Matlab Array and Matrix Manipulations and Graphics

Dr. Antonio A. Trani
Dept. of Civil and Environmental Engineering

Objectives of the Handout

- To illustrate examples of matrix manipulation in MATLAB
- To learn some of the basic plotting functions in MATLAB
- Just for the fun of learning something new (the most important reason)

Basic Matrix Manipulation

- Matlab basic rules are derived from Linear Algebra

Let $A = \begin{bmatrix} 4 & 3 & 4 \\ 4 & 6 & 8 \\ 3 & 6 & 6 \end{bmatrix}$ and $b = \begin{bmatrix} 35 \\ 22 \\ 40 \end{bmatrix}$

```
A = [4 3 4; 4 6 8; 3 6 6];
b = [35 22 40]';
y = A*b;
```

Results in column vector y,

```
y =
   366
   592
   477
```

Example # 1: Solution of Linear Equations

- Linear equations are important in many engineering problems (optimization, structures, transportation, construction, etc.)

Suppose we want to solve the set of linear equations:

$$4x_1 + 3x_2 + 4x_3 = 35$$

$$4x_1 + 6x_2 + 8x_3 = 22$$

$$3x_1 + 6x_2 + 6x_3 = 40$$

Then in matrix form we have:

$$Ax = b$$

Example # 1: Solution of Linear Equations

where:

$$A = \begin{bmatrix} 4 & 3 & 4 \\ 4 & 6 & 8 \\ 3 & 6 & 6 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } b = \begin{bmatrix} 35 \\ 22 \\ 40 \end{bmatrix}$$

Using MATLAB this can be solved using the operator \

$$x = A \backslash b$$

"Backslash" operator

% Solution of linear equations

```
A = [4 3 4; 4 6 8; 3 6 6]; b = [35 22 40]';x = A\b;
```

Example # 1: Solution of Linear Equations

Yields the following answer for x,

```
x =
   12.0000
   15.6667
  -15.0000
```

% Another solution of the linear equations

```
A = [4 3 4; 4 6 8; 3 6 6]; b = [35 22 40]';

x = inv(A)*b;
```

This gives the same result taking the inverse of A

Array vs. Matrix Operations

- MATLAB differentiates between array and matrix operations
- **Matrix operations** apply to matrices using Linear Algebra rules (hence also called Scalar operations)
  - An example of this is solving a set of linear equations as shown in the previous example
- **Array operations** apply when you want to do element by element calculations on a multi-dimensional array
  - An example of this is calculating the deflection of a cantilever beam problem as shown next

Examples of Matrix Operations

Let matrix A = [3 3 3 ; 2 2 2; 1 1 1] and B = [3 4 5]’

Valid matrix operations are:

c = A ^ 2

d = A * A

e = A *B

f = A * 3

g = A + 5

Array Operations Nomenclature

- Array operators have a period in front of the operand (e.g., .*)
- For example:

```
x = 0:0.05:8;
y = sin(x^2)*exp(-x);
```

Creates a vector x with cell values from 0 to 8 at steps 0.05

- Will not execute correctly because the manipulation of array x requires a period in front of the * and ^ operands
- The following statements will work:

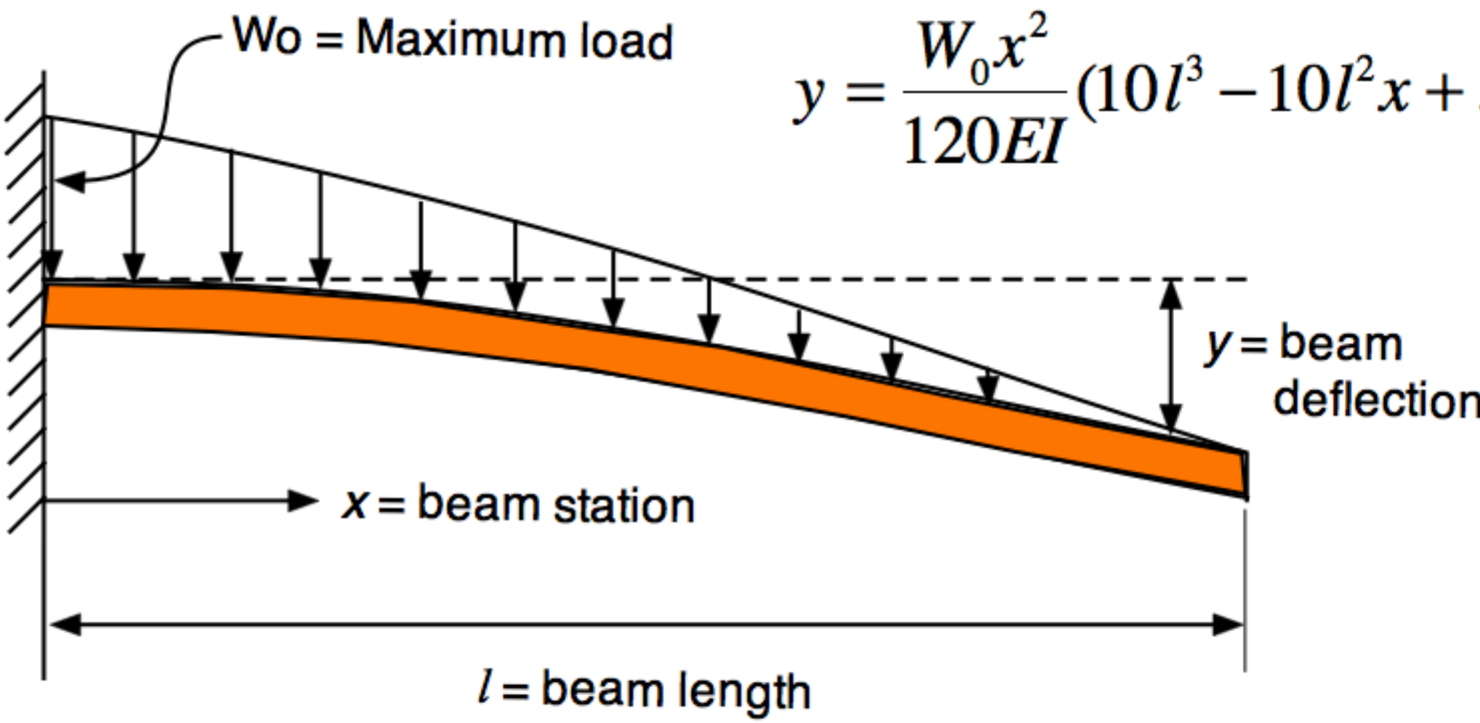
```
x = 0:0.05:8;
y = sin(x.^2).*exp(-x);
```

Note the (.) in front of the operands

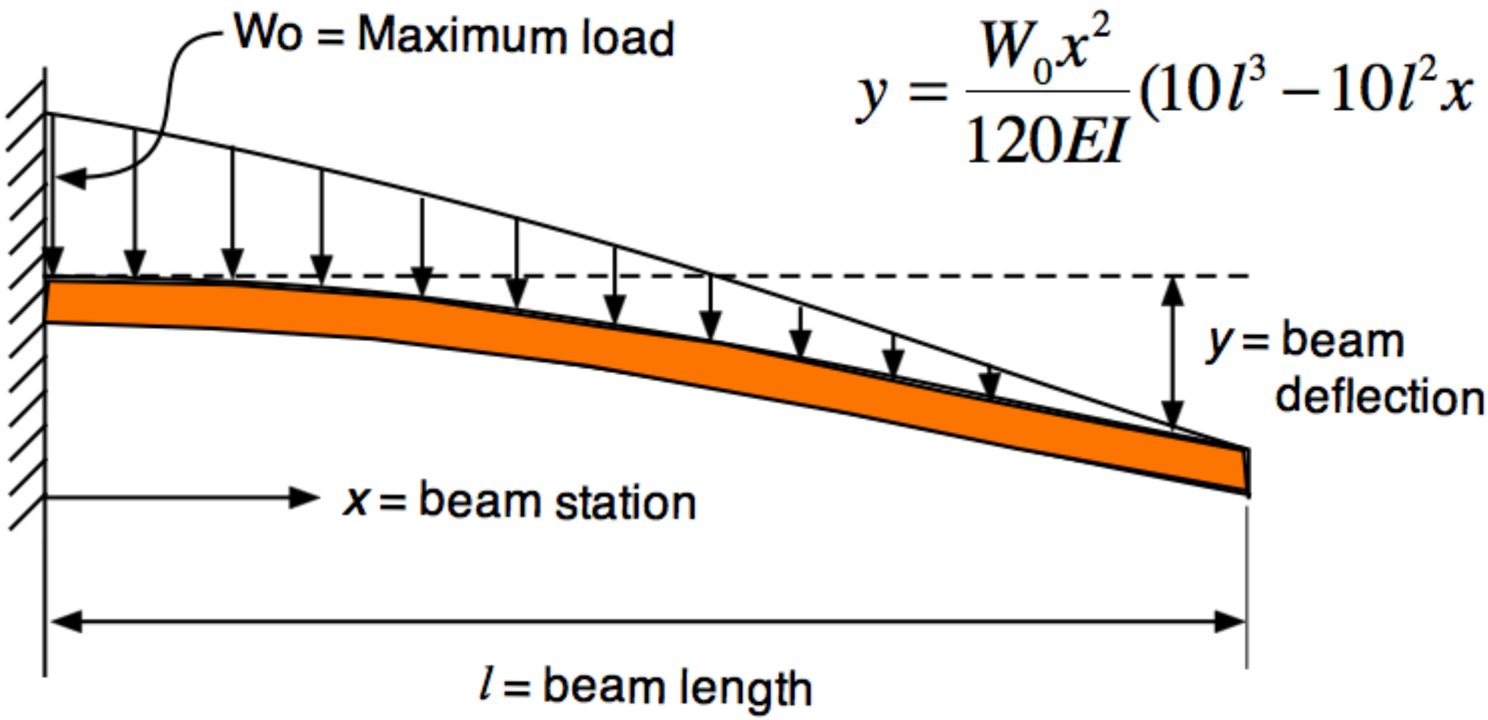
Example # 2 : Cantilever Beam Calculations

- A cantilever beam with a uniformly varying load is shown below
- We would create a simple Matlab script to estimate the beam deflection for any station (x) along the beam
- The formula to estimate the deflection is:

$$y = \frac{W_0 x^2}{120EI}(10l^3 - 10l^2 x + 5lx^2 - x)$$



Example # 2 : Cantilever Beam Calculations (cont.)



$$y = \frac{W_0 x^2}{120EI}(10l^3 - 10l^2 x + 5lx^2 - x)$$

$y =$ deflection of the beam at station x (m)

$E =$ Young's modulus (N/m^2)

$I =$ bean moment of inertia (m^4)

$x =$ beam station (m)

Example # 2 : Cantilever Beam Matlab Script

```
% Scrip to calculate the deflection (y) of a cantilever beam
% subject to a linearly decreasing load (W)
% A. Trani (October 10, 2013)

% W = load at station x (N/m)
% Wo = maximum load at station x=0 (N/m)
% E = Modulus of elasticity (N/m-m)
% I = Moment of inertia (m-m-m-m)
% x = beam station = distance from datum point (wall) to any point on the
%       beam (m)
% l = beam length (m)
% y = beam deflection at any station (m)

% Deflection equation
% y = -Wo * x.^2 / (120*E * I * length_of_beam) .* (10 * length_of_beam^3 ...
%         - 10 * length_of_beam^2 .*x + 5 * length_of_beam .* x.^2 - x.^3);

% Beam properties

Wo = 6000;                        % Newtons/m
E = 200e9;                        % N/m-m - value for Steel = 200e9
I = 0.001;                        % meters to the fourth power
length_of_beam = 9;               % meters

x = linspace(0,length_of_beam,100);        % 100 points to the end of the beam

% Calculate deflection to the beam at any point in the beam length

y = -Wo * x.^2 / (120*E * I * length_of_beam) .* (10 * length_of_beam^3 ...
      - 10 * length_of_beam^2 .*x + 5 * length_of_beam .* x.^2 - x.^3);
```

