CSI606

MATLAB

Syllabus

- **Instructor - Jeff Solka**

- **Contact Information**
  - **jlsolka@gmail.com**
  - **540-653-1982 (W)**
  - **540-371-3961 (H)**

- **Dates and Times**
  - **11/5/2005 10 a.m. - 5 p.m. ST228**
  - **11/12/2005 10 a.m. - 5 p.m. ST228**

- **Texts**
  - **Mastering MATLAB 6, Hanselman and Littlefield**
  - **Graphics and GUIs in MATLAB by Marchand and Holland**
  - **Data Analysis and Graphics Using R (Hardcover) by John Maindonald, John Braun**

- **Grades**

  - Grades are based on 2 labs
  - **Labs due to me by 12/10/05**

Functions

- **Script M-Files**
  - **Creating script files**
  - **Running script files**
  - **Useful MATLAB functions for script files**
  - **Examples of script files**
- **Function M-Files**
  - **Properties of M-File functions**
  - **Syntax**
  - **Examples of function M-Files**
  - **Debugging and Profiling Tools**

Script M-files

- **Sometimes you will need to execute your commands in MATLAB using a script file rather than interactively.**

- **Reasons:**

  - **The number of commands is large.**

  - **You might want to change values of your variables and reevaluate the commands.**

  - **You need a history of what you've done.**

- **This allows you to type MATLAB commands in a text file and tell MATLAB to execute the commands as if you had typed them at the command prompt.**

- **These are called *script files* or *M-files*. We can use these terms interchangeably.**

- **These must be saved with the .m extension.**

- **To create a script file on a PC, choose New from the File menu and select M-file.**

- **This will open up the *Editor/Debugger* where you can enter your MATLAB commands.**

-

Example Script File

```
%  This is an example of an M-file

% Generate a matrix of normal random
   variables

x=randn(100,5);

% find the mean of each column

x_mu=mean(x)

% find the standard deviaion of each
   column

x_std=std(x)

% find the mean & std of the entire data
   set

x_mu_all=mean(x(:))
x_std_all=std(x(:))
```

Some Properties of Script Files

- **Once you type the name of your script file at the command line, MATLAB does the following:**
  - **It searches the current variables and built-in MATLAB commands.**
  - **If your M-file name is not there, it searches the current directory.**
  - **If the file is there, it opens it and runs the commands.**
  - **If it is not in the above places, MATLAB searches the path.**
- **Variables created by the commands in your M-file stay in the workspace.**
- **Commands in your M-file have access to all of the variables referred to in the M-file.**
- **Usually, the commands in the M-file are not displayed as they are executed.**
- **The command echo on displays the commands to the command window as they are read and evaluated.**
- **The command echo off stops this and the command echo just toggles the state.**

Some Useful Script Functions

Recall that MATLAB provides several functions that are useful in M-files.

disp	**Display results.**
echo	**Echo the file commands.**
input	**Prompt the user for input.**
keyboard	**Gives temporary control to the keyboard.**
	(return quits)
pause	**Pause until any key is pressed.**
pause(n)	**Pause for 'n' seconds.**
waitforbuttonpress	**waits for a mouse click**
	or keystroke over a plot

Functions

- **There is a different type of M-file called an M-file Function.**
- **You can speed up your code significantly by writing things as M-file functions rather than just script files.**
- **Functions can be thought of as black boxes: all you see is what goes in and what comes out.**
- **Any commands that are evaluated or variables that are created are hidden from the user.**
- **These are very useful for evaluating mathematical sequences of commands that you might want to use many times.**
- **It is similar to the script file, in that it also is a text file with the .m extension, however intermediate variables within the function do not appear in or interact with the MATLAB workspace.**
- **You create this file in the same manner, but with slightly different syntax.**
- **Pages 176-178 of the text lists the rules and criteria governing Function M-Files.**

Functions

- **A Function M-file is different from a script file; the key differences are:**

- **Functions communicates with the workspace through the variables passed into it and that are produced from it.**

- **Any intermediate variables created by the function are hidden and do not interact with the workspace.**

- **There is a specific syntax that tells MATLAB that an M-file is a function and not a script file.**

- **The first line of an M-file function must be either:**

```
function arg_out = function_name(arg_in)
```

```
or
```

```
function [arg1,arg2] = func_name(in1,in2)
```

- **It is a good idea to put several comment lines in the beginning of your function file.**

- **These will be returned by the `help` command.**

- **The first comment line, also called the H1 line, is searched by the `lookfor` command.**

Functions

- **Rules about function M-Files:**
- **You should always name the function and its file name the same.**
- **The first time MATLAB executes a Function M-file, it opens the text file and compiles the commands.**
- **The function is now represented in memory, which speeds execution time.**

- **Other functions that are called by a function M-File are also compiled.**

- **In contrast, script M-files are interpreted and are not compiled even if they are called by another function.**

- **Here is a simple function that returns a square of the value passed.**

```
function x = squareit(y)
%SQUAREIT returns the square of an array or
scalar.
%This is to show a very simple example of a
function.

x=y^2;
```

```
» help squareit

SQUAREIT returns the square of an array or
scalar.
This is to show a very simple example of a
function.
```

```
» squareit(5)
ans =
25
»
```

Function M-files

- **Key characteristics:**
- **Functions can have zero or more input arguments.**
- **Functions can have zero or more output arguments.**
- **Functions can be called with fewer input or output variables than were specified in the function...but not more.**
- **An error is returned if they are called with more input or output arguments.**
- **If a function has more than one output variable, then they are enclosed in brackets.**

```
function [mu,std] = stats(x)
```

- **The number of input and output arguments that are used when a function is called are available inside the function.**
- **These are available with the nargin and nargout variables.**
- **They are usually used to set default input variables and to determine what output to use.**
- **You can also have a variable number of inputs and outputs.**
- **Use the varargin and varargout functions.**
- **MATLAB packs the specified inputs and outputs into a cell array.**

Functions

- **When a function declares one or more output variables and you do not want any output, then do not give the output variable a value.**

- **Functions have their own workspace that is created with each function call and then deleted when the function completes execution.**

- **For this reason, you can call variables the same in both workspaces.**

- **If a predefined variable (e.g., pi) is redefined in the MATLAB workspace, it does not carry over into the function's workspace and vice versa.**

- **The input variables are not copied into the function workspace, but their values are readable within the function.**

- **If any of the values within an input variable are changed, then the array is copied into the function workspace.**

- **If an output variable is named the same as an input variable, then it is copied.**

- **To save memory you should extract the portions of arrays that you wish to operate on.**
-

Global Variables and GUIs

- **Functions can share variables with other functions, the MATLAB workspace, and recursive calls to themselves if the variables are declared `global`.**

- **To gain access to a global variable it must be declared `global` within each desired workspace.**

- **The use of global variables should be avoided as they often can lead to conflicts, confusion, and be difficult to debug. To avoid these problems consider the following suggestions when creating global variables;**

- **Use all capital letters in the global variable's name.**

- **Include the M-File name in the variable's name.**

- **If you can find an alternative to a global, do it.**

- **MATLAB searches for functions, as mentioned with script files.**

- **If you call a script file within a function, then the script file sees only the function workspace, not the MATLAB workspace.**

- **Functions can be called recursively.**

- **This is common with Graphical User Interfaces (GUI' s).**

Functions

- **M-file functions stop executing and return when they reach the end of the file or the command** return **is reached.**
- **For error reporting and debugging there are three functions you can use.**
- `disp` **displays a variables value without showing its name; you can use this with string variables to show messages.**
- **error displays a string in the command window, aborts the function execution and returns control to the keyboard.**
- `warning` **displays a string as well, but does not abort the function.**
- **MATLAB keeps track of the modification date of M-files that you write.**
- **If an M-file function is referenced that was previously compiled into memory, then it compares the dates with the one on disk.**
- **If the dates are the same, the compiled code is executed.**
- **If the file on disk is newer, the newer file is compiled and used.**

Subfunctions and Local Functions

.

- **Function M-files can contain code for more than one function. These are called *subfunctions* or *local functions*.**

- **The first function is the primary function and is invoked with the M-file name.**

- **Subsequent functions in the file are subfunctions.**

- **Sub-functions are visible only to the primary function or other sub-functions in the same file.**

- **Each sub-function begins with its own function definition line.**

- **When you call a function within an M-file, it first checks to see if it is a sub-function.**

Private M-files

- **Private M-Files are standard function M-Files that reside in a subdirectory (which must be named 'private') of the calling function.**

- **These functions are visible only to the same or parent directory.**

- **Since these functions are not visible outside the parent directory, they are not visible to the command line or any outside functions.**

- **Therefore they can use the same names as other functions in other directories.**

- **Once MATLAB checks for sub-functions, it next checks in the private directory.**

- **Directory structure must be maintained. This is a concern when porting M-Files.**

Creating Your Own Toolbox

- **The 'Toolbox' directory is a subdirectory containing completed functions that are cached by MATLAB. We can add our own subdirectory in there.**

- **We need to place Readme.m and Contents.m in our subdirectory (let's call it MyToolBox).**

- **Readme.m is a script file containing comment lines that describe late breaking changes or undocumented features of our toolbox.**

- **Contents.m contains comment lines that list all M-files in our Toolbox. The first line should contain the name of the Toolbox and the second line the date and version.**

- **`Readme.m` is accessed by `whatsnew` MyToolBox**

- **`Contents.m` is accessed by `help` MytoolBox.**

Command and Function Duality

- **You have used some MATLAB commands such as `clear`, `whos`, `dir`, `ver`, `help`, etc.. MATLAB lets you create your own new commands.**

- **There are two differences that distinguish commands from functions.**

- **Commands do not have output arguments**

- **Input arguments to commands are not enclosed in parenthesis.**

- **Commands are actually interpreted as the following example indicates.**

```
>> whatsnew MyToolbox     %command form
```

is interpreted as

```
>> whatsnew('MyToolbox')
```

Here's an example,

```
» which colordef
C:\PROGRAM FILES\MATLAB\toolbox\local\colordef.m
```

We can also do this:

```
» s=which('colordef')

s =

C:\PROGRAM FILES\MATLAB\toolbox\local\colordef.m
```

Command Function Duality

- **The following does not work since it mixes function and command syntax.**

```
» s=which colordef
??? s=which colordef
|
Missing operator, comma, or semi-
   colon.
```

feval

- **You can pass a character string name of a function to another function for evaluation. You saw this in converting strings with `eval`. MATLAB provides a more efficient method for certain cases. This is `feval`.**

- **`eval` calls the entire MATLAB interpreter to evaluate a string.**

- **`feval` executes only functions given by a string.**

- **`feval` is usually used inside functions which have the names of other functions as arguments.**

- **These are equivalent:**

```
a = feval('myfunction',x)
```

```
a = myfunction(x)
```

`feval` works with multiple arguments.

```
[a,b] = feval('myfunction',x,y,z,t)
```

is the same as

```
[a,b] = myfunction(x,y,z,t)
```

`feval` is much more efficient than `eval`

- **Use `feval` when you have to evaluate a function many times or in an interative procedure.**

in-line functions and feval

- **Normally *myfunction* is the name of an M-file function, however you can use `feval` with `inline` and express an entire function as a character string.**

- **Here's how we would do this with `eval`**

```
» myfun = '100*(y -x^2)^2 + (1-x)^2'; %just a
   string
» %we can use eval if we set the values of x
   and y first
» x=1.2;
» y=2;
» a=eval(myfun)

a =

31.4000
```

variables had to be defined previous to `eval(myfun)`

- **Here's what happens when we use `inline`**

```
» myfuni = inline(myfun, 'x', 'y') % make it
   inline

myfuni =

Inline function:
myfuni(x,y) = 100*(y -x^2)^2 + (1-x)^2
```

