will bring up the figure and let you manually select where the contours will be labeled
- Contour group properties can also be modified to set various properties of contour lines

Contour Plots

- To change the properties of the contour labels you need to create a text object and use that object handle

```
Z = peaks;
[C,h] = contour(interp2(Z,4));
text_handle = clabel(C,h);
set(text_handle,'BackgroundColor',[1 1 .6],...
    'Edgecolor',[.7 .7 .7])
```



For more examples using contour, see Contour Plots.

Contour Plots

- Contour labeling done using **clabel** function as we've seen

Syntax

```
clabel(C,h)
clabel(C,h,v)
clabel(C,h,'manual')
clabel(C)
clabel(C,v)
clabel(C,'manual')
text_handles = clabel(...)
clabel(...,'PropertyName',propertyvalue,...)
clabel(...'LabelSpacing',points)
```

Description

The `clabel` function adds height labels to a 2-D contour plot.

`clabel(C,h)` rotates the labels and inserts them in the contour lines. The function inserts only those labels that fit within the contour, depending on the size of the contour.

`clabel(C,h,v)` creates labels only for those contour levels given in vector `v`, then rotates the labels and inserts them in the contour lines.

`clabel(C,h,'manual')` places contour labels at locations you select with a mouse. Press the left mouse button (the mouse button on a single-button mouse) or the space bar to label a contour at the closest location beneath the center of the cursor. Press the **Return** key while the cursor is within the figure window to terminate labeling. The labels are rotated and inserted in the contour lines.

`clabel(C)` adds labels to the current contour plot using the contour array `C` output from `contour`. The function labels all contours displayed and randomly selects label positions.

`clabel(C,v)` labels only those contour levels given in vector `v`.

`clabel(C,'manual')` places contour labels at locations you select with a mouse.

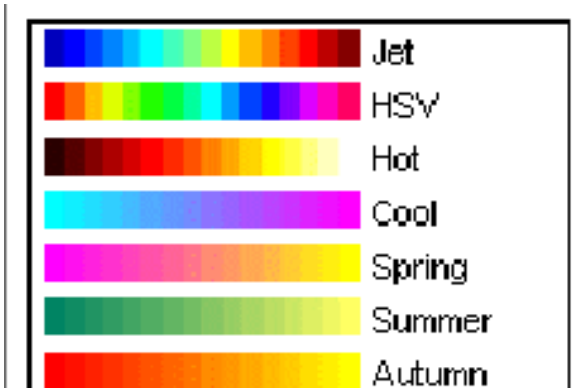
`text_handles = clabel(...)` returns the handles of text objects created by `clabel`. The `UserData` properties of the text objects contain the contour values displayed. If you call `clabel` without the `h` argument, `text_handles` also contains the handles of line objects used to create the `'+'` symbols.

`clabel(...,'PropertyName',propertyvalue,...)` enables you to specify text object property/value pairs for the label strings. (See Text Properties.)

`clabel(...'LabelSpacing',points)` specifies the spacing between labels on the same contour line, in units of points (72 points equal one inch).

Colormaps

- Colormaps can be specified for contour plots
  - MATLAB has many built in colormaps
  - **colormap(map)**
    - This sets the colormap to the one specified
  - **colormap(map(n))**
    - This will set the colormap and use **n** colors evenly spaced from the given colormap