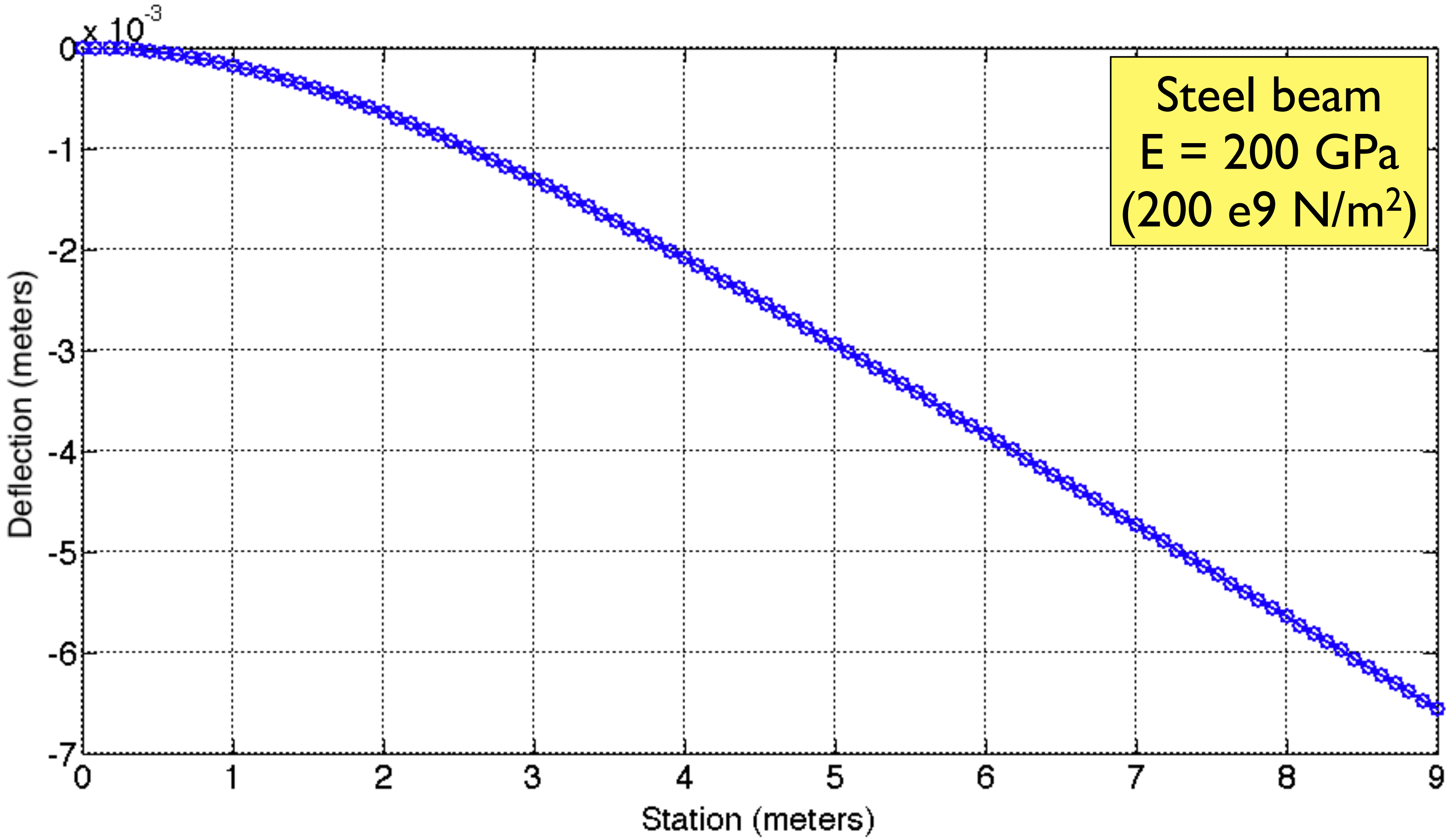
Example # 2 : Cantilever Beam Matlab Script (cont.)

```
% Calculate deflection to the beam at any point in the beam length

y = -Wo * x.^2 / (120*E * I * length_of_beam) .* (10 * length_of_beam^3 ...
    - 10 * length_of_beam^2 .*x + 5 * length_of_beam .* x.^2 - x.^3);

% Plot the deflection

figure
plot(x,y,'^b')
xlabel('Station (meters)')
ylabel('Deflection (meters)')
grid
```

Note the use of (.) in front of the operands

Ellipses is used in Matlab to indicate a continuation statement

Example # 2 : Cantilever Beam Output Plot (beam deflection)



Observations

```
25    x = linspace(0,length_of_beam,100);       % 100 points to the end of the beam
26
27    % Calculate deflection to the beam at any point in the beam length
28
29    y = -Wo * x.^2 / (120*E * I * length_of_beam) .* (10 * length_of_beam^3 ...
30          - 10 * length_of_beam^2 .*x + 5 * length_of_beam .* x.^2 - x.^3);
31
```

- A vector ***x*** is defined using the "**linspace**" function (linearly spaced vector) (see line 25 above)
  - **linspace** (starting point, ending point, no. of points)
- Since ***x*** is a vector with 100 elements, vector **y** (deflection) is automatically set by Matlab to have 100 elements
- The period before ***** and **^** operands is needed to tell Matlab to do element by element computations while calculating ***y***

Array Operators

Operation	MATLAB Operators
Array multiplication	.*
Array power	.^
Left array division	.\
Right array division	./
Matrix multiplication	*
Matrix power	^
Matrix division	/
Left matrix division	\

Use these to do basic operations on arrays of any size

Array Manipulation Tips

Always define the size of the arrays to be used in the program (static allocation)

- Define arrays with zero elements (defines statically array sizes and thus reduces computation time)

```
»d=zeros(1,3)
d =
    0    0    0

»c=ones(1,3)
c =
    1    1    1
```

Array Manipulation Tips

Sample of for-loop without array pre allocation

Example:

```
tic;
for i=1:1:10e6;
    d(i) = sin(i) ;
end
t=toc;
disp(['Time to compute array   ', num2str(t), ' (seconds)'])
```

Time to compute array   5.2982 (seconds)

Times calculated using a Mac Book Air (10.8.5 OS and i7 Processor)

Array Pre allocation

Array pre allocation saves time because MATLAB does not have to dynamically change the size of each array as the code executes

```
d=zeros(1,10e6);    % pre allocates a vector with zeros
tic;
for i=1:1:10e6;
    d(i) = sin(i) ;
end
t=toc;
disp(['Time to compute array   ', num2str(t), ' (seconds)'])
```

Time to compute array 1.395 (seconds)

Times calculated using a Mac Book Air (10.8.5 OS and i7 Processor)

Vector Operations in MATLAB

The following script is equivalent to that shown in the previous page.

```
tic;
i=1:1:10e6;
    d = sin(i);
t=toc;
disp(['Time to compute array   ', num2str(t), ' (seconds)'])
```

Time to compute array 0.90465 (seconds)

Note: MATLAB vector operations are optimized to the point that even compiling this function in C/C++ offers little speed advantage (10-15%).

Times calculated using a Mac Book Air (10.8.5 OS and i7 Processor)

Comparison of Results

The following table summarizes the array manipulation results

Procedure	CPU Time[a] (seconds)	Ratio[b]
Standard for-loop	5.29820	1.00
Array Pre allocation	1.39500	1.54
Vectorization	0.90465	5.85

a. Times calculated using a Mac Book Air (10.8.5 OS and i7 Processor)

b. Higher ratio means faster execution times

Vectorization Issues

- To illustrate with a numerical example instances where vectorization is not possible unless the problem is partitioned into two sub-problems
- Problem partitioning to speed up computations

Example # 3: Beam Problem

- Consider the following beam loading condition

$$R = W\left(\frac{3b^2L - b^3}{2L^3}\right)$$

$$R_1 = W\left(\frac{3aL^2 - a^3}{2L^3}\right)$$

At x: when $x < a$

$$V = R$$

At x: when $x > a$

$$V = R - W$$

At point of load:

$$M\,(\max) = Wa\left(\frac{3b^2L - b^3}{2L^3}\right)$$

At fixed end:

$$M_1\,(\max) = WL\left(\frac{3b^2L - b^3}{2L^3}\right) - W(L-a)$$

At x: when $x < a$

$$M = Wx\left(\frac{3b^2L - b^3}{2L^3}\right)$$

At x: when $x > a$

$$M = Wx\left(\frac{3b^2L - b^3}{2L^3}\right) - W(x-a)$$

At x: when $x = a = 0.414L$

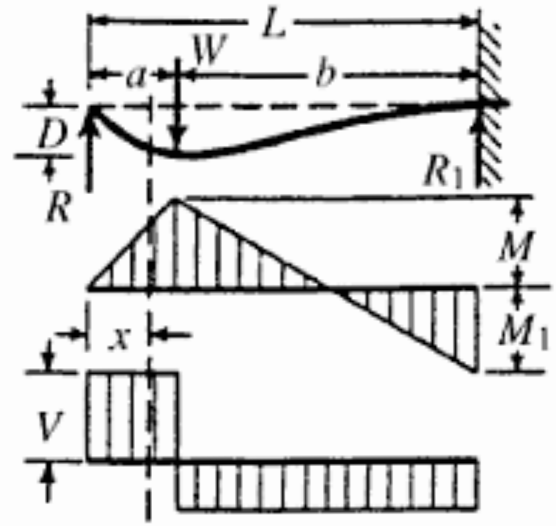
$$D\,(\max) = 0.0098\,\frac{WL^3}{EI}$$

At x: when $x < a$

$$D = \frac{1}{6EI}\left[3RL^2x - Rx^3 - 3W(L-a)^2x\right]$$

At x: when $x > a$

$$D = \frac{1}{6EI}\left[R_1(2L^3 - 3L^2x + x^3) - 3Wa(L-x)^2\right]$$



Observations

- The beam deflection and moment formulas change as the station changes from left to right (i.e., x< a or x> a)
- Handling two distinct formulas requires a branching statement (like an IF statement in the computations)