In-lie functios and feval

Natural argument usage

```
» a = feval(myfuni,x,y)

a =

31.4000

» b = feval(myfuni, -2, 1) % works for any arguments

b =

909
```

Examine the function

```
» argnames(myfuni) %returns the arguments for the
   function

ans =

'x'
'y'

» formula(myfuni) %returns the formula for the
   function

ans =

100*(y -x^2)^2 + (1-x)^2
```

MATLAB Debugging

- **When developing MATLAB M-Files eventually errors will occur. MATLAB will tell you when you have errors. There are two types of errors, *syntax* and *run-time*.**

- ***Syntax* errors can be generated when an expression or function is compiled into memory.**

- **These can be things like**

  - **misspelled variables and function names, misplaced quotes or parenthesis, etc..**

- **MATLAB flags these errors immediately and provides feedback describing the error and where it occurred.**

- **Syntax errors are usually easy to identify.**

- ***Run-time* errors are generated when an operation leads to unnatural results.**

- **These can be caused by operations the result in things like**

- **Empty arrays**

- **NaNs**

- **MATLAB flags these errors then returns control to the command window.**

- **By their nature, run-time errors can be difficult to debug.**

Debugging by Hand

- **For simple problems *manual* debugging techniques can be quite useful.**

- **Remove semicolons from selected lines within your function so that intermediate results are dumped to the screen.**

- **Add statements that display variables of interest within the function.**

- **Place the `keyboard` command at places in the function where you want to examine a variable.**

- **Remember to use `return` to exit the keyboard state at the K>> prompt.**

- **Change the function M-file into a script M-file by placing a % before the function definition statement at the beginning of the M-file.**

- **This will let you examine the workspace when the termination occurs.**

Debugging Functions

When functions are complicated you can use the MATLAB *inline* debug commands.

- **MATLAB debugging functions do not require you to edit the M-File you are debugging. Debugging functions are similar to those in other high-level languages. The following table summarizes the *inline* debugging commands:**

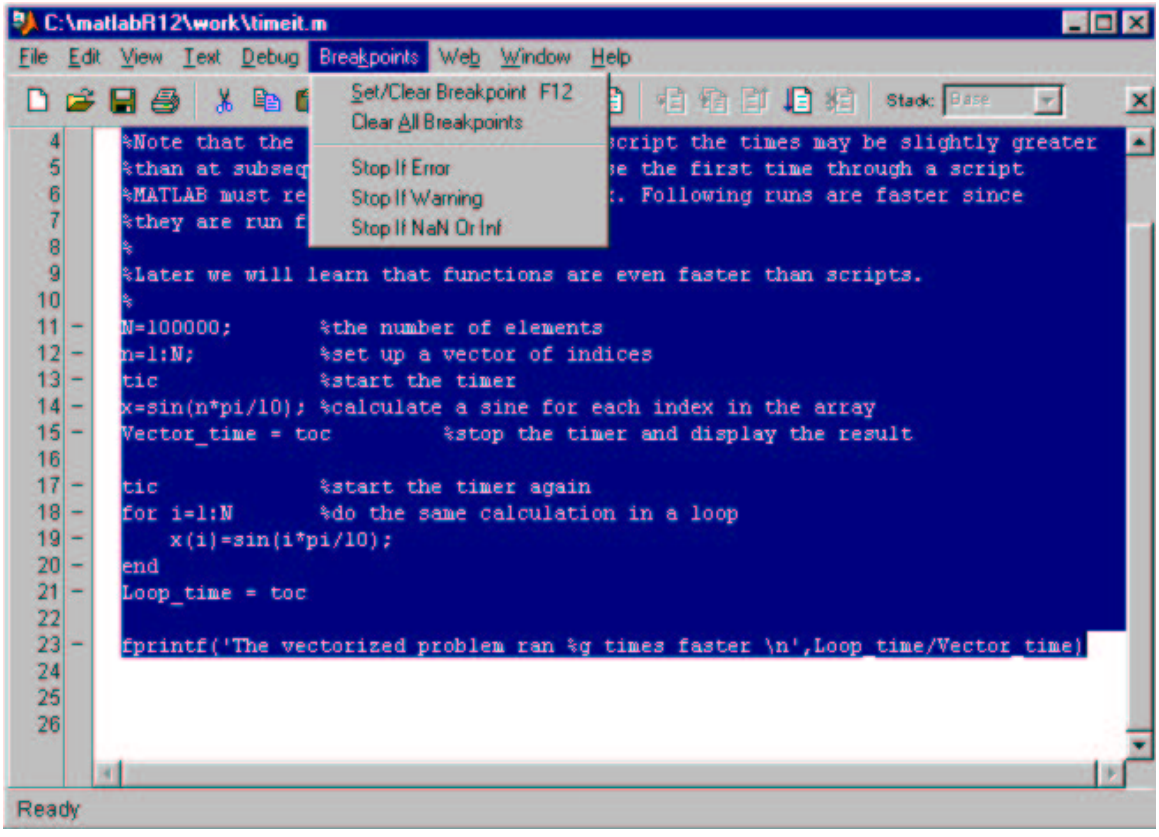
Debugging Command

Description

`dbstop in mfile`
`dbstop in mfile at lineno`
Set a breakpoint in `mfile` (at `lineno`)
`dbstop if warning`
`error`
`naninf (or infnan)`
Stop on any warning, run-time error, or when a NaN or Inf is generated.
`dbclear all`
`all in filename`
`in filename`
`if warning`
`if error`
`if naninf (or infnan)`
Remove breakpoints.
`dbstatus`
`dbstatus filename`
List all breakpoints (in filename).
`dbtype mfile`
`mfile m:n`
List `mfile` with line numbers (between line numbers `m` and `n`).
`dbstep`
`dbstep n`
Execute one or *n*-lines and stop.
`dbcont`
Resume execution.
`dbstack`
List who called whom.
`dbup`
Move up one workspace level.
`dbdown`
Move down one workspace level.
`dbquit`
Quit debug mode.

The Graphical Debugger

- **MATLAB on the PC features an integrated M-file editor / debugger.**
- **It can be launched right from your M-file editing session by typing `edit` at the command prompt.**
- **Or launch it by choosing File ▸ New or Open.**



The Graphical Debugger

- **Buttons exist on the toolbar to do rudimentary procedures such as single step, continue, and quit debugging.**

- **You can set and clear breakpoints in an easy manner.**

- **You can view the value of a variable or expression by highlighting it in the editor then using Text ▸ Evaluate Selection**

Profiling in MATLAB

- **MATLAB gives you tools to help you fine-tune, e.g., optimize, your code to avoid unnecessary function calls and calculations.**

- **The *Profiler* examines a running program to determine where the program is spending most of its time.**

- **With the profiler you can identify functions that are consuming the most time, then determine why you are calling them and look for ways to minimize their use.**

- **When you reach the point where most of the time is spent on calls to a small number of built-in functions, you have probably optimized the code as much as you can expect.**

- **Here is a rundown on some of the profiling functions.**

- **profile on** — **Begin profiling for the session.**

- **profile report** — **Display a profile report in HTML.**

- **profile plot** — **Plot the profile report using a `pareto` plot.**

- **profile off** — **Disable profiling.**

Basics of Plotting

- **Getting hardcopy**
- **2-D Plotting**
- **Using the 'plot' function**
  - **Linestyles**
  - **Grids and labels**
  - **Legends and Axes**
  - **Subplots**
  - **Multiple Figure Windows**
  - **Retrieving Data From Plots**
  - **Other Plotting Commands**
- **Special Symbols and Text**
- **3-D Plotting**
  - **The 'plot3' function**
  - **Mesh and Surface Plots**
  - **Contour Plots**

Getting Graphics Hardcopy

- **You can use the normal Windows 'Print' command from the 'File' menu.**

- **Use the menu from the Figure window.**

- **You can Copy/Paste into your document.**

- **NOTE: in previous versions of MATLAB the default background color is black. Be sure to invert the background or else you will have a black figure.**

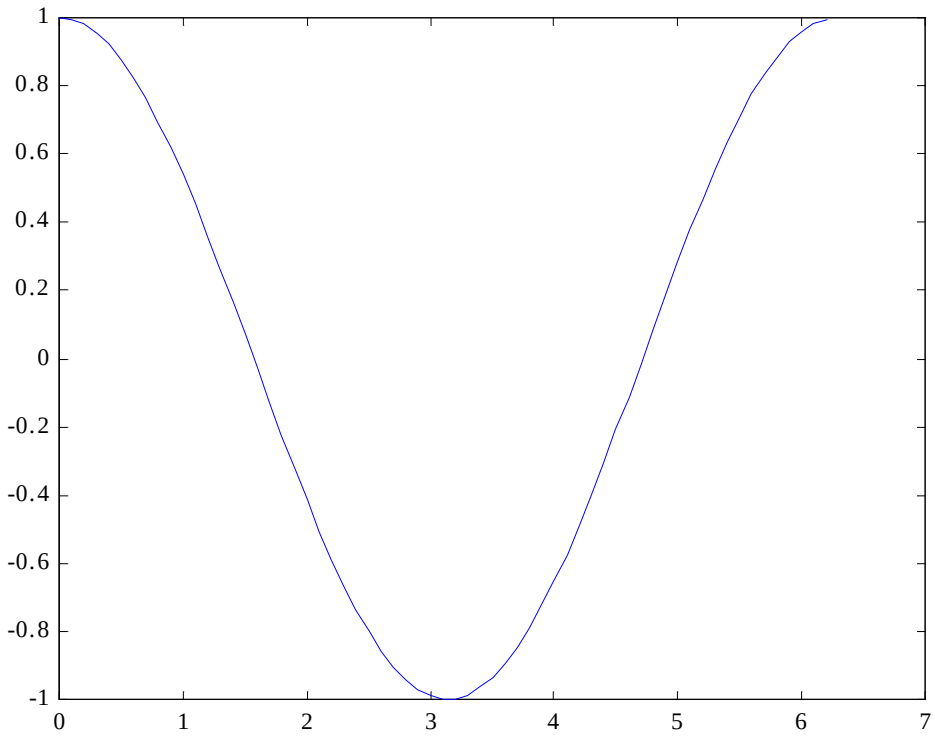
- **You can also use the 'print' command at the command line.**

- **Arguments to the print function call different devices.**

Plot Function

- **The most common command for plotting data in 2-D is the plot function. This function plots sets of data (vectors) using appropriate axes and connects the points with straight lines.**

```
» x=0:.1:2*pi;  % create horiz vector
» y=cos(x);     % find cos of each one
» plot(x,y)     % plot
```



plot Function

- **`plot` opens a Figure window, scales the axes to fit the data and plots the points.**

- **It adds a scale and tic marks to both axes.**

- **If a Figure window already exists, then it clears the current window and draws a new plot.**

- **You can plot several lines on the same plot by putting a series of them as arguments to the plot function.**

```
plot(x1,y1,x2,y2)
```

- **If one of the arguments is a matrix and the other a vector, then it plots each column of the matrix versus the vector.**

- **If you provide just one argument, then the following can happen:**

- **If it is complex, then it plots the real part versus the imaginary.**

- **If it is real-valued, then it plots the vector (or matrix) versus the index of its values.**

Linestyles, Symbols, and Colors

- **The default linestyle is a solid line...MATLAB allows you to choose from several.**

  solid line
  : **dotted line**
  -. **dash-dot line**
  – **dashed line**

- **If you plot several lines on one plot, then MATLAB starts with blue and cycles through the colors green, red, cyan, magenta, yellow, black, and white.**