Colormaps

- Can create your own colormaps
- Need to be an array of RGB triplets (3 column array) in the range of 0-1
- Then pass array name to **colormap** function

```
>> ex_cmap=zeros(5,3)

ex_cmap =

     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

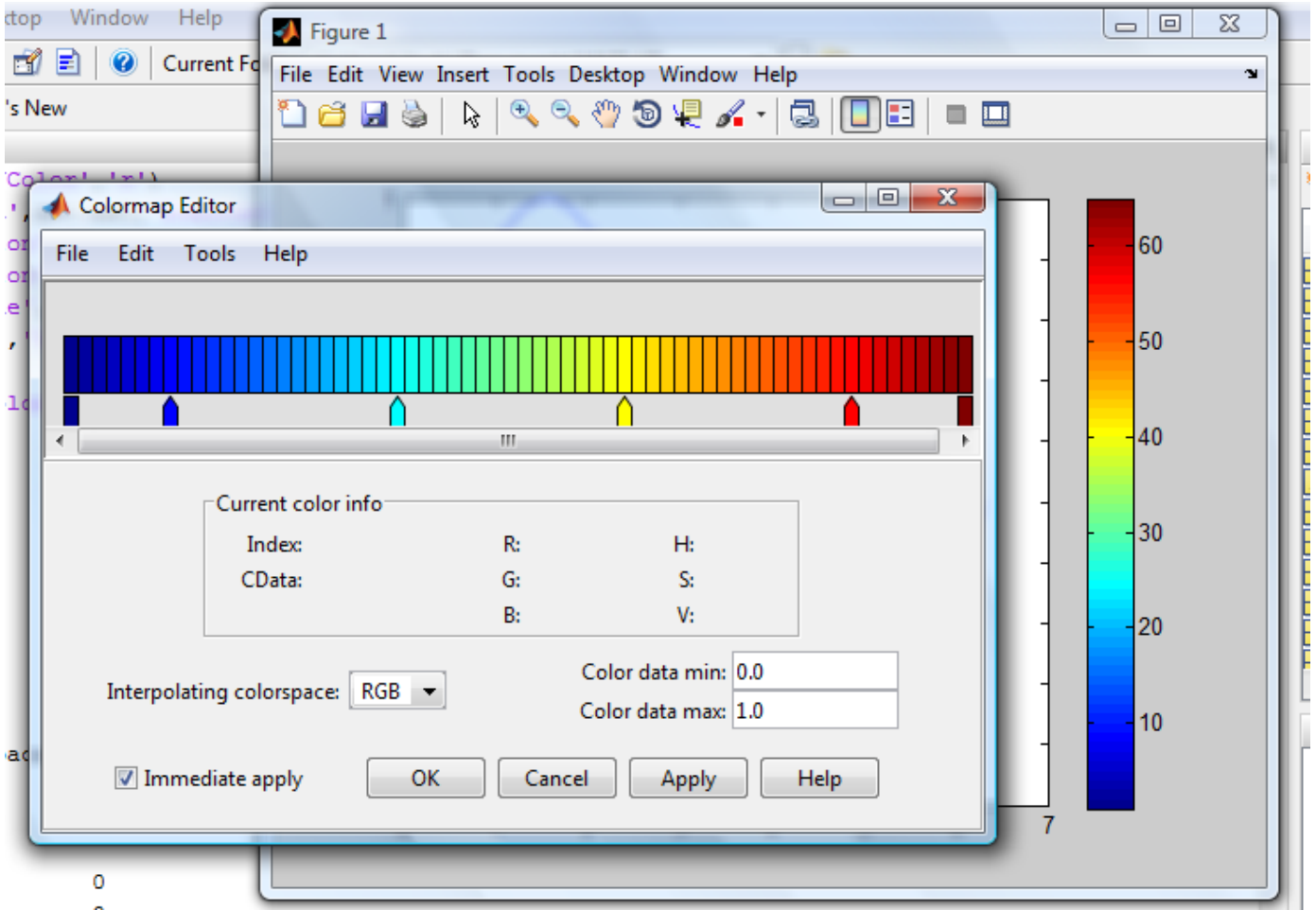
```
>> ex_cmap(:,1) = linspace(0,1,5)

ex_cmap =

         0         0         0
    0.2500         0         0
    0.5000         0         0
    0.7500         0         0
    1.0000         0         0