At x: when $x = a = 0.414L$

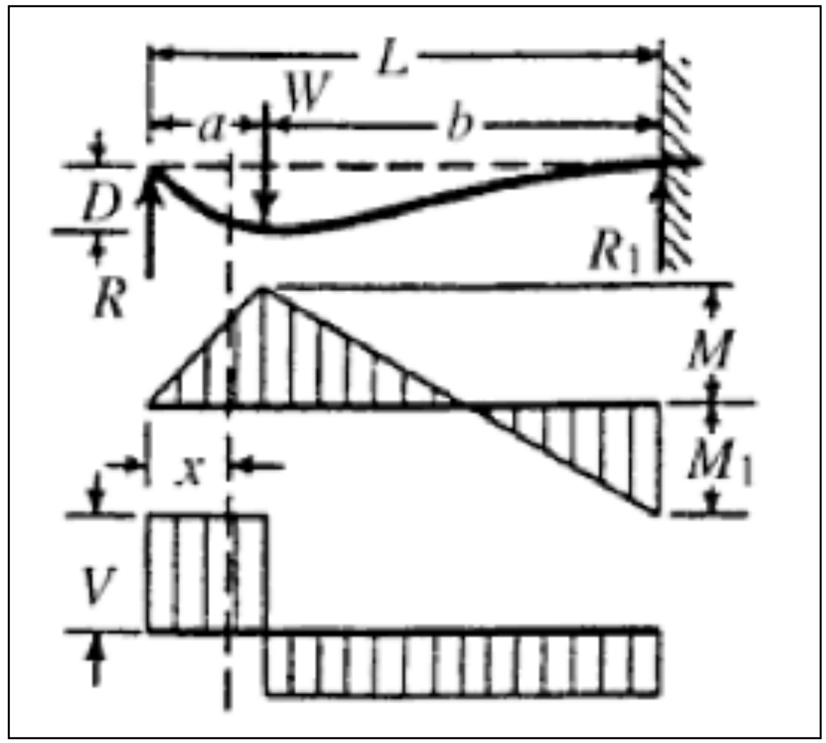
$$D\ (\max) = 0.0098\ \frac{WL^3}{EI}$$

At x: when $x < a$

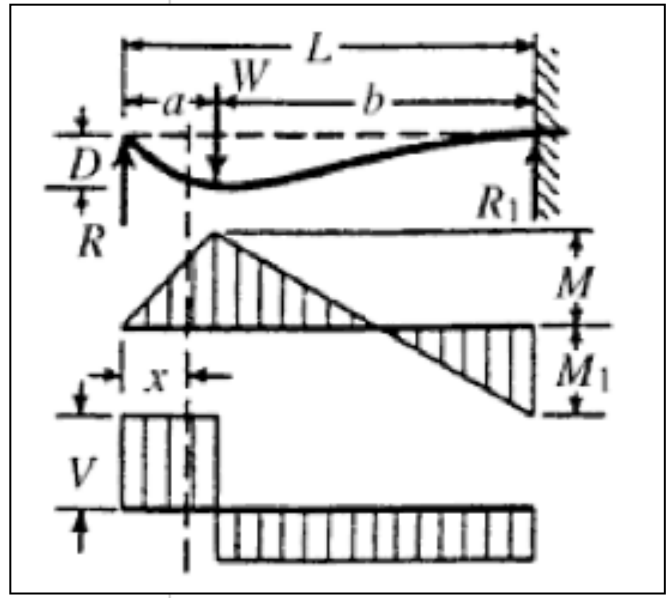
$$D = \frac{1}{6EI}\left[3RL^2x - Rx^3 - 3\ W\ (L-a)^2\ x\right]$$

At x: when $x > a$

$$D = \frac{1}{6EI}\left[R_1(2L^3 - 3L^2x + x^3) - 3\ Wa\ (L-x)^2\right]$$



Matlab Script (with Branching)

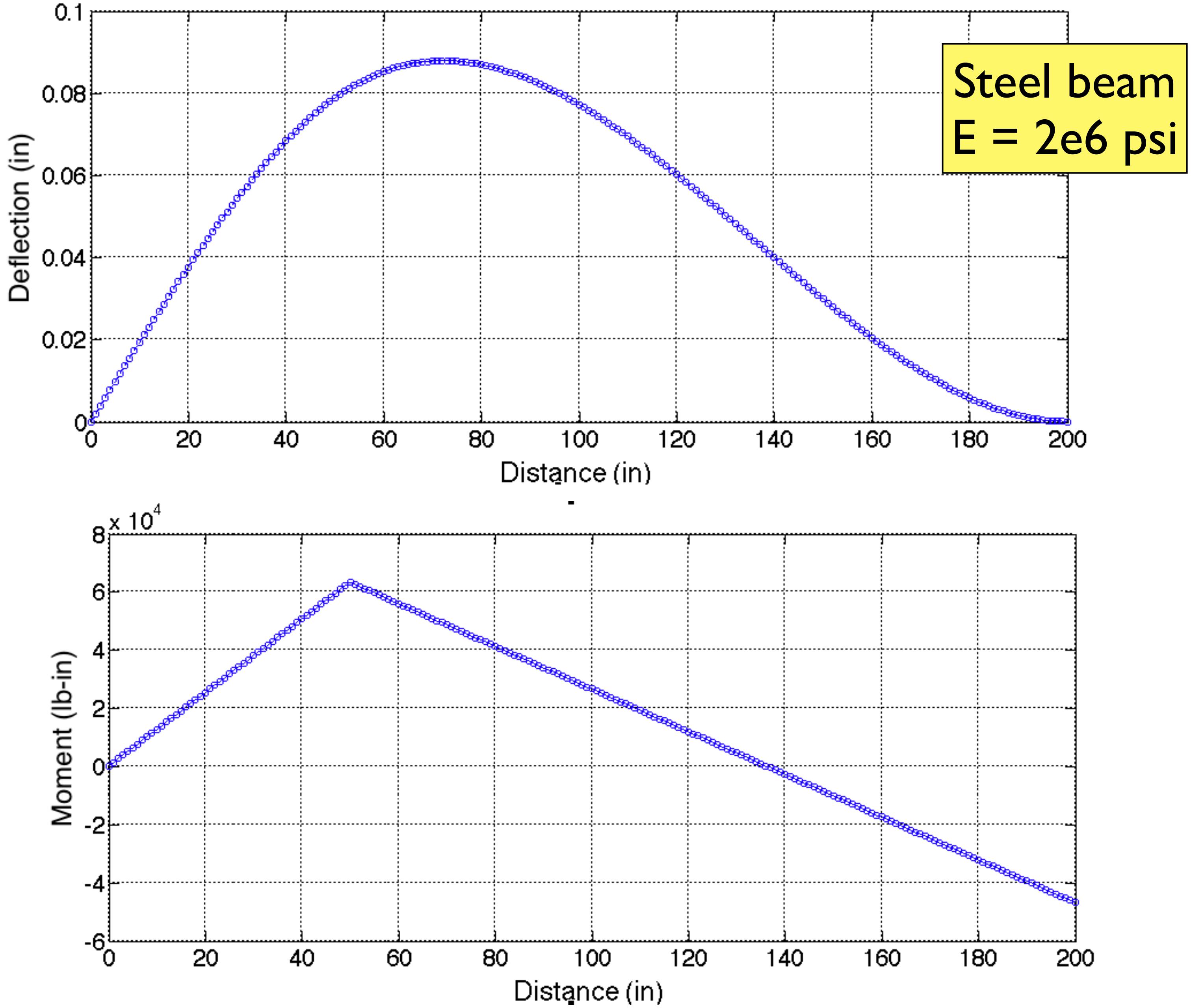



```
% Calculates the reaction and shear diagram of the beam
% beam is cantilever on one side and supported on the other side
% A. Trani (October 11, 2013)

% Given the following quantities:

% W = 2000 lb
% a = 50 in
% b = 120 in
% L = a + b
% I = 120 in-in-in-in
% E = 29e6 lbf/in-in

W = 2000;              % load in pounds
a = 50;                % left distance from load to single support (in)
b = 150;               % right distance from load to cantilever section (in)
L = a + b;             % total beam length (in)
I = 50;                % beam moment of inertia (in-in-in-in)
E = 29e6;              % Young's modulus (psi)

% Calculate the Reaction forces

R = W * (3*b^2*L-b^3) / (2*L^3);       % reaction at support
R1 = W * (3*a*L^2-a^3) / (2*L^3);      % reaction at wall
```

Matlab Script (Branching - cont.)

```
counter = 1;          % keeps track of the station count

% Start loop to compute Moments (M)

for x = 0:1:L
    if x < a
        Moment(counter) = W * x * (3*b^2*L - b^3) / (2*L^3);
        Deflection(counter) = 1/ (6*E*I) *(3*R *L^2 * x - R * x^3 - 3 *W *(L-a)^2 * x);
    else
        Moment(counter) = W * x * (3*b^2*L - b^3) / (2*L^3) - W*(x-a);
        Deflection(counter) = 1/ (6*E*I) *(R1 *(2*L^3 -3*L^2*x + x^3) - 3*W*a*(L-x)^2);
    end
    distance(counter) = x;
    counter = counter + 1;
end

figure
plot(distance,Moment,'o--')
xlabel('Distance (in)')
ylabel('Moment (lb-in)')
grid
```

Example # 3 : Beam Deflection



Graphs in MATLAB

There are many ways to build plots in MATLAB. Two of the most popular procedures are:

1) Using built-in MATLAB two and three dimensional graphing commands

2) Use the MATLAB Handle Graphics (object-oriented) procedures to modify properties of every object of a graph

Handle Graphics is a fairly advanced topic that is also used to create Graphic User Interfaces (GUI) in MATLAB. For now, we turn our attention to using Matlab built-in two and three graphics.

Plots Using Built-in Functions

MATLAB can generally handle most types of 2D and 3D plots without knowing Handle Graphics

- ‘plot’ command for 2D plots
- ‘plot3d’ for 3D plots
- Use ‘hold’ command to superimpose plots interactively or when calling functions
- Use the ‘zoom’ function to dynamically resize the screen to new [x,y] limits
- Use the ‘subplot’ function to plot several graphs in one screen

Basic Plots in MATLAB

Two-dimensional line plots are easy to implement in MATLAB

```
% Sample line plot
x=0:0.05:5;
y=sin(x.^1.8);
plot(x,y);                        % plot command

xlabel('x')                       % builds the x label
ylabel('y')                       % builds the y label
title('A simple plot')            % adds a title
grid                              % adds hor. and vert.
                                  % grids
```

Try this out now.

Other Types of 2-D Plots

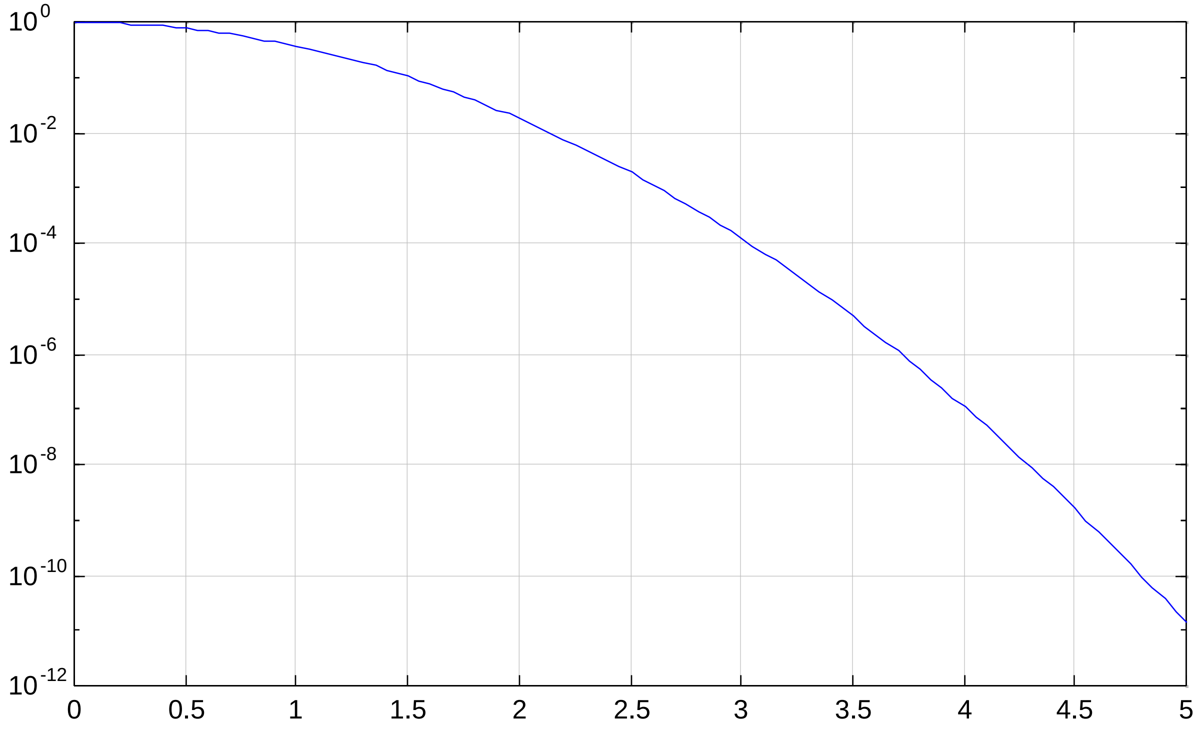
bar	bar plot
fplot	simple plot of one variable (x)
semilogx and semilogy	semilog plots
loglog	logarithmic plot
polar	polar coordinates plot
plotyy	dual dependent variable plot
errorbar	error bar plot
hist	histogram plot