- **You can use a symbol for each point and they will not be connected by lines.**

- **The symbols that are available are:**

. **point**
circle
x **x-mark**
+ **plus**
star
s **square**
d **diamond**
v **down triangle**
^ **up triangle**
< **left triangle**
right triangle
p **pentagram**
h **hexagram**

Linestyles, Symbols, and Colors

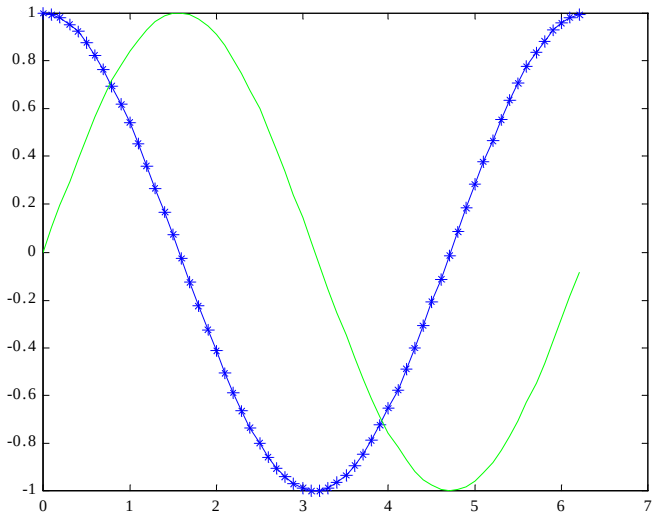
- **You can select the color of a line or symbol.**

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

- **You can combine both lines and symbols.**

- **For example, plot the sine and cosine on the same plot, with the cosine plotted as a line and with symbols.**

```
» y2=sin(x);
» plot(x,y,x,y,'b*',x,y2,'g-')
```



Grids and Labels

- **The command** grid **adds grid lines to the plot at the tic marks.**

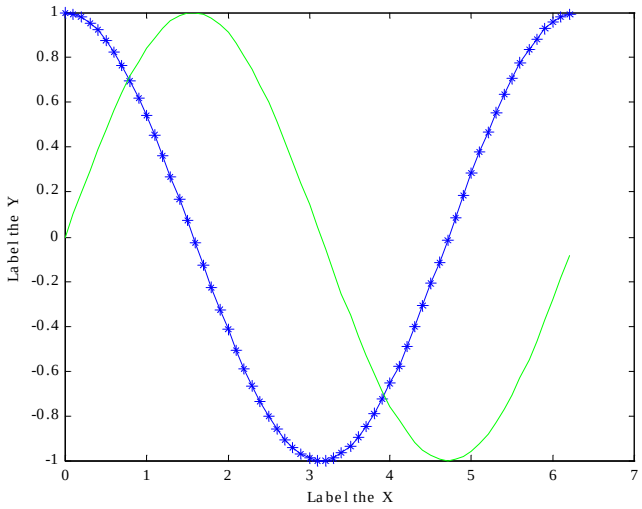
- **Repeated use of the command toggles the grid lines on and off.**

- **You can easily add a title to your plot using:**

- `title('My Plot Title')`

- **You can add labels to the horizontal and vertical axes by using:**

```
xlabel('Label the X')
ylabel('Label the Y')
```



Grids and Labels

- **Text can be added to any location on the plot with the text command:**

```
text(x,y,'label')
```

- **where the x and y represent the coordinates of the center left edge of the string in units corresponding to the axes.**

- **You can use the function gtext('label') to place the label with the mouse.**

- **The gtext function activates the current Figure window, gives you a cross-hair that follows the mouse and waits for the mouse click.**

- **The text is placed in the lower left corner of the first character at that location.**

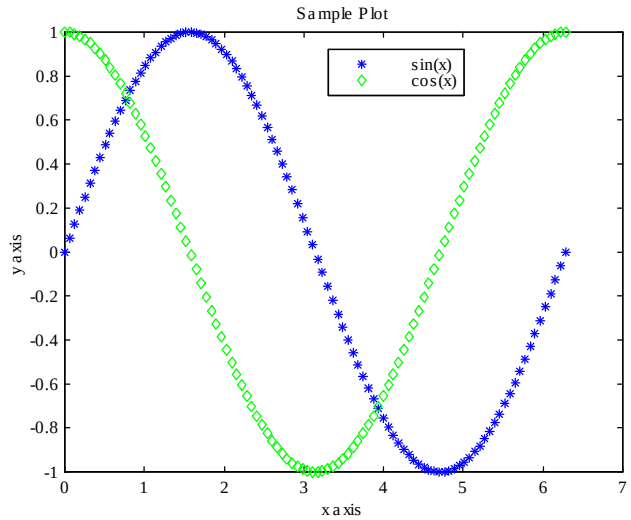
Legends

- **MATLAB provides the capability of showing legends to identify the different data.**

- **You can move the legend by holding the mouse button near the lower left corner and dragging.**

- **You can remove the legend from plots using legend off.**

```
» x=linspace(0,2*pi,100);
» sinx=sin(x);
» cosx=cos(x);
» plot(x,sinx,'b*',x,cosx,'gd')
» legend ('sin(x)', 'cos(x)')
» title('Sample Plot')
» xlabel('x axis')
» ylabel('y axis')
```



Zoom

- **zoom on turns zooming on.**
- **Click left mouse button in the Figure window to expand by a factor of two.**
- **Click right mouse button to zoom out by a factor of two.**
- **Click and drag rectangle to zoom into a particular area.**
- **zoom(n) zooms by a factor of n.**
- **zoom out returns the plot to its initial state.**
- **zoom off turns off zoom mode.**
- **zoom toggles the zoom state.**
- **For zoom to be used the legend must be turned off.**
- **Since both `zoom` and `legend` respond to mouse clicks they can interfere with each other.**

axes

- **You can use the command** axis **to change the axes of your plot.**

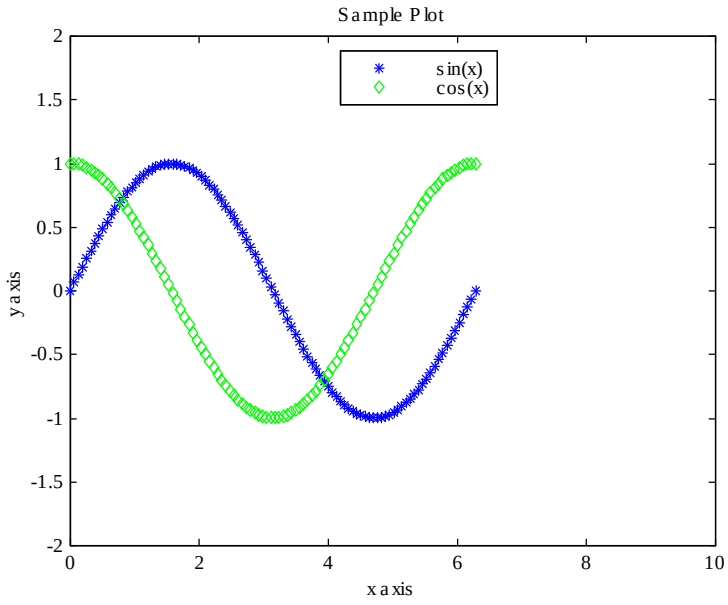
- **The argument to this is a four element vector containing the following information:**

- `[xmin xmax ymin ymax]`

- **You can use the function without any arguments to get the current axes values.**

```
» axis   % get current axes values
ans =
0     7    -1     1

» axis([0 10 -2 2]) % reset to new ones
```



Multiple Plots Per Page

- **You have seen that you can plot multiple data against a single axis using the `plot` command.**
- `plot(x,sinx,'b*',x,cosx,'gd')`
- **You can also add new plots to an existing plot by using the `hold` command.**
- **`hold on` tells MATLAB not to remove the existing axes when new `plot` functions are issued.**
- **If the new data do not fit within the current axes limits, the axes are rescaled.**
- **`hold off` releases the current figure window for new plots.**
- **`hold` with no arguments toggles the setting.**
- **The color cycle starts anew with each call to `plot`. You might want to specify plot colors when using `hold` so that lines aren't plotted in the same color.**
- **`ishold` returns 1 if `hold` is on.**

```
 » hold
Current plot held
» ishold

ans =

1
```

subplots

- **Sometimes you might want to plot more than one data set on multiple axes, rather than several plots on one axis. You can do this with the** subplot(m,n,p) **command.**

- **This gives a matrix of `m x n` plots in a single Figure window.**

- **The p stands for the p-th area to be active.**

- **The subplots are numbered left to right from the top row to the bottom.**

- **Be careful about putting too many plots in one Figure window.**

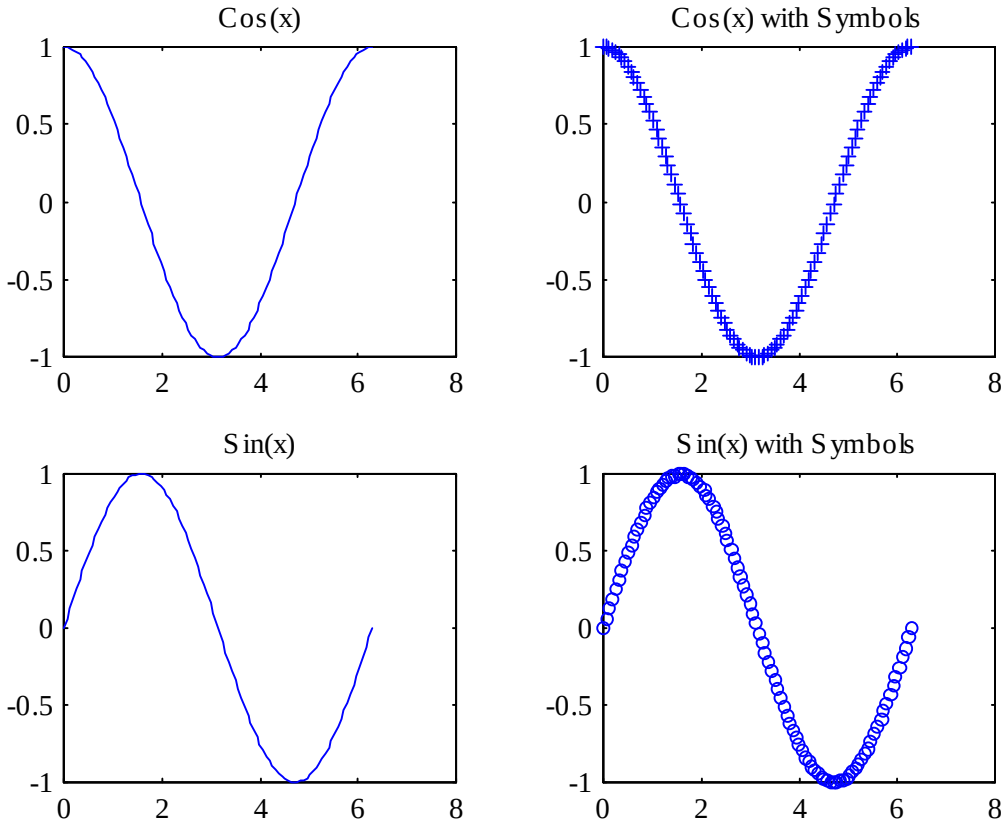
- **The active subplot is the one responsive to the previous commands to a Figure (e.g., axis, xlabel, ylabel, title)**

- **When you want to go back to one axis in a Figure window, you must use:**

- **subplot(1,1,1)**

subplots

```
» subplot(2,2,1),plot(x,cosx)
» title('Cos(x)')
» subplot(2,2,2),plot(x,cosx,'+')
» title('Cos(x) with Symbols')
» subplot(2,2,3),plot(x,sinx)
» title('Sin(x)')
» subplot(2,2,4),plot(x,sinx,'o')
» title('Sin(x) with Symbols')
```