>> colormap(ex_cmap)
```

Image Plots

- Use **imagesc** to plot a matrix as an image
  - Useful if you don't want to use contours
  - If you have high resolution data these look fairly nice
  - **imagesc(C)**
    imagesc(x,y,C)
    imagesc(...,clims)
    imagesc('PropertyName',PropertyValue,...)
    h = imagesc(...)
  - Plot matrix **C**, specify x and y axis bounds, **clims** specifies limits of colormap

Image Plots

- You can interactively change the colormap on any given plot



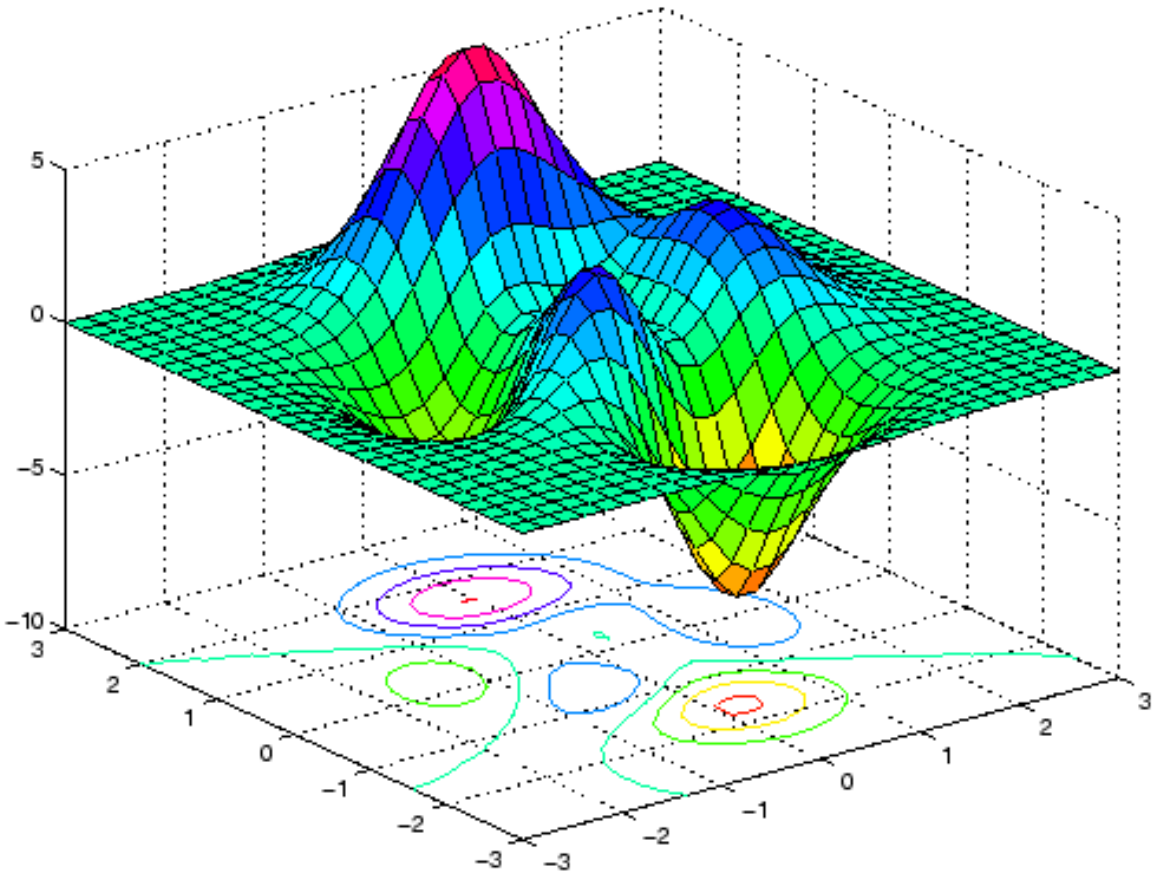
Surface Plots

- **surf(Z)** will create a 3-D surface plot of **Z** using the current colormap to color the surface
- **surfc(Z)** is the same as **surf** except that it also draws a contour map under the surface
- **mesh** function will create a surface without filled faces

Surface Plots

Display a surfaceplot and contour plot of the peaks surface.

```
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv
axis([-3 3 -3 3 -10 5])
```



Questions?