More 2D Plots

stem	generates stems at each data point
stairs	discrete line plot (horizontal lines)
comet	simple animation plot
contour	plots the contours of a 2D function
quiver	plots fields of a function

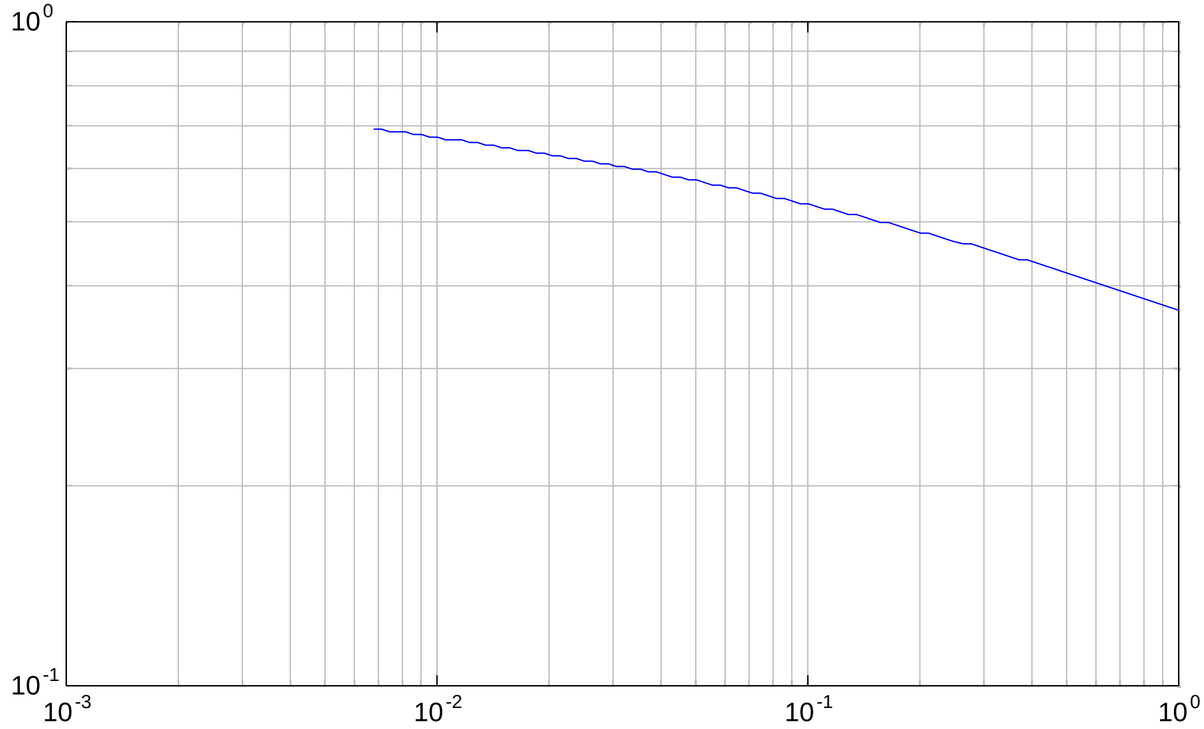
Sample 2D Plots (semilog plot)

```
x=0:0.05:5;
y=exp(-x.^2);
semilogy(x,y); grid
```



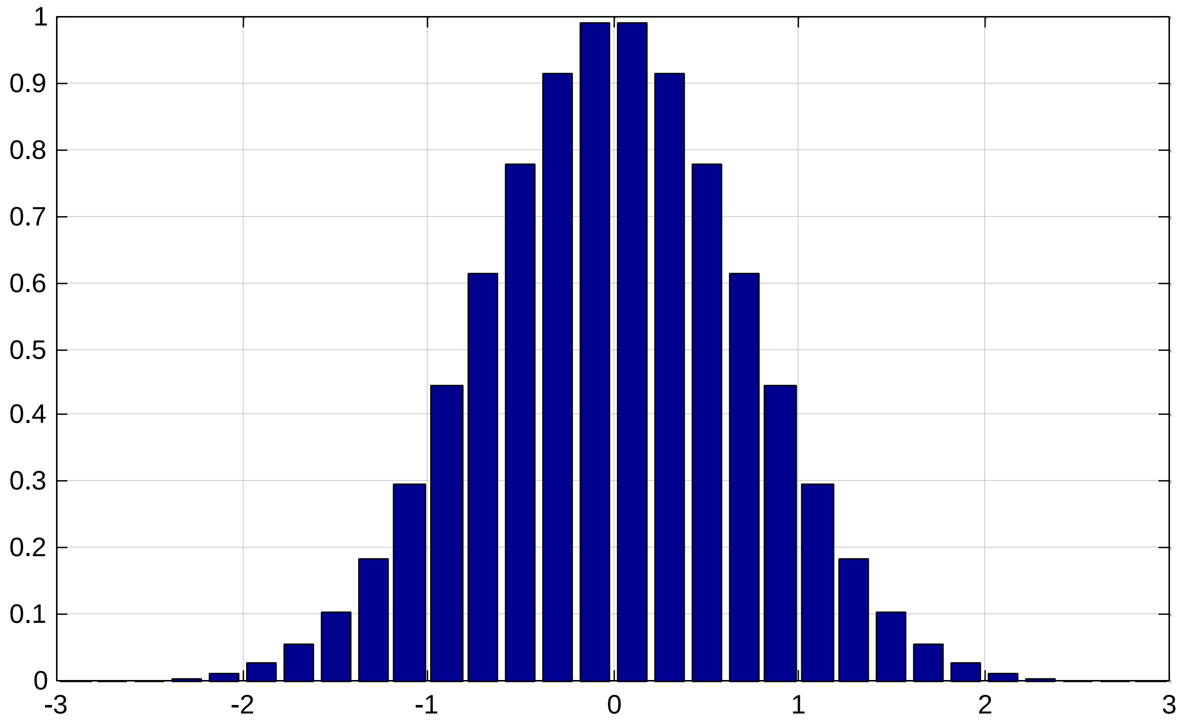
Sample 2D Plots (loglog plot)

```
x=0:0.05:5;
y=exp(-x.^2);
loglog(x,y); grid
```



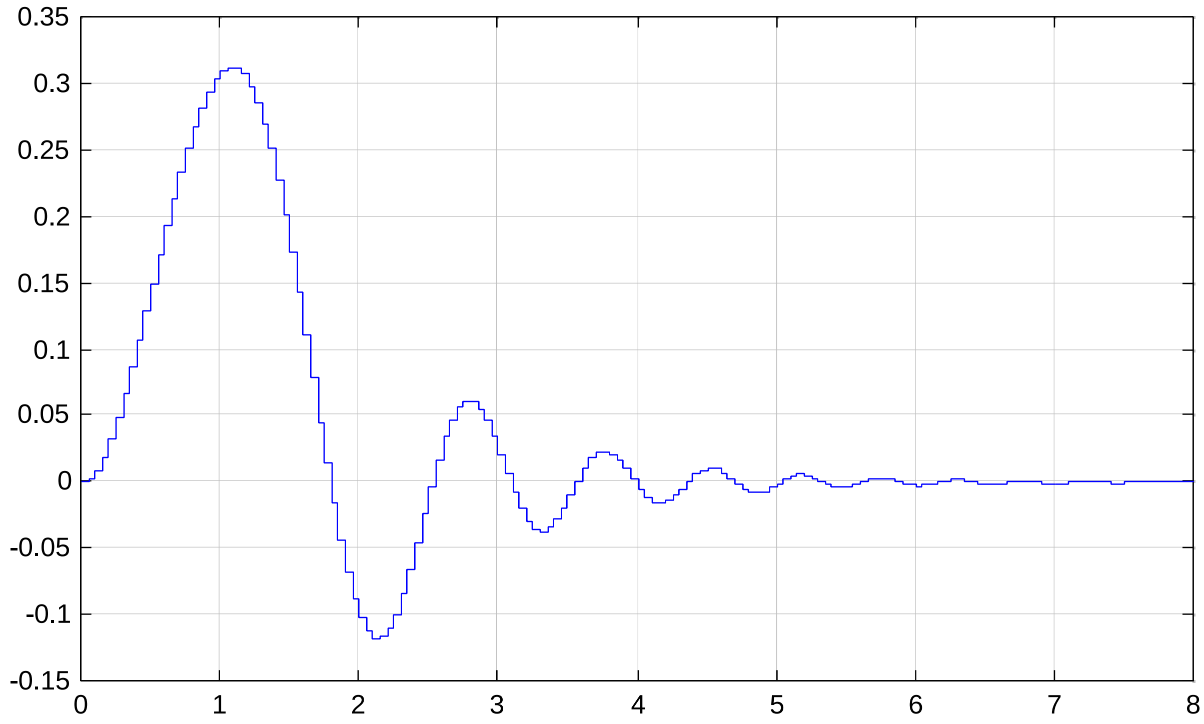
Sample 2D Plots (bar plot)

```
x = -2.9:0.2:2.9;
bar(x,exp(-x.*x));
grid
```



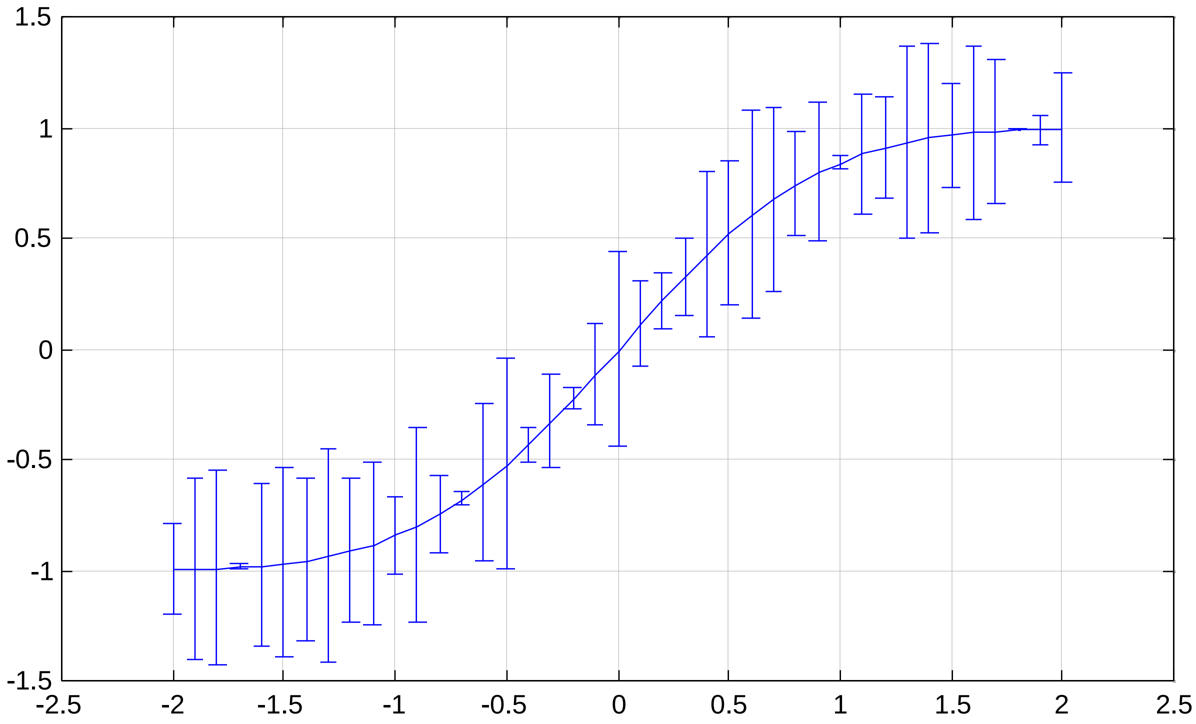
Sample 2D Plots (stairs plot)

```
x=0:0.05:8;
stairs(x,sin(x.^2).*exp(-x));
grid
```