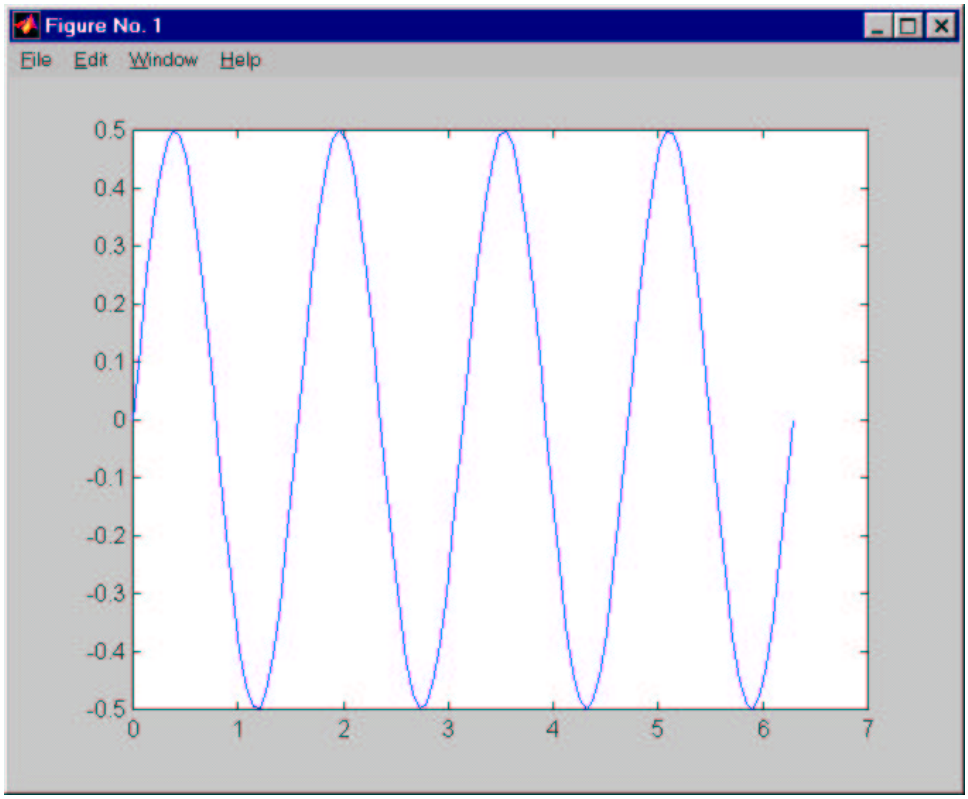
Multiple Figure Windows

- **You can create multiple figure windows and plot different data sets in different ways in each one.**

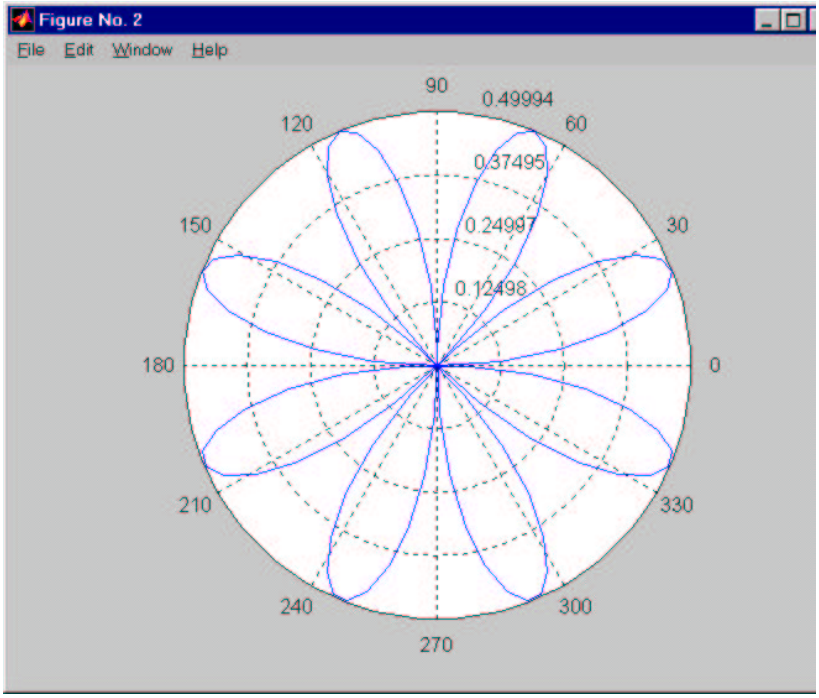
- **Select New Figure from the File menu or,**

- **Use `figure(n)` in the command window**

```
» figure(1)
» plot(t,r)
» figure(2), polar(t,r)
```



Multiple Figure Windows



- **Every time a new figure window is created, a number identifying it is returned and stored for future use.**
- **Each new figure is placed in the default figure position: click and drag to move figures around.**
- **Select active figures by ;**
- **clicking with mouse**
- **use `figure(n)` command**
- **Only the current figure is responsive to commands.**

Retrieving Data From Plots

- **The function** ginput **allows you to select points from a plot based on the position of a mouse click.**
- **The returned data are not necessarily points from the data set used to create the plot, but rather the explicit x and y coordinate values where the mouse was clicked.**
- **If points are selected outside the plot axes limits, the points returned are extrapolated values.**
- **Returned data are with respect to the current or active subplot.**
- `[x,y]=ginput(n)` **will retrieve** `n` **points. Not specifying** `n` **will allow retrieves until the Return key is pressed.**
- **Before using** `ginput`, `zoom` **and** `legend` **should be turned off, since all respond to mouse clicks and can interfere with each other.**

```
» [u,v]=ginput(3)

u =

2.5091
2.7273
2.8000


v =

0.5607
0.4451
0.2717
```

Other 2-D Plots

MATLAB provides a host of specialized 2-D plots.

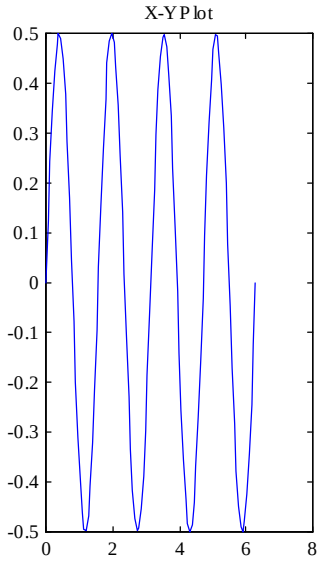
`polar`	**plot of polar coordinates as a function of angle and radius**
`bar`	**bar graph**
`stairs`	**stairstep graph...no spacing or lines**
`stem`	**stem plot**
`Errorbar`	**graph with errorbars**
`feather`	**displays angle and mag as arrows**
`compass`	**same as above, except it emanates from origin.**
`bar3`	**vertical 3-D bar chart**
`bar3h`	**horizontal 3-D bar chart**
`barh`	**horizontal bar chart**
`pie`	**pie chart**
`pie3`	**3-D pie chart**
`rose`	**draws polar histogram**

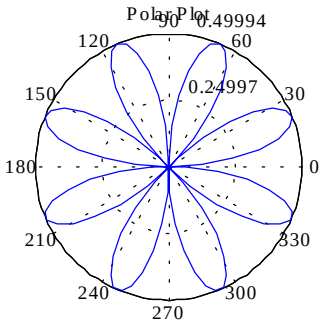
Polar Plots

- `polar(t,r,S)` **will create a plot in polar coordinates.**

- **`t` is the angle vector in radians**
- **`r` is the radius vector**
- **`S` is an optional character string describing color, marker symbol, and linestyle**

```
» t=linspace(0,2*pi);
» r=sin(2*t).*cos(2*t);
» subplot(1,2,1)
» plot(t,r),title('X-Y Plot')
» subplot(1,2,2)
» polar(t,r),title('Polar Plot')
```



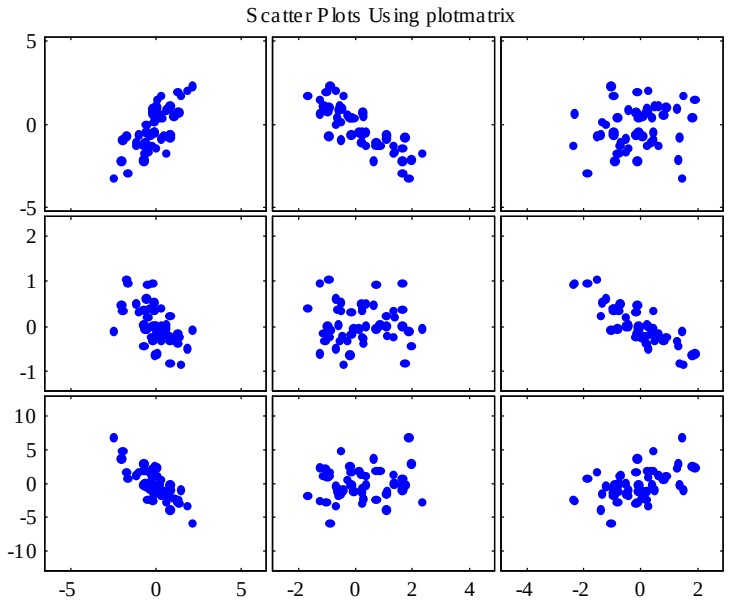


Other Plotting Commands

- **`plotmatrix(x,y)` scatter plots columns of x against the columns of y.**

- **If X is P-by-M and Y is P-by-N, PLOTMATRIX will produce an N-by-M matrix of axes.**

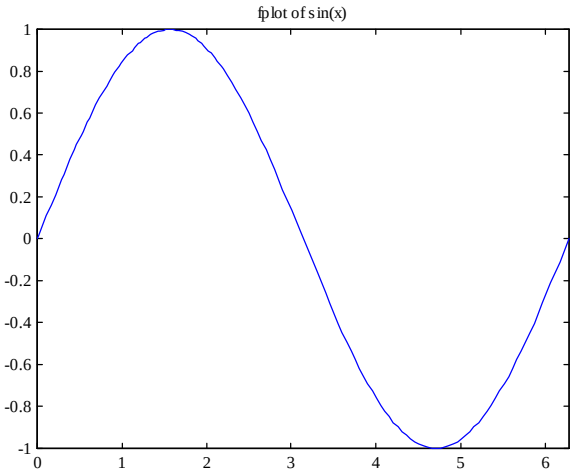
```
» x=randn(50,3); % 50 rows by 3 cols
» y=randn(3); % 3 rows by 3 cols
» plotmatrix(x,x*y)
» title('Scatter Plots Using
  plotmatrix')
```



Other plotting Commands

- **`fplot` allows you to plot a 1-d function without creating a data set.**
- **`fplot('fun',[xmin xmax])`**
- **`fplot('fun',[xmin xmax ymin ymax])`**
- **fun is a symbolic expression in one variable or the name of an M-file uses adaptive step control to produce a representative graph, concentrating its evaluation in regions where the function's rate of change is the greatest.**

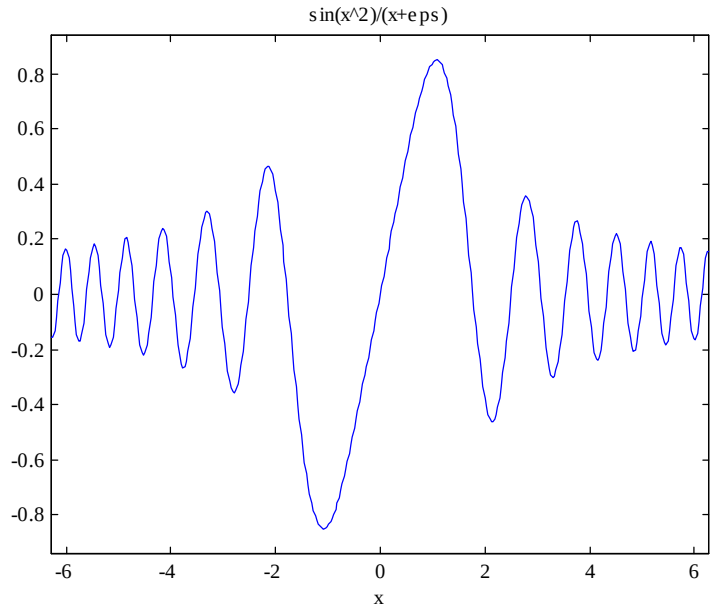
```
» fplot('sin(x)',[0,2*pi])
» title(' fplot of sin(x)' )
```



Other Plotting Commands

- `ezplot` **plots a function over the domain** `[-2*pi  2*pi]`.
- **The x-domain can be specified using the form**
- **`ezplot('FUN',[xmin xmax])`**
- **The x-axis label is the variable name.**
- **The title is the function `'FUN'`.**

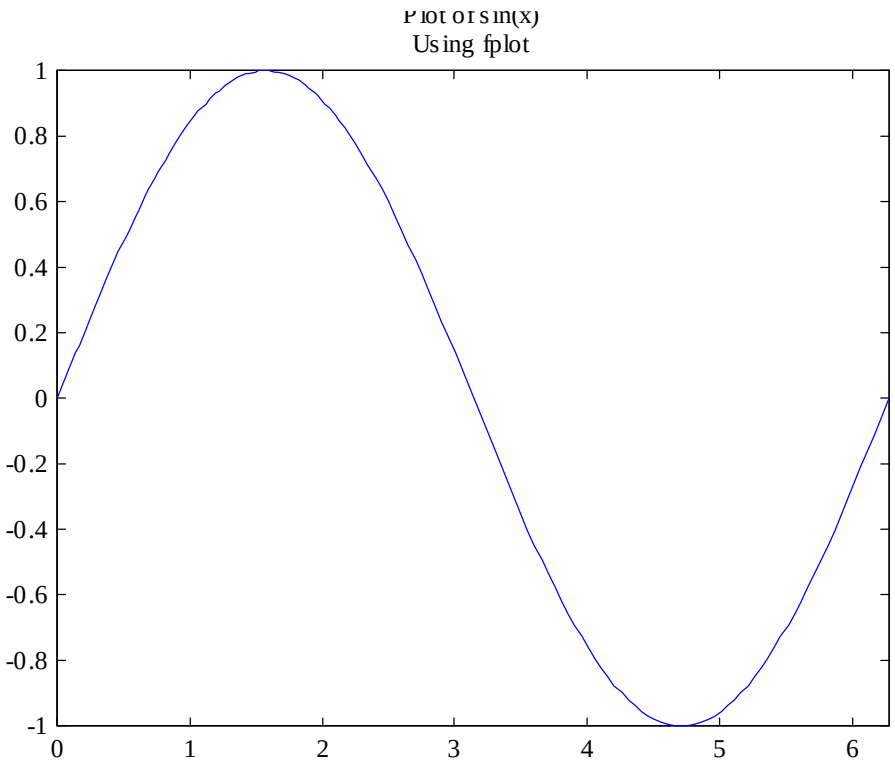
```
» ezplot('sin(x^2)/(x+eps)')
```



Special Text Formatting

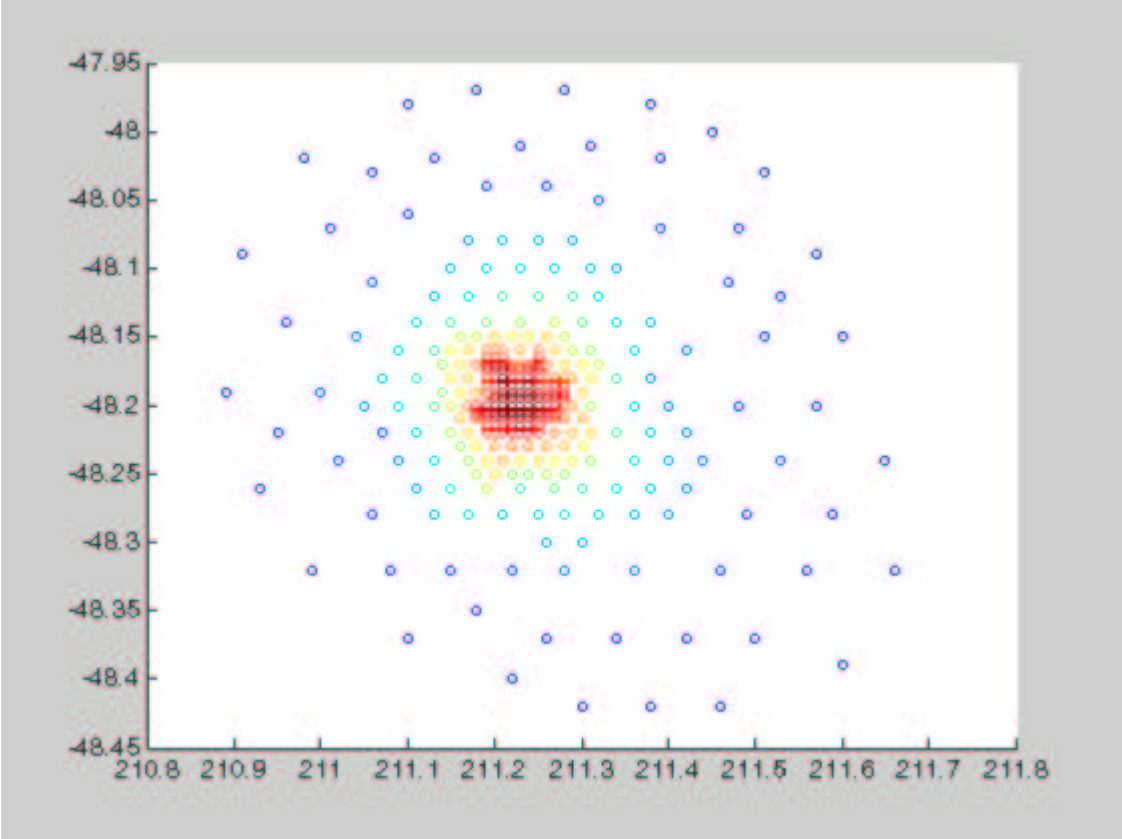
- **You can create multi-line text with any text string, including titles and axis labels by taking advantage of string arrays or cell arrays.**

```
» title({'Plot of sin(x)','Using fplot'})
```



Scatterplot Example

```
load seamount
%comes with
%the standard
%edition
scatter(x,y,5,z)
```



Symbols and Special Characters

- **There are over 75 symbols, including Greek letters and other special characters, that can be included in MATLAB text strings.**

- **You access these by embedding a subset of TeX commands within your string using the \ character.**

- **The available symbols and the character strings used to define them are listed in the table on page 375-376 of your text.**

- **A limited subset of TeX formatting commands are also available.**

- `^` **superscript**

- `_` **subscript**

- `\fontname` **font type**

- `\fontsize` **font size**

- `\bf,\it,\sl,\rm` **bold, italic, slant, normal roman**

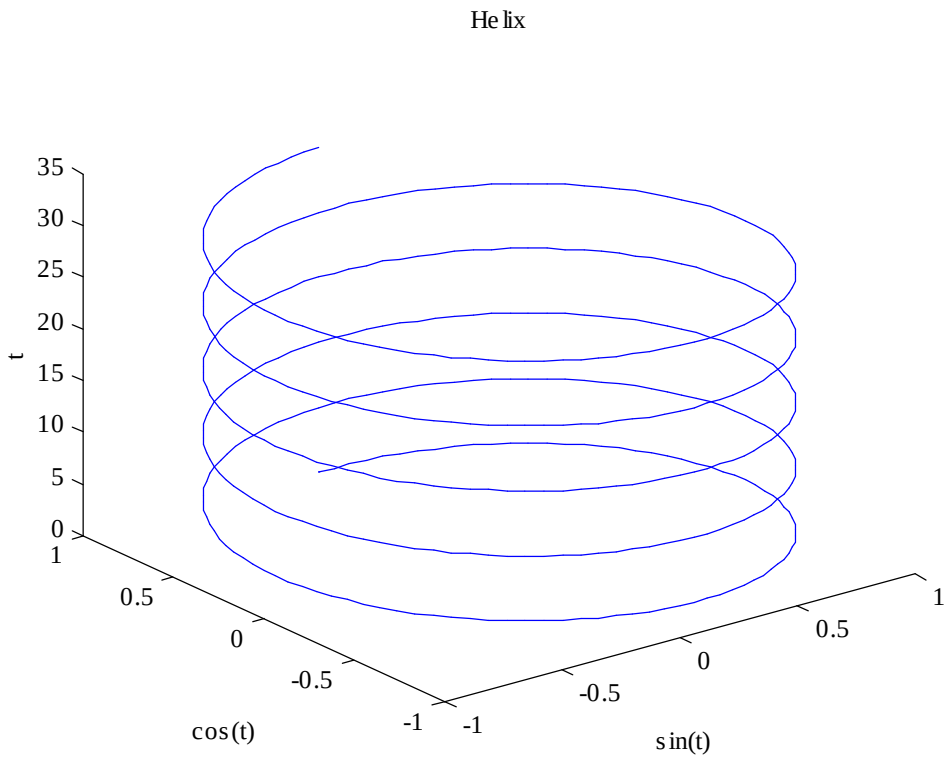
- ```
  gtext('\fontname{courier} \fontsize{16} \it x_{\alpha} + y^{2\pi}')
  ```

- **Produces** $x-_{\alpha} + y^{2\pi}$
  ```

3-D Plotting

- **The plot3 function is similar to the 2-D plot function, except we are now in three dimensions. The syntax is similar to plot except that you need to provide 3 data sets or vectors.**

```
» t=0:pi/50:10*pi;
» plot3(sin(t),cos(t),t)
»xlabel('sin(t)'),ylabel('cos(t)'),zlabel('t')
» title('Helix')
```



3-D Plotting

- As you just saw, there is a zlabel function that you can use with 3-D plots.
- You can use the hold command or several arguments to the plot3 function just like in the 2-D case.
-
- You have a certain viewpoint with 3-D graphics, specified by azimuth and elevation.
- The azimuth is the angle with respect to the x=0 plane.
- The elevation is the angle with respect to the z=0 plane.
- You can change this view with the command:
- `view([az,el])`
- See the page 398 - 400, of your text for a complete description of the `view` command.
- You can interactively rotates the view of a 3-D plot with the command `rotate3d`.
- Try the following;

```
» [x,y,z]=peaks;
» mesh(x,y,z);
» rotate3d
```

3-D Plotting

- **Sometimes we need to view a scalar function of two variables:**
- **z=f(x,y)**
- **A plot of this is a surface in 3 dimensions.**
- **To plot this in MATLAB, the values of z are stored in a matrix.**
- **One way to get these values is to first create a matrix of x and y values:**

```
[X,Y]=meshgrid(-3:3,1:5)
```

- **`meshgrid` will duplicate x for each of the rows in y and will duplicate y as a column for each of the columns in x.**
- **This allows all the values of z to be computed in a single statement.**
- **If $f(x,y) = (x + y)^2$ then,**