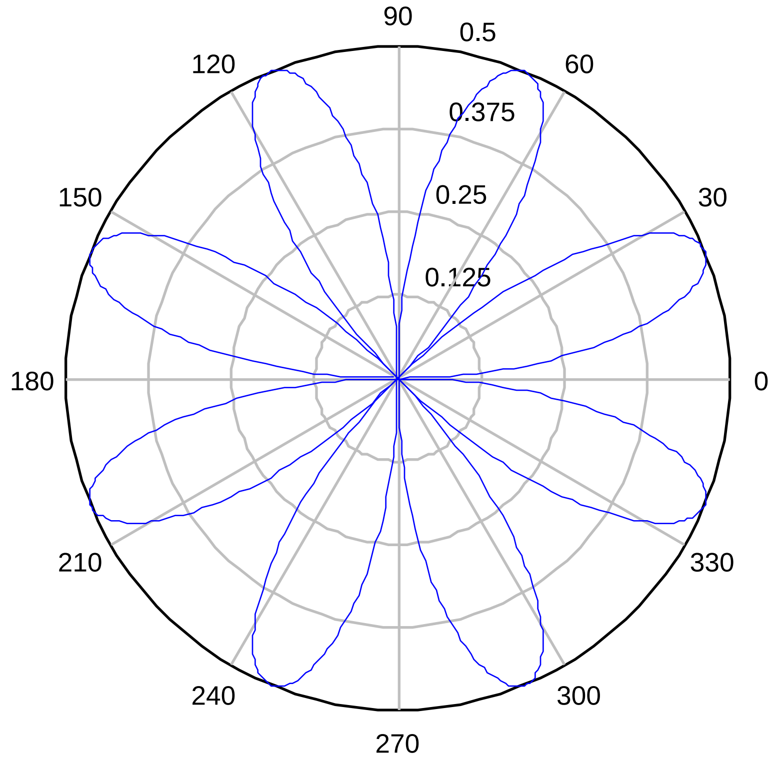
Sample 2D Plots (errorbar plot)

```
x=-2:0.1:2;
y=erf(x);
e = rand(size(x))/2;  errorbar(x,y,e);   grid
```



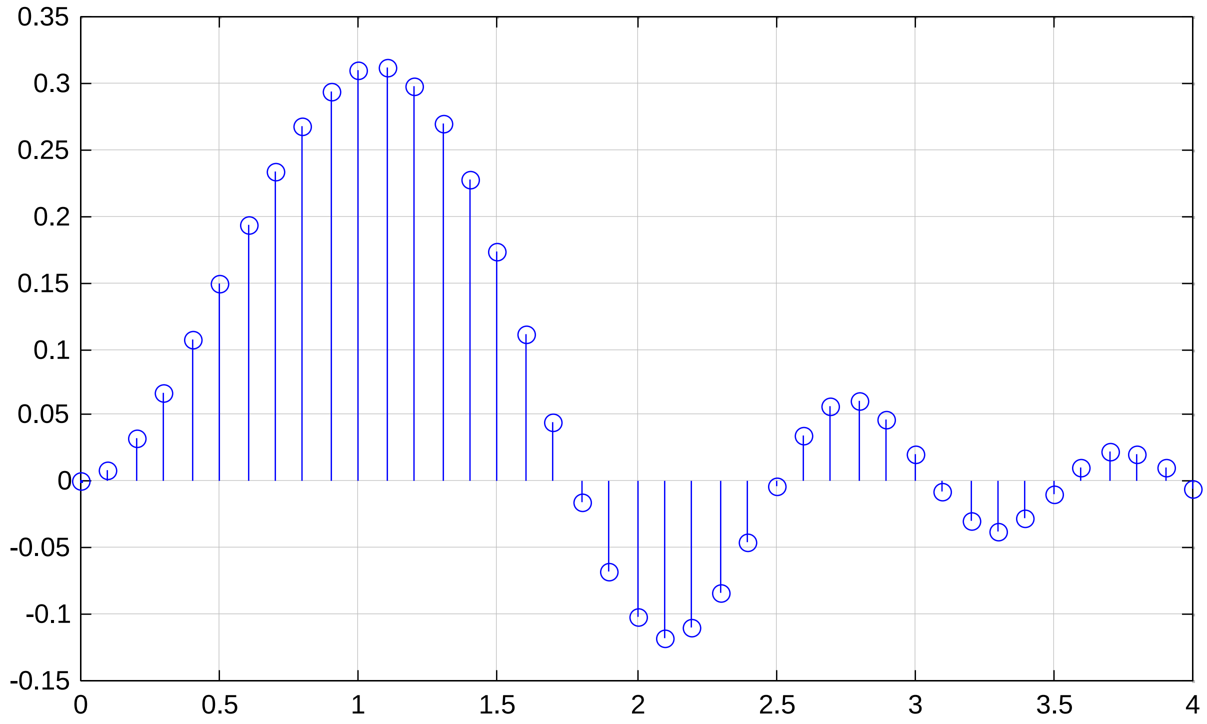
Sample 2D Plots (polar plot)

```
% Polar plot
 t=0:.01:2*pi;
 polar(t,sin(2*t).*cos(2*t));
```



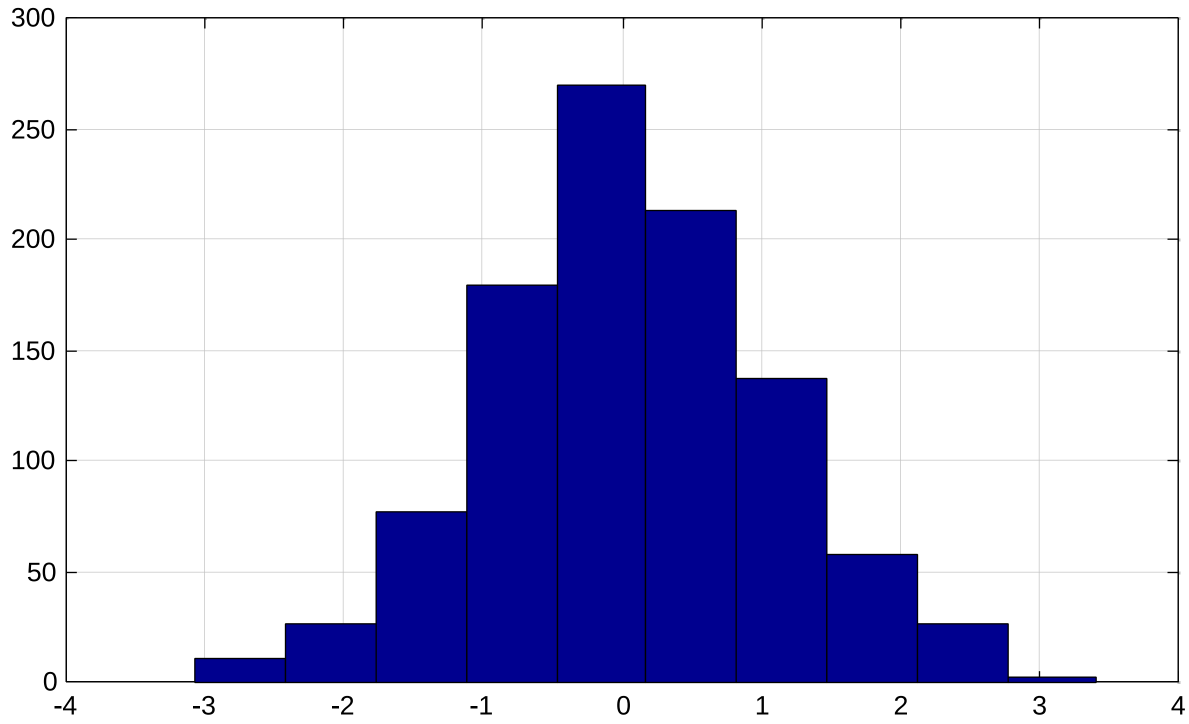
Sample 2D Plots (stem plot)

```
x = 0:0.1:4;
y = sin(x.^2).*exp(-x);
stem(x,y); grid
```



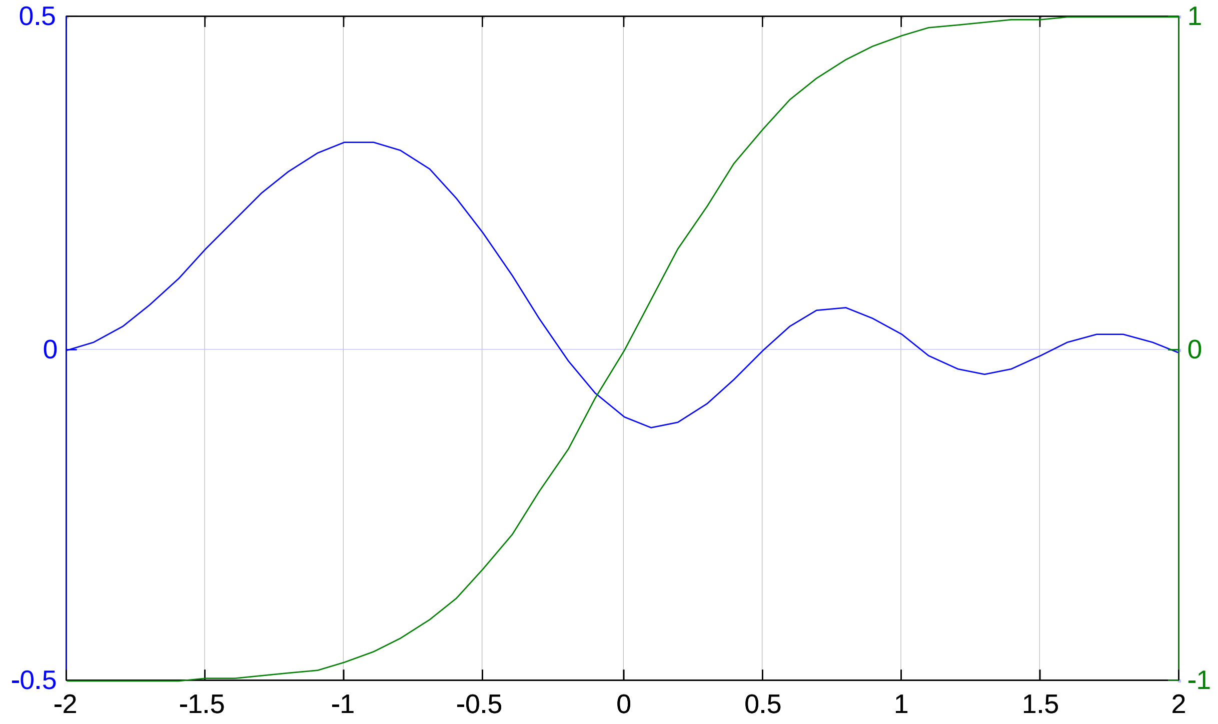
Sample 2D Plots (Histogram)

```
x=randn(1,1000);
hist(x);
grid
```



Sample 2D Plots (plotyy)

```
x=-2:0.1:2;  y1=erf(x);
y2=erf(1.35.*x);
plotyy(x,y,x,y2);grid
```