```
Z = (X + Y).^2 % element by element
```

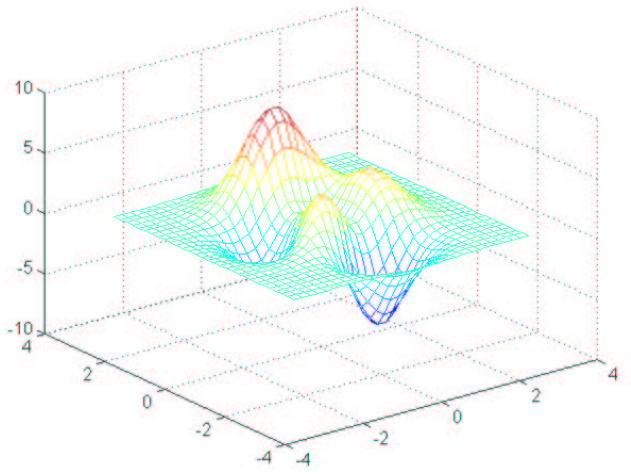
3-D Plotting

- **You can plot a mesh surface defined by the z-coordinates of points above a rectangular grid in the x-y plane.**

- **A mesh is formed by joining adjacent points with straight lines.**

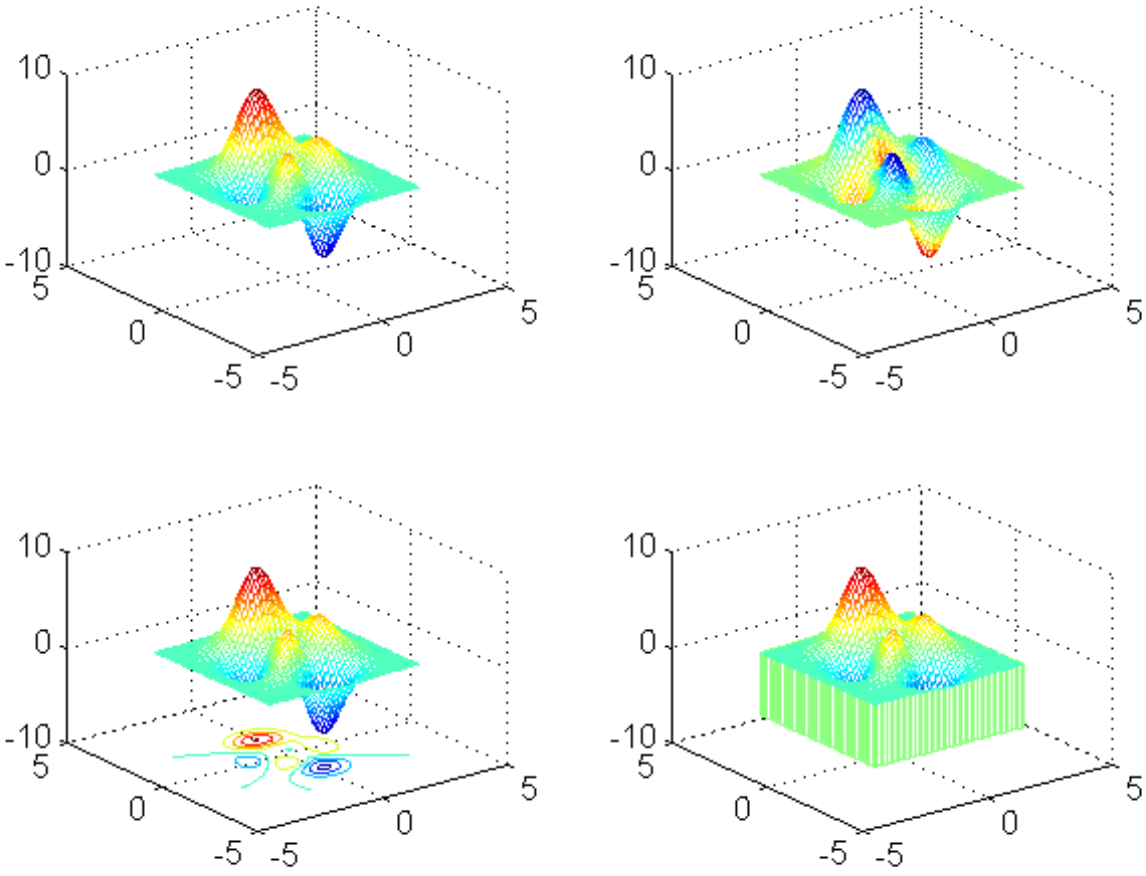
```
» [X,Y,Z]=peaks(30);
» mesh(X,Y,Z)
```

- **`peaks` is a function of two variables, obtained by translating and scaling Gaussian distributions.**



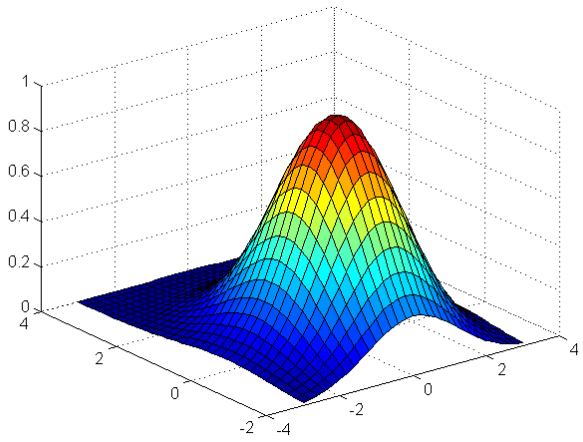
Some 3-D Plot Examples

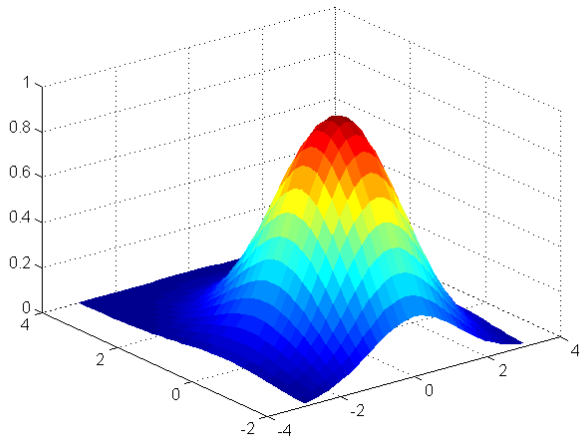
```
[x,y,z]= peaks;
subplot(2,2,1),mesh(x,y,z)
C = del2(z);
subplot(2,2,2),mesh(x,y,z,C)
subplot(2,2,3),meshc(x,y,z)
subplot(2,2,4),meshz(x,y,z)
```

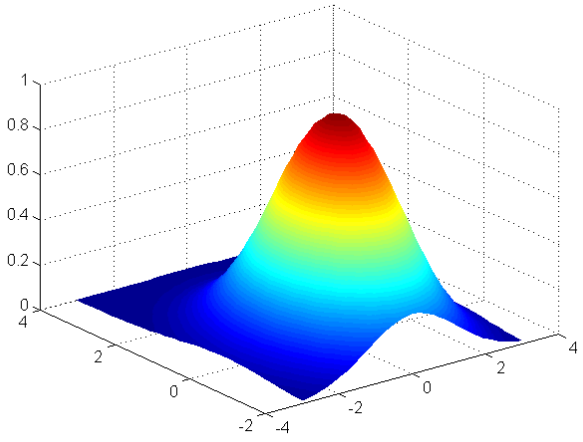


Different Shading Pattern

```
[X,Y]=meshgrid(-3:.2:3,-2:.2:4);
Z=exp(-(X.^2+Y.^2)/3);
subplot(1,1,1)
surf(X,Y,Z)
% plot lines with color of
  quadrilateral
shading flat
% interpolate shading across
  quadrilaterals
shading interp
% return to original shading
shading faceted
```







3-D PLOTS – OTHER USEFUL FUNCTIONS

- **Add labels, titles, change axes as with 2-D plots**

- **Can look at contours of the surface using `contour` function.**

- **You can rotate the surface plot using the toolbar button or `rotate3d`**

Visualizing Volume

- **Visualizing volume is the representation of data that are defined on 3-D grids:** ***f(x,y,z)***

- **Volume data sets have multidimensional arrays of scalar or vector data defined on lattice structures.**

- **We will look only at scalar data.**

- **Example of scalar data might be air pressure or temperature at a points in space.**

Visualizing Volume

- **Scalar volume data is best viewed with isosurfaces, slice planes, and contour slices.**
- **You can view the distribution of data within a volume by mapping values to colors using slice planes.**
- **Contour slices are contour plots drawn at specific coordinates and let you see where in a plane the data values are the same.**
- **Isosurfaces are surfaces created by using points of equal value as vertices of patch objects.**

Contourslice

```
load mri

%remove empty
%dimension

D = squeeze(D);

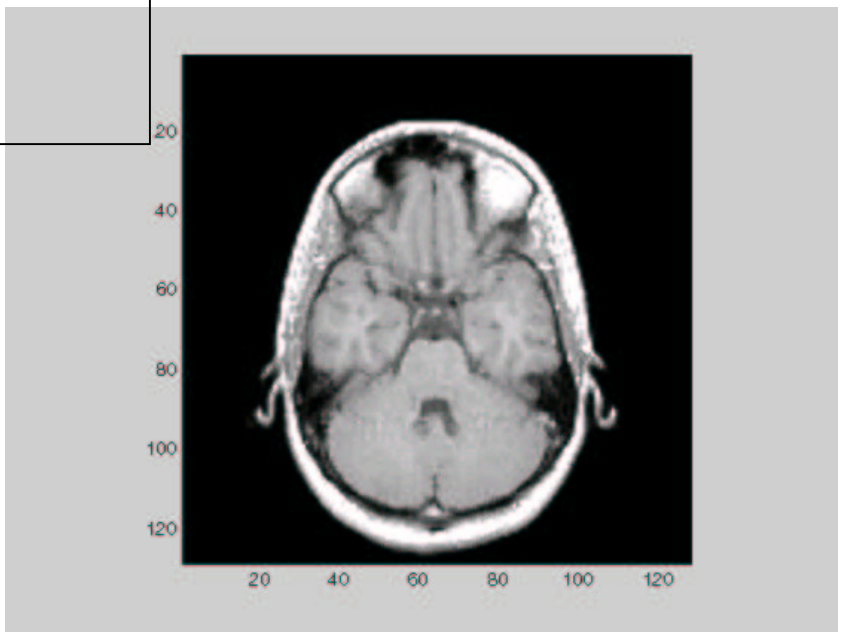
x=xlim;

y=ylim;

contourslice(D,[],[],8)

axis ij,xlim(x),ylim(y)

daspect([1 1 1])
```



ISOSURFACES

- **Use `isosurface` to display overall structure of a volume.**
- **You can combine it with `isocap`.**
- **This technique can reveal information about data on the interior of the `isosurface`.**
- **The following will create and process some volume data and create `isosurface` and `isocap`.**
- **To add some other effects, lights will be added.**
- **`isocap` indicate values above (default) or below the value of the `isosurface`**

ISOSURFACES

- **Try this example, generating uniform random numbers:**

```
data = rand(12,12,12);

data = smooth3(data,'box',5);

isoval = 0.5;

H=patch(isosurface(data,isoval),...
   'FaceColor','blue','Edgecolor','none',...
  'AmbientStrength',.2,'SpecularStrength',0.7,...
   'DiffuseStrength',.4);

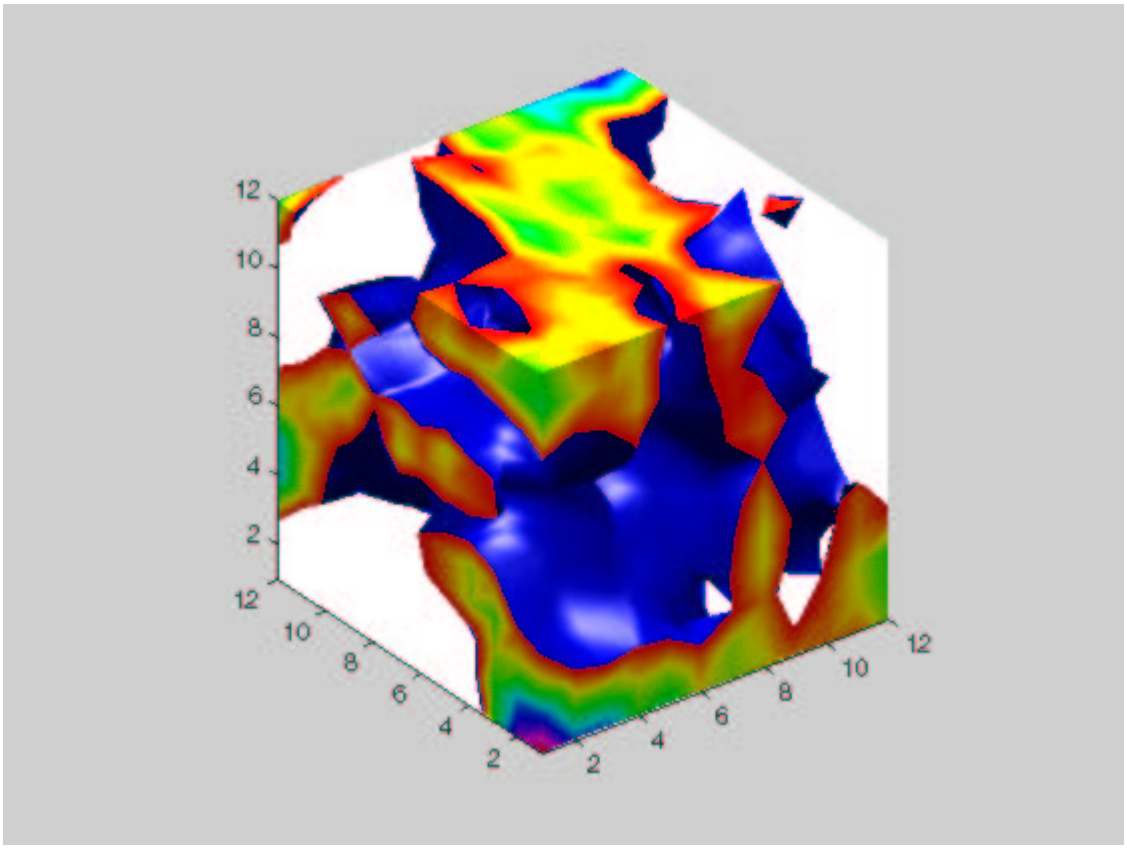
isonormals(data,H)  %produces smoother lighting
patch(isocaps(data,isoval),...
   'FaceColor','interp','Edgecolor','none');

colormap hsv

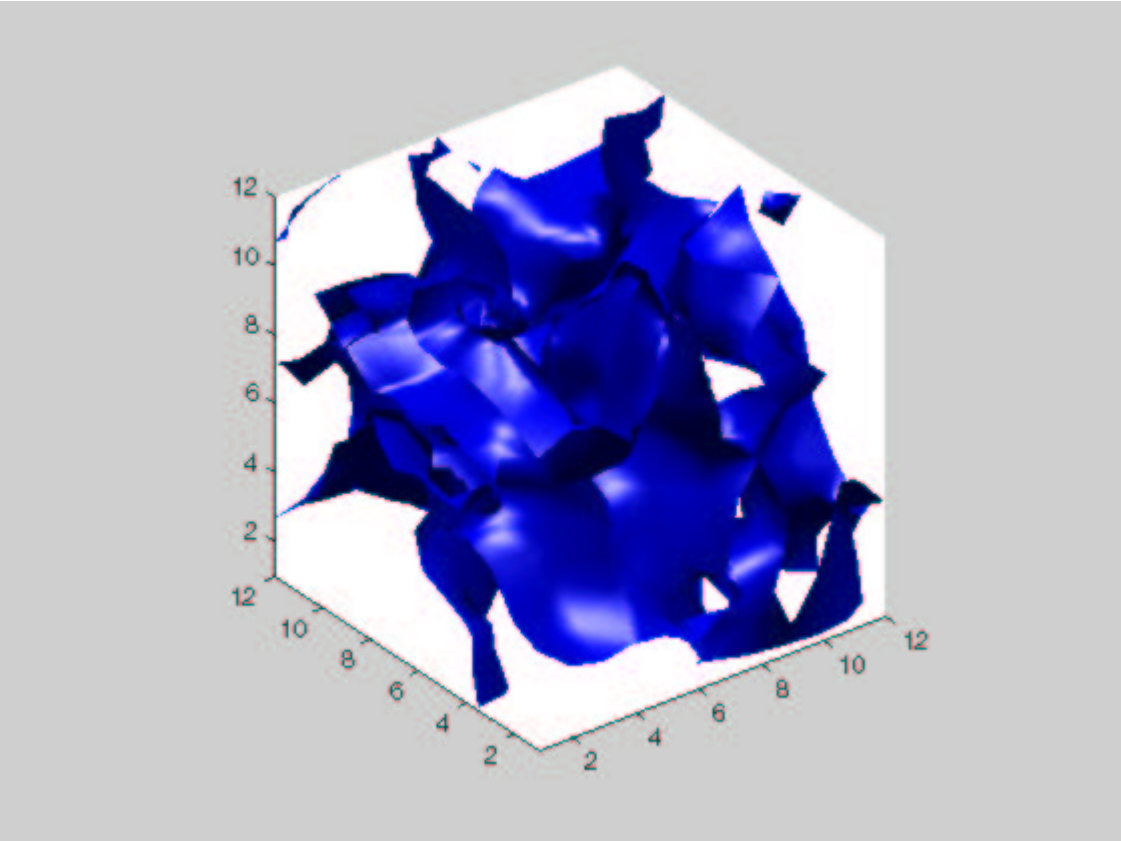
daspect([1 1 1]),axis tight, view(3)

camlight right, camlight left, lighting phong
```

ISOSURFACES



ISOSURFACES Without Caps

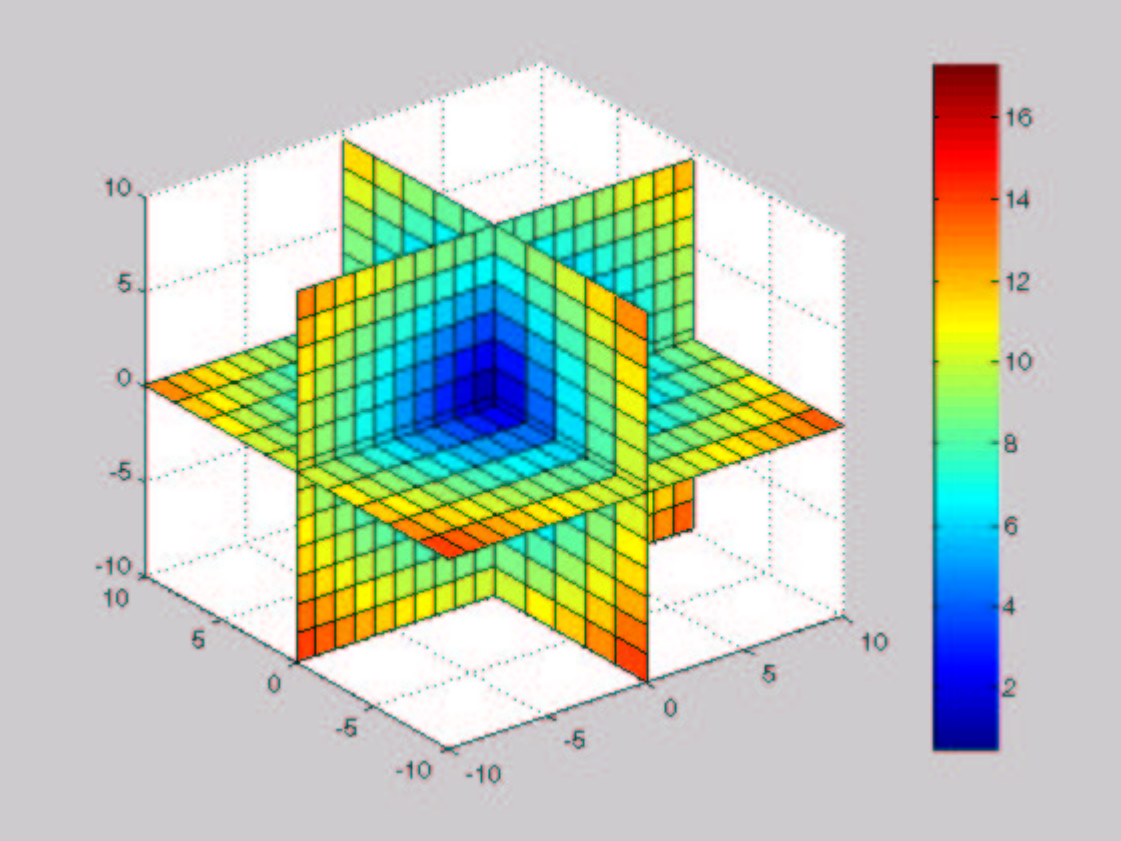


Slice

- **`slice` displays orthogonal slice planes through volumetric data.**
- **Color indicates the scalar value.**
- **Example:**