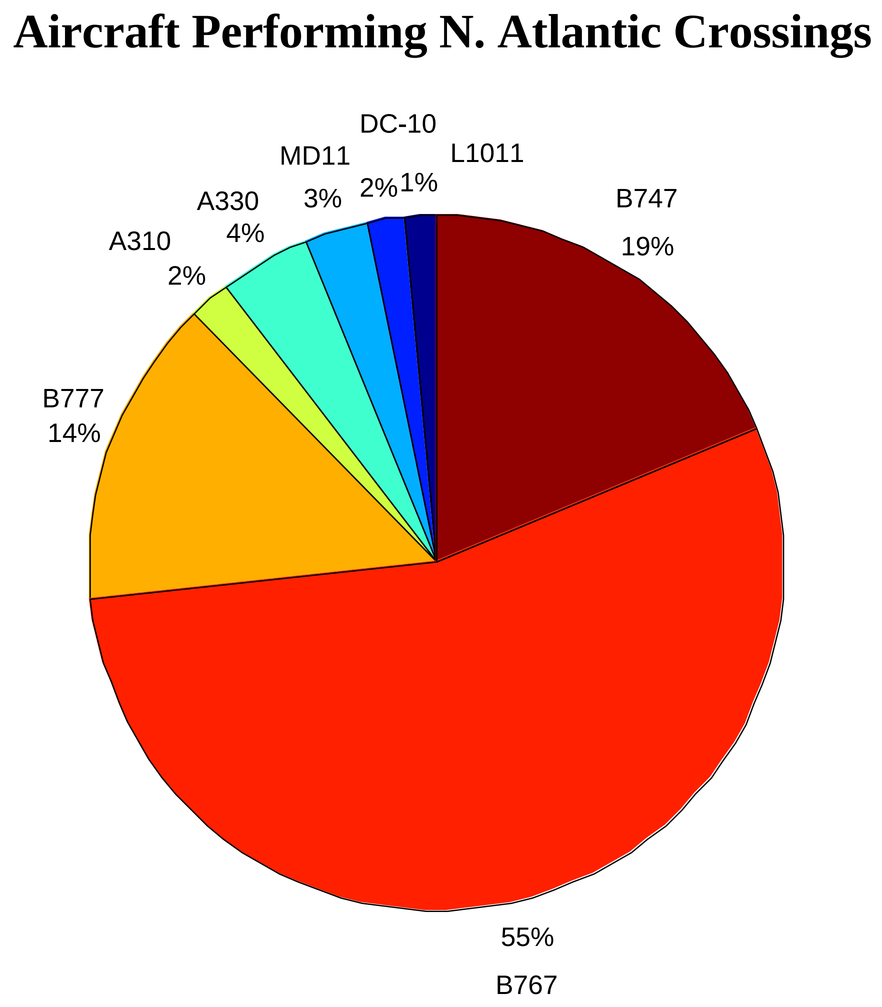


Sample 2D Plot (pie plot)

In this example we demonstrate the use of the gtext function to write a string at the location of the mouse

```
acft = char('A310','A330','MD11','DC-10', 'L1011',...
                                   'B747','B767','B777');
numbers=[12 15 24 35 16 120 456 156];
pie(numbers)
for i=1:8                          % array of strings
     gtext(acft(i,:));             % get text from char variable
end
title('Aircraft Performing N. Atlantic Crossings')
```

Resulting Pie Plot



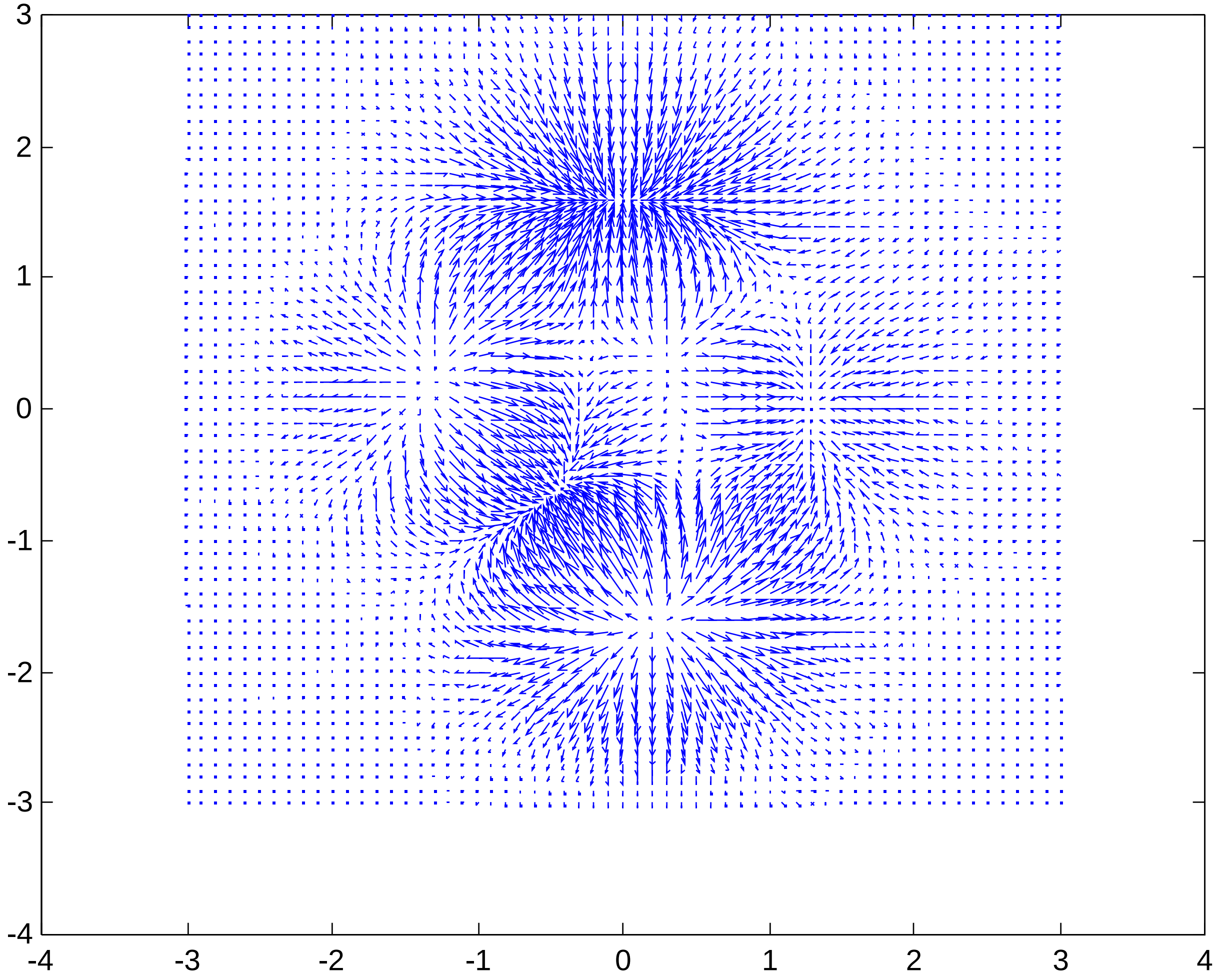
Quiver Plot

The quiver plot is good to represent vector fields.
In the example below a quiver plot shows the gradient of a function called ‘peaks’

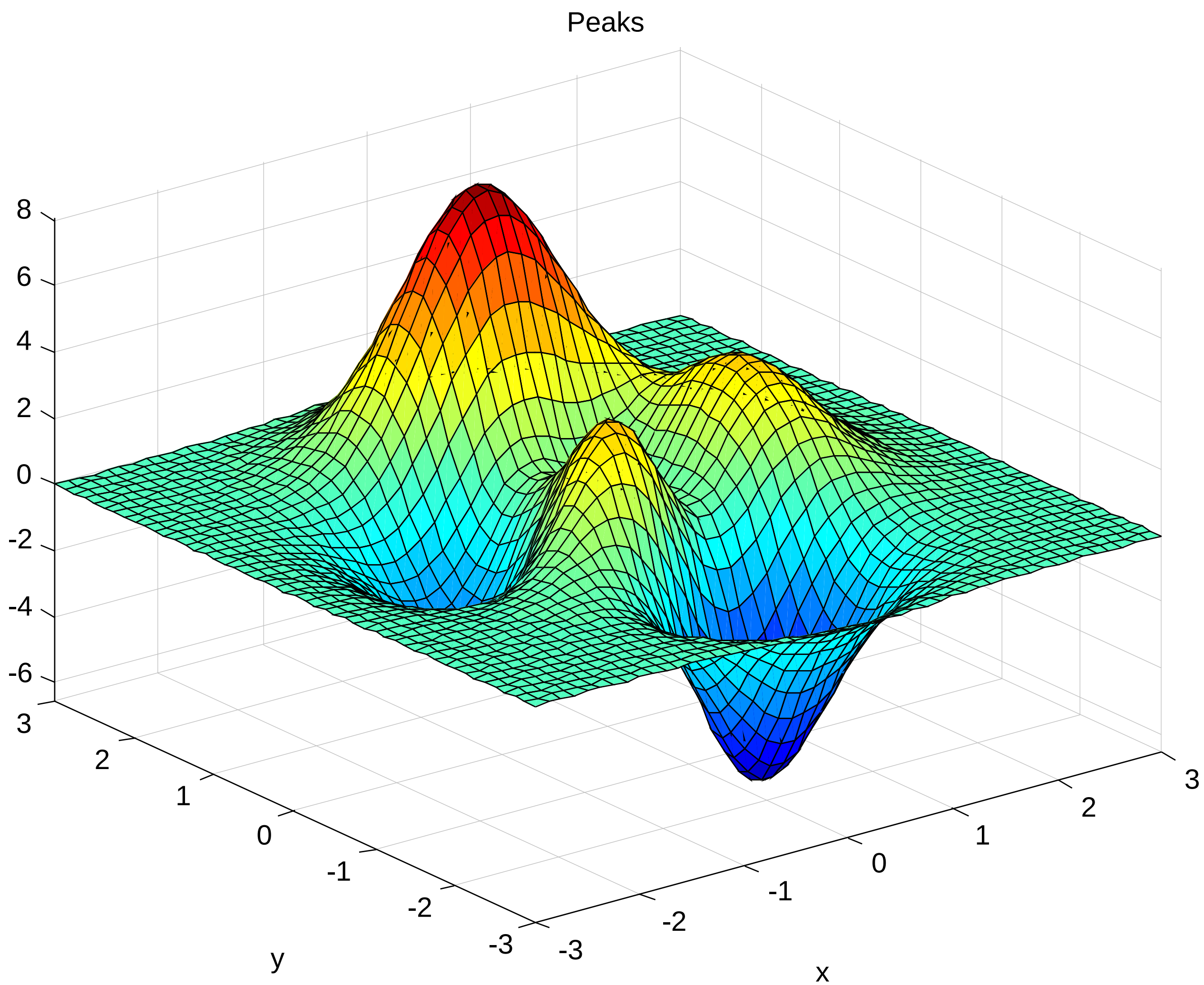
```
t=-3:.1:3;
[x,y]=meshgrid(t,t);
z= 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
  - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
  - 1/3*exp(-(x+1).^2 - y.^2);
[dx,dy]=gradient(z,.2,.2);
quiver(x,y,dx,dy,3)
```

Note: ‘peaks’ is a built-in MATLAB function

Quiver Plot of ‘Peaks’ Function



3D Representation of the Peaks Function

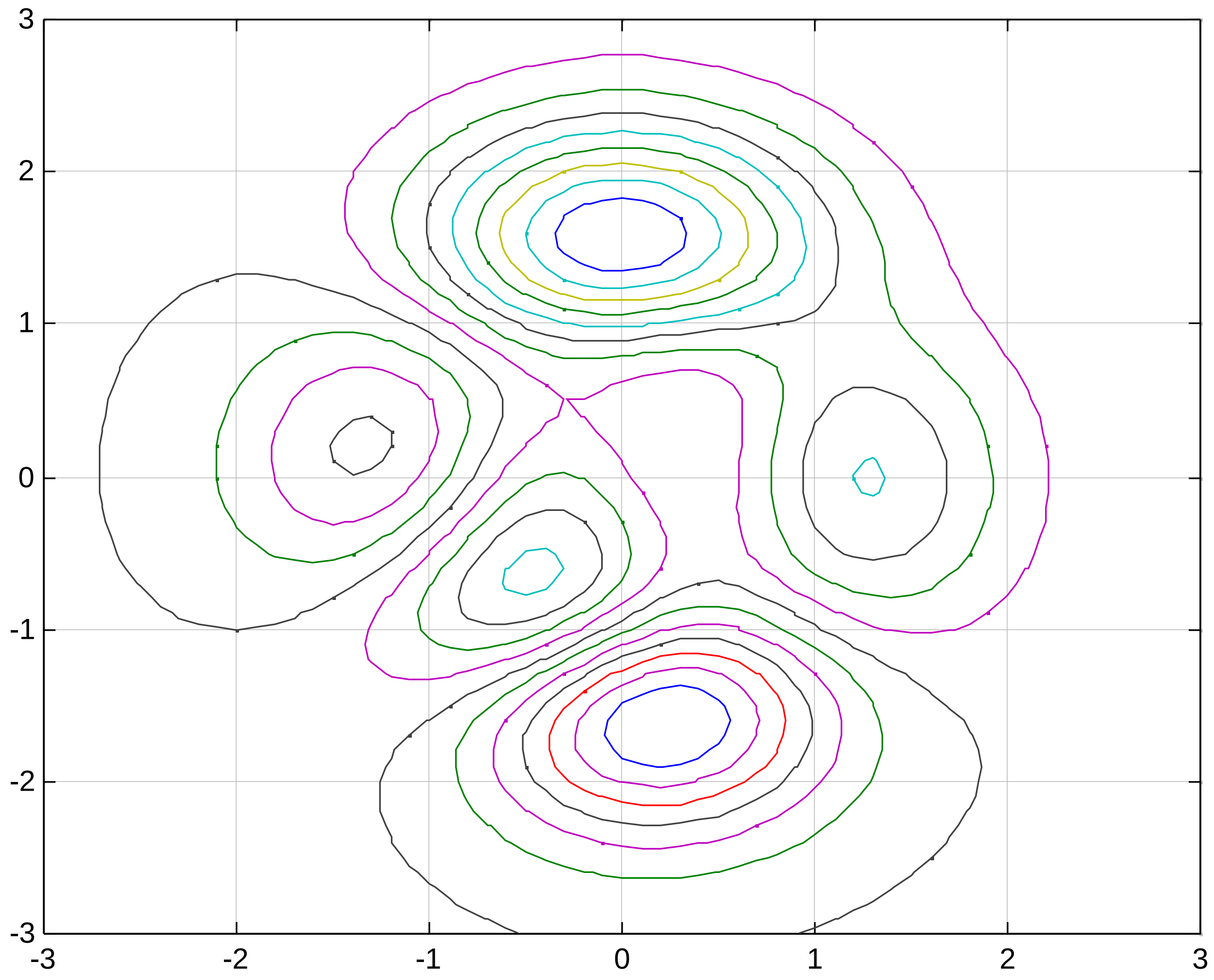


2D Plots (Contour Plot)

The following script generates a contour plot of the peaks function

```
t=-3:.1:3;
[x,y]=meshgrid(t,t);
z= 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
   - 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
   - 1/3*exp(-(x+1).^2 - y.^2);
colormap(lines)
contour(x,y,z,15)            % 15 contours are generated
grid
```

Resulting Contour Plot of the ‘Peaks’ Function

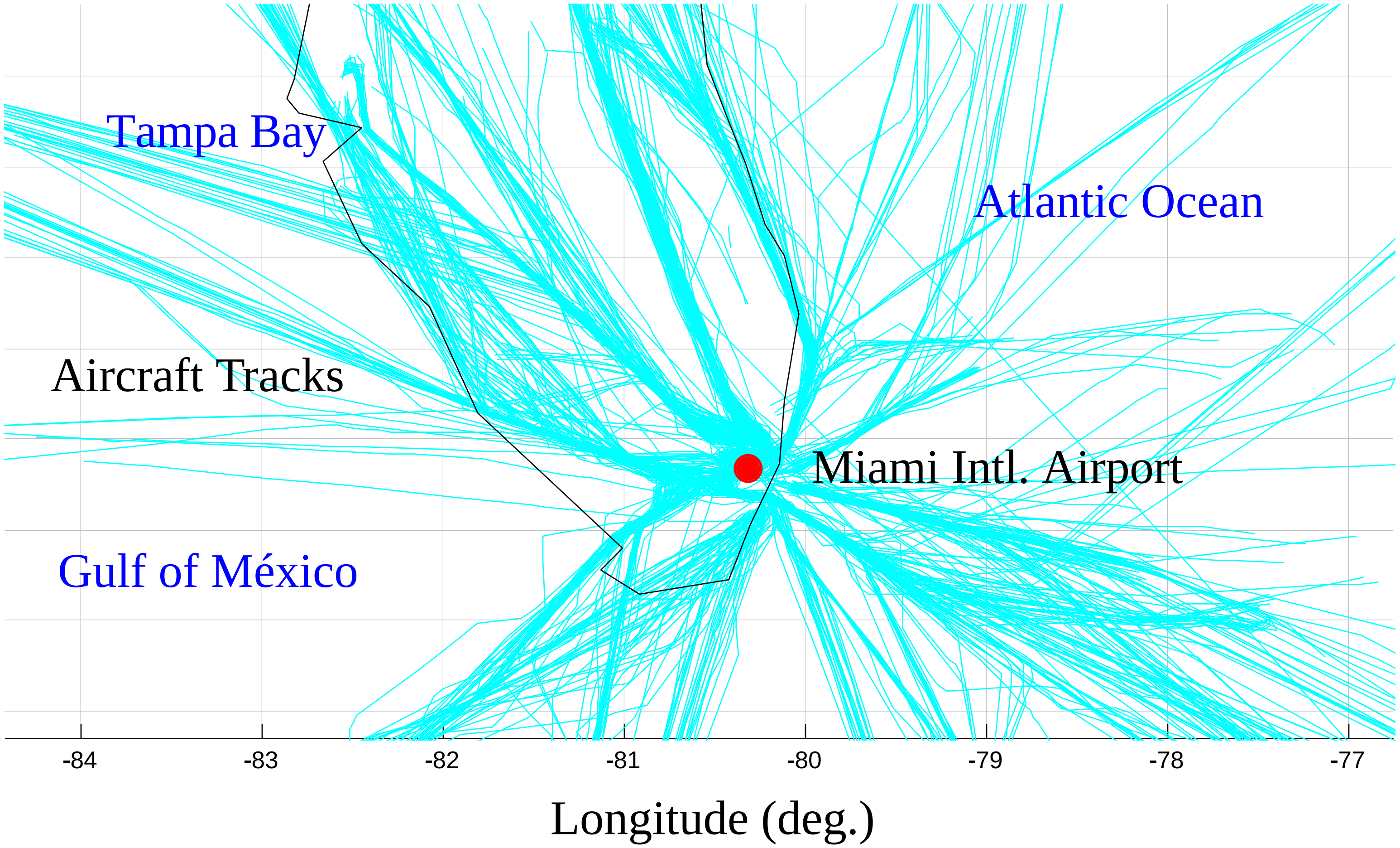


Sample 2D Plots (comet plot)

- Useful to animate a trajectory
- Try the following script

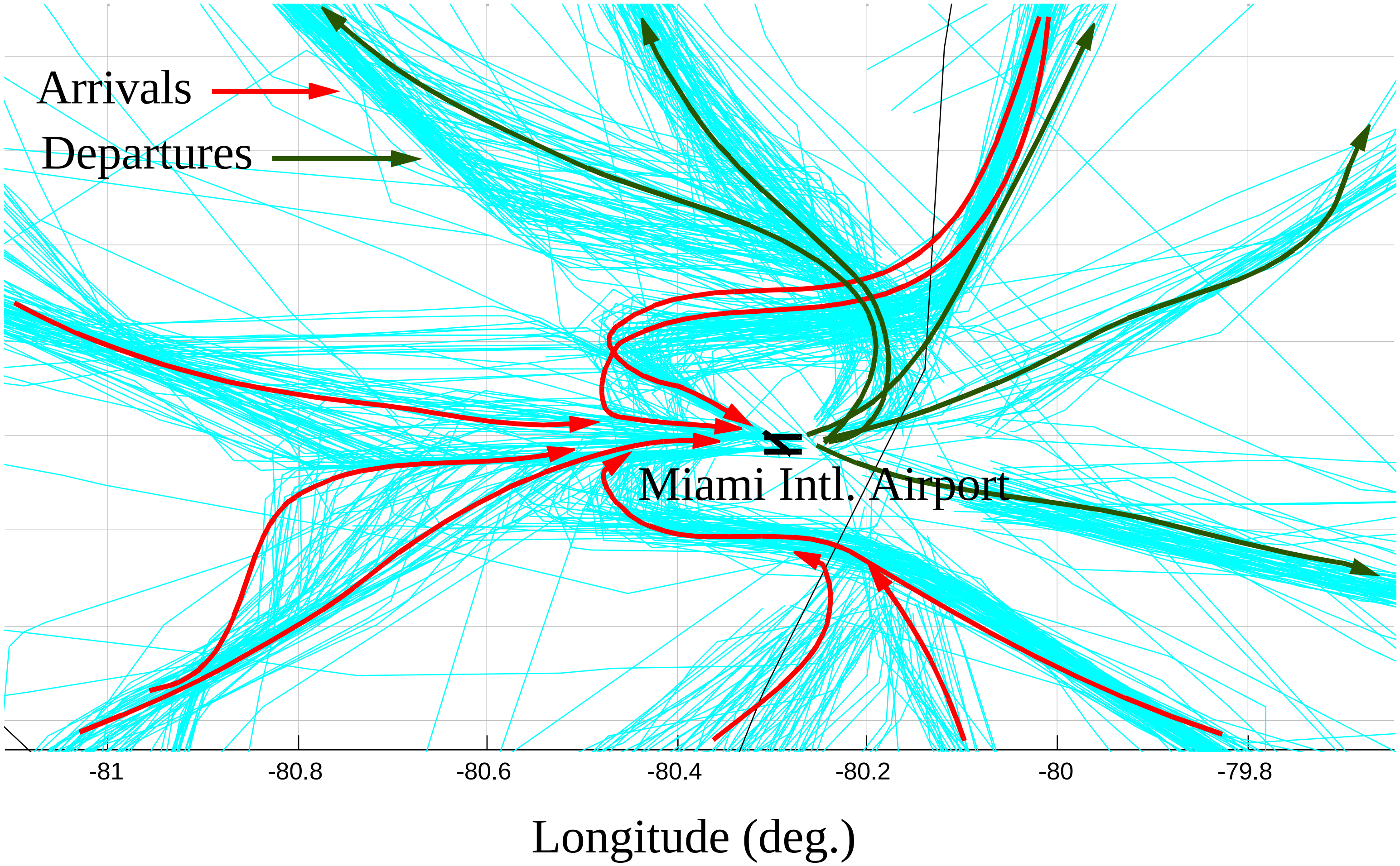
```
% illustration of comet plot

x = 0:0.05:8;
y = sin(x.^2).*exp(-x);
comet(x,y)
```