```
[x,y,z] = meshgrid(-10:10, -10:2:10, -
  10:1.5:10);
v = sqrt(x.^2 + y.^2 + z.^2);
% slice through the 0 planes
slice(x,y,z,v,0,0,0)
colorbar
```

Slices



Many Others Graphics are Available

- **Texture mapping to a surface**
- **Images**
- **Animation – movies**
- **Animation – on-the-fly**
- **Lighting**
- **Camera graphics**

Handle Graphics

- **Handle Graphics**
- **When do you need to use Handle Graphics**
- **What are Handle Graphics objects.**
- **Object Handles**
- **Object Properties**

Handle Graphics

- **Handle Graphics is the collection of low-level graphics functions that actually do the work of generating graphics in MATLAB.**

- **These details are usually hidden from the user in graphics files such as plot, axis, etc.**

- **Handle Graphics can be used to make a small change or global changes that affect all graphical output.**

- **We will cover only the highlights of Handle Graphics.**

- **You are encouraged to refer to Chapter 31 of *Mastering MATLAB 5*, if you will be using these capabilities.**

- **Appendices B - J contain lists of object properties.**

- **The MATLAB Help Desk is an excellent resource for information on Handle Graphics.**

Handle Graphics

- **Who needs Handle Graphics?**
- **When you must have more control over your plots.**
- **When you need to change objects in your graphics that you cannot do with the high-level plot functions.**

- **Handle Graphics Objects**
- **Every component of a graph is an object: axis, text, lines, etc.**
- **Each object has a handle associated with it.**
- **A handle is a number that identifies the object.**
- **Each object has properties that can be changed: color, position, etc.**

- **In MATLAB, a graphics object is a component of a graph that can be manipulated.**

Handle Graphics

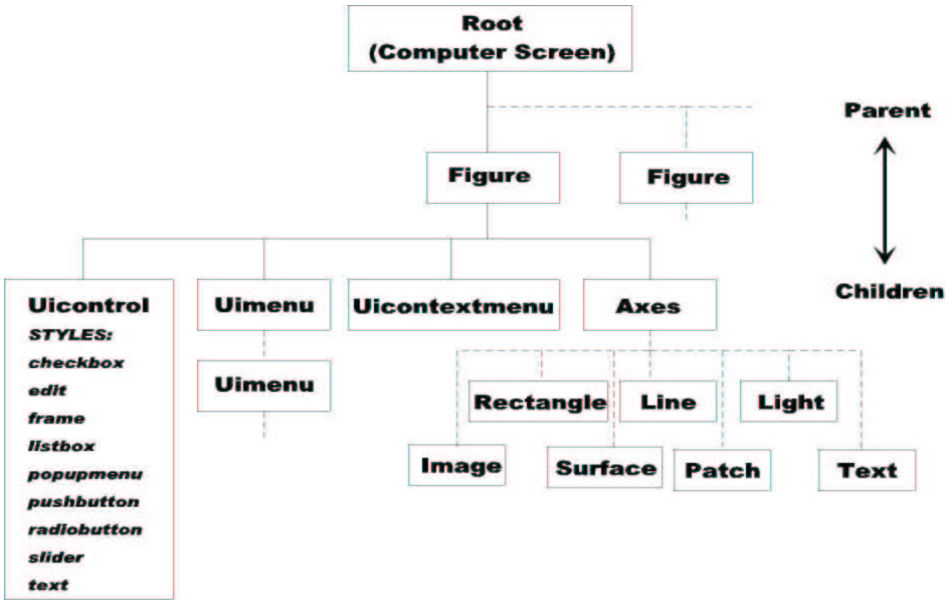
- **Everything created by a graphics command is a graphics object.**

- **Examples:**
  - **Figure windows**
  - **Axes**
  - **Lines**
  - **Surfaces**
  - **Text**

- **These are arranged in a hierarchy of parent and child objects.**

- **The computer screen is the root object and is the parent of all other objects.**

- **Figures are the children of the root.**



HANDLE GRAPHICS

- **Axes and GUI (uicontrol, uimenu, uicontextmenu) objects are children of figures.**

- **Line, text, surface, patch and image objects are children of axes.**

- **The root can contain one or more figures.**

- **Each figure can contain one or more sets of axes.**

- **All functions that create an object will create the parent if they do not exist.**

HANDLE GRAPHICS

- **Recall that an object is identified by a handle. When an object is created, a unique handle is created for it.**

- **The handle of the root object or computer screen is always zero.**

- **Figure handles are usually integers, which are displayed in the window title bar.**

- **Other object handles are floating-point numbers.**

- **You can create a figure object and save its handle in a variable using the following:**

- **Hf_fig = figure**

- **For example, `figure` creates a figures window and saves its handle in the variable Hf_fig.**

HANDLE GRAPHICS

- **There are several MATLAB commands that can be used to determine the handles of figures, axes and other graphics objects.**
-
- **gcf is a function that ‘gets current figure’ handle.**
-
  - **Hf_fig = gcf returns the handle of the current figure and assigns it to the variable.**
-
- **gca is a function that ‘gets current axes’ handle.**
-
  - **Ha_ax = gca returns the handle of the current axes in the current figure and assigns it to the variable.**
-
- **gco is a function that ‘gets current object’ handle.**
-

  Hx_obj = gco returns the handle of the current object (the last object clicked on by the mouse) in the current figure.

HANDLE GRAPHICS

- **You should follow a naming convention for handle variables.**

- **In the *Mastering MATLAB* book, each handle variable starts with the letter 'H'.**

- **You should also use a naming convention that describes the type of object referred to by the handle.**

- **Whatever convention you decide to use should facilitate handle recognition.**

- **You do not need to save handles for objects unless you think you will need to change the properties of those objects later on.**

- **It is important to save the handles that have floating point values, because these follow the full precision of MATLAB.**

Handle Graphics

- **All graphics objects have properties that define their characteristics:**

Position

Color

Size

...

- **You can manipulate your graphics by changing these properties.**

- **The properties for each object are unique.**

- **Some properties are valid for all objects.**

- **Object properties have property names and their values.**

HANDLE GRAPHICS

- **The properties are usually displayed with letters in mixed case, with the first letter of each word capitalized. However, MATLAB recognizes a property regardless of case.**

- **For example: 'LineStyle'**

- **You only need to use enough letters to uniquely identify the property.**

- **For example: 'Position' and 'Pos' and 'pos' would access the *position* property.**

- **When an object is created, it has a set of default property values.**

- **You can set or change these at creation time, by arguments to the object creation function.**

HANDLE GRAPHICS

- **For example,**
- `figure('Color', 'white')`
- **changes the background color from gray to white.**
- **You can also change properties later on using the following two functions. These are the main functions for manipulating graphics object properties.**
- `get` **returns the current value of an object property.**
- `set` **allows you to change the values of object properties.**
- **The general syntax is:**

```
set(handle,'PropertyName',Value,...)
get(handle,'PropertyName',Value,...)
```

- **You can set several of these property values in one command.**

HANDLE GRAPHICS

- **Example:**

```
figure
set(gcf,'Color', 'white')
```

- **You can use set(handle,'PropertyName') to get a list of values that can be used for the object referred to by handle.**

- **If you use set(handle), then you get a list of properties and possible values (if appropriate) for the object belonging to the handle.**

- **If you use the function get(handle) then it lists the properties and current values for the object.**

- **If you want a specific value for a property, use `get(handle, 'PropertyName')`.**

- **Example:**

```
pos = get(gcf, 'Position')
```

HANDLE GRAPHICS

- **You can use Handle Graphics to change the printed output of your graphics.**

- **For example, you can use it to orient the page (landscape or portrait) or figure placement.**

- **Recall that MATLAB sets object properties to their default values when it is created.**

- **You can change these by using a special property name consisting of `'Default'` followed by the object type and property name.**

- **For example**

`'DefaultFigureColor'.`

HANDLE GRAPHICS

- **You should take care when changing defaults. If you change a default in a function or other file, then always save the previous settings using the `get` command and restore them when you are done.**

- **You can use the property-value `'remove'` to reset a property back to the original defaults:**

```
set(gcf,'DefaultAxesLineWidth','remove')
```

- **You can use the MATLAB default temporarily using the property-value 'factory'.**

- **This changes the default for the current command only.**

HANDLE GRAPHICS

- **RECALL:**

- **Any object that appears in a MATLAB figure is a part of Handle Graphics.**

- **Every object has a unique identifier called a ‘handle’.**

- **This handle allows you to modify the object.**

- **Most of the time you do not need to worry about these, but they are always there in the background.**

- **The Property Editor in *MATLAB* is one of the GUIDE tools.**

- **It is very useful in GUI development or for editing graphics objects.**

- **The Property Editor allows you to change object properties without knowing their handle or using the MATLAB command line.**

PROPERTY EDITOR

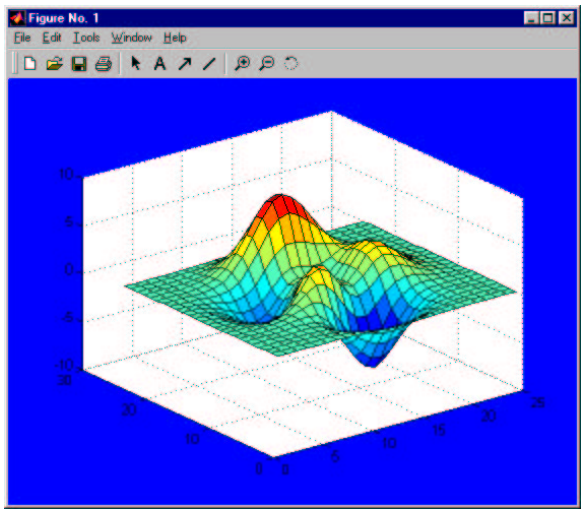
- **First, lets review how to edit an object without the editor.**

- **Create a figure using**

```
» surf(peaks(25))
» set(gcf,'Color','blue')
```

- **These commands created a surface picture of the ‘peaks’ function and set the background color to blue.**
- **You can also change the color (to green) using**

```
» set(gcf,’Color’,[0 1 0])
```



Properity Editor

- **The Property Editor provides convenient access to many properties of objects in a graph.**

- **You can edit these objects:**

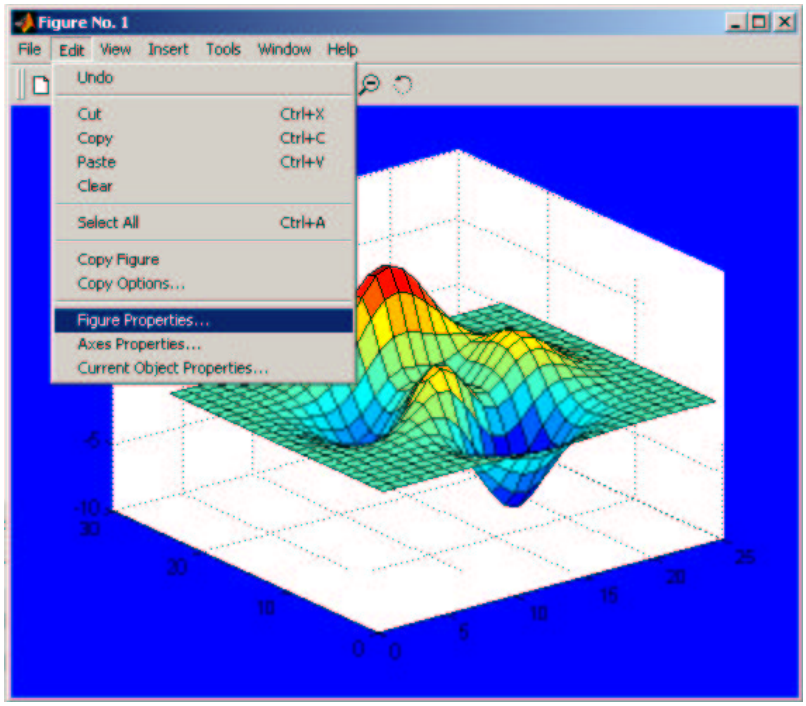
- **Figures**
- **Axes**
- **Lines**
- **Lights**
- **Patches**
- **Images**
- **Surfaces**
- **Rectangles**
- **Text**
- **Root Object**

- **To start the Property Editor, use the command**

```
» propedit(gcf)
```

Property Editor

- **You can also start the Property Editor selecting it from the pull-down menu in the figure window.**



If you place the cursor over a field, a data tip will appear that displays the name of the property and its current value.

- **If you keep the Property Editor open, clicking on other objects in the graph will change the set of panels to those associated with that object type.**

- **Simply click on the tab of the panel that contains the property you want to modify.**

- **After changing a value of a property, click the Apply button to make your change permanent.**

PROPERTY EDITOR

- **If you select multiple objects of the same type, the Property Editor displays the set of panels specific to that object type.**

- **Having selected multiple objects of the same type, when you change one value it will be applied to all objects of that type.**

- **If you select multiple objects of different types, the Property Editor will only display the Info panel, since it is common to all object types.**

- **You can also select objects using the Navigation Bar.**

- **Here you can see a hierarchical list of all objects in the current figure.**

- **You can use the navigation bar to search for a particular object, or group of objects, in a figure.**

- **Tag**

- **Type**

- **Handle**

Property Editor – Creating Tags

- **The navigation bar will list all objects by their type and their tag, if they have one.**
- **Tags can help identify which object in a list is being acted on.**
- **You can easily create a tag for an object.**
- **With Plot editing mode enabled, double-click on the object in a graph.**
- **Click on the Info tab in the Property Editor**
- **Enter a text string in the Tag field.**
- **Click on Apply**
- **Try changing the color of the background using the property editor**