Handling Complex 2-D Plots



Plot containing 830 flight tracks arriving or departing Miami Airport
Each track has 325 data points
Source: FAA, plot by A. Trani (Air Transportation Systems Lab)

Zooming Into Previous Plot



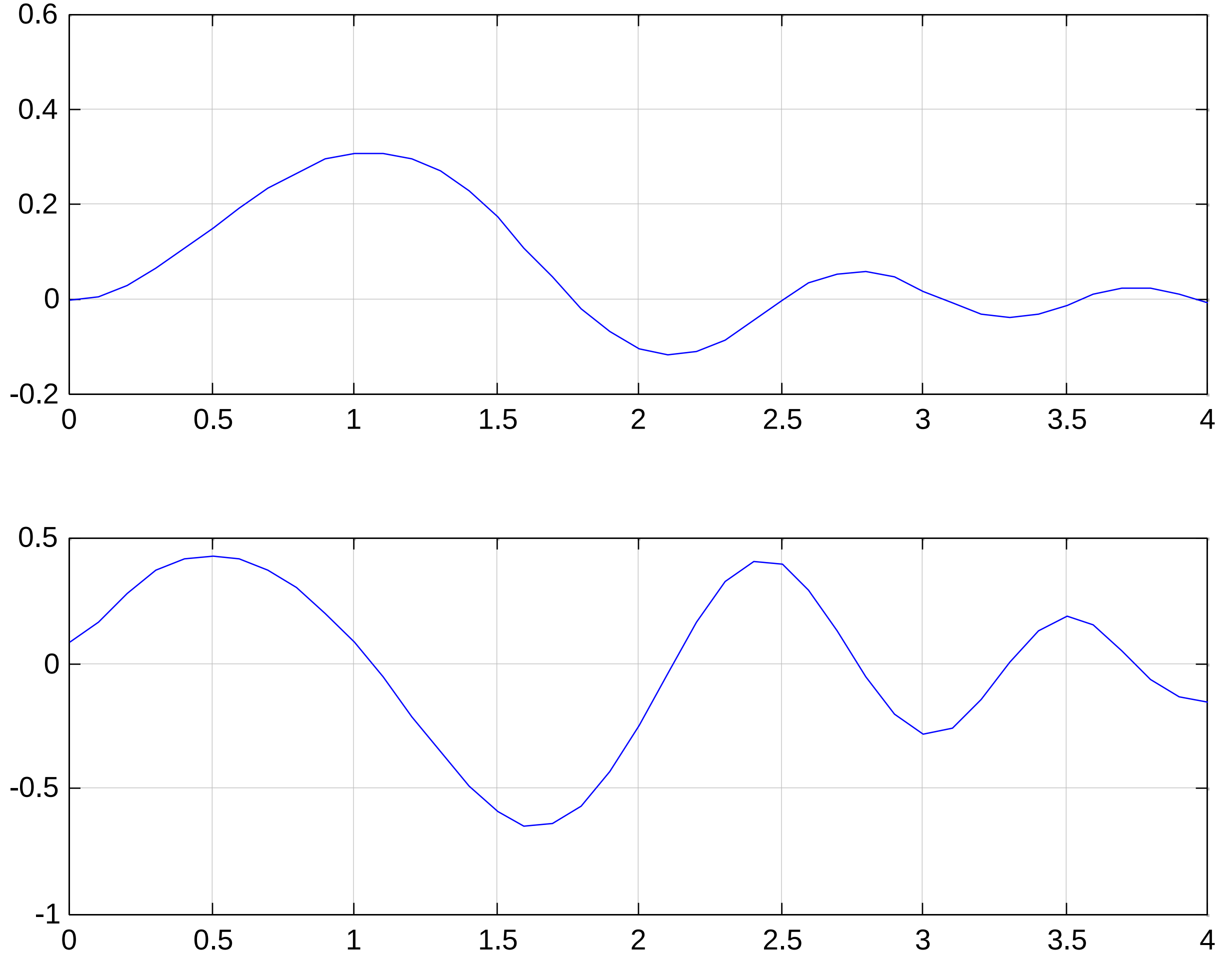
Plot containing 830 flight tracks arriving or departing Miami Airport
Each track has 325 data points
Source: FAA, plot by A. Trani (Air Transportation Systems Lab)

Sample Use of Subplot Function

Used the subplot function to display various graphs on the same screen

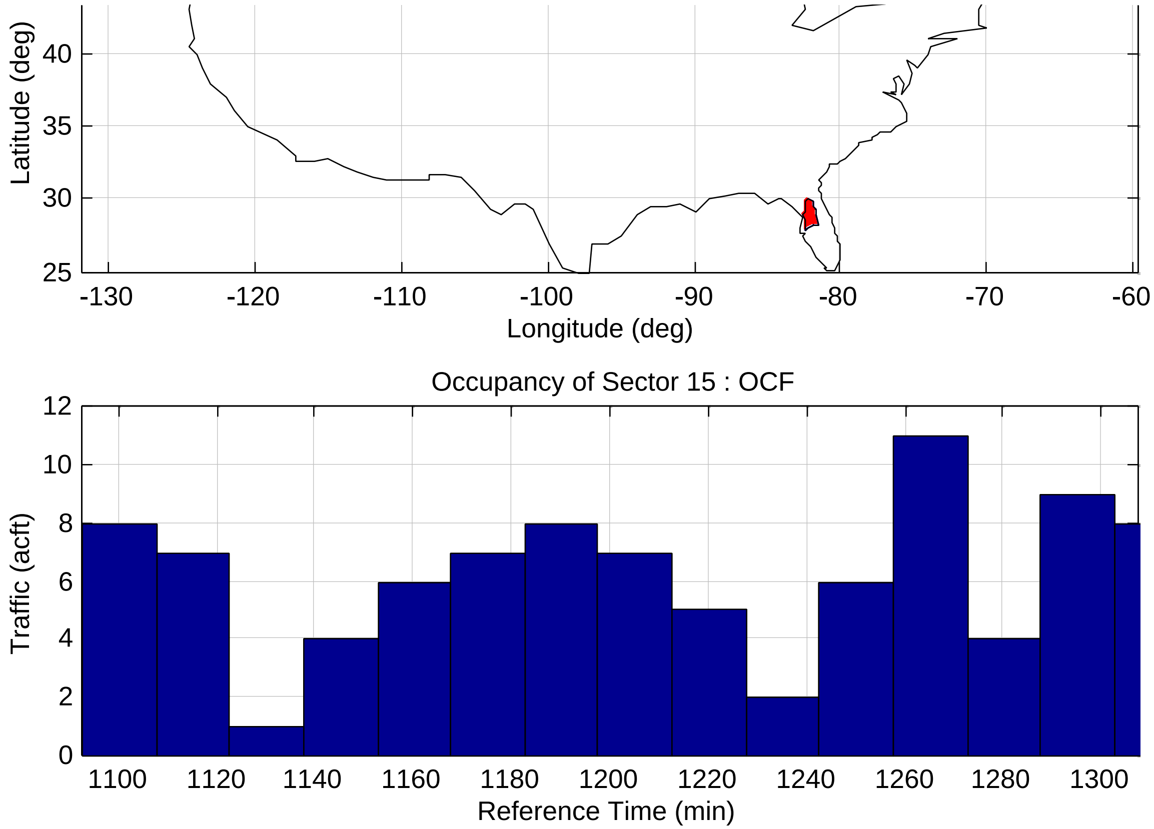
```
% Demo of subplot function
x = 0:0.1:4;
y = sin(x.^2).*exp(-x);
z=gradient(y,.1)          % takes the gradient of y every
                          % 0.1 units

subplot(2,1,1)            % generates the top plot
plot(x,y); grid

subplot(2,1,2)            % generates the lower plot
plot(x,z); grid
```

Resulting Subplot



Sample Plot Commands

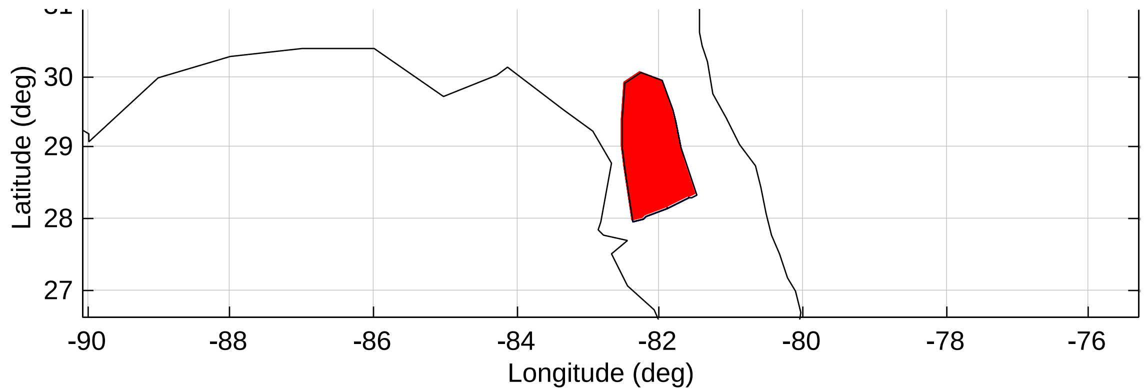
Standard 2D plot using the 'subplot' function

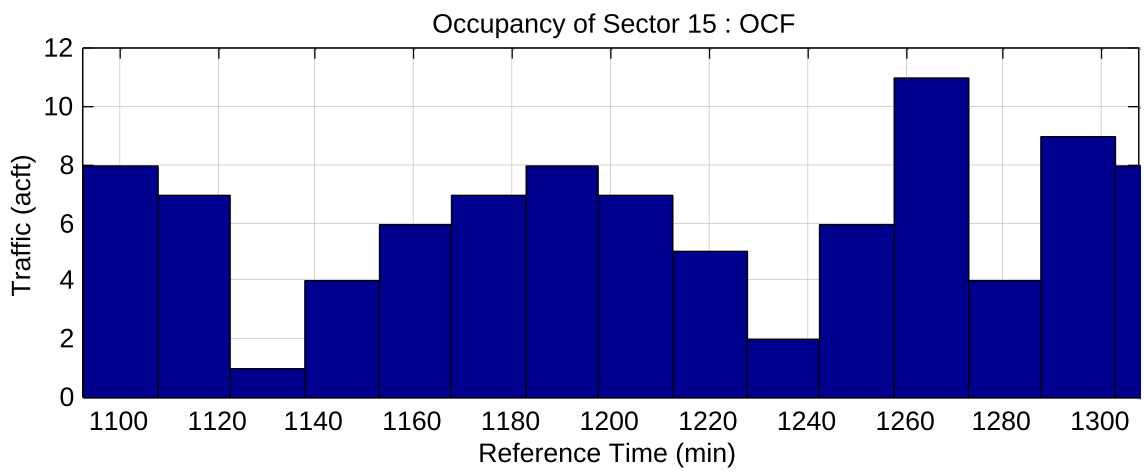


Plot containing 104 flight tracks crossing an airspace sector in Florida
Source: FAA, plot by A. Trani (Air Transportation Systems Lab)

Zoom Command

The ‘zoom’ command is used to examine a smaller area





Plot containing 104 flight tracks crossing an airspace sector in Florida
Source: FAA, plot by A. Trani (Air Transportation Systems Lab)

3-D Graphing in MATLAB

- A 3-D plot could help you visualize complex information
- 3D animations can be generated from static 3D plots
- 3D controls fall into the following categories:
  - viewing control (azimuth and elevation)
  - color control (color maps)
  - lighting control (specular, diffuse, material, etc.)
  - axis control
  - camera control
  - graph annotation control
  - printing control

Viewing Control

- 3D plots have two viewing angles than can be controlled with the command view
  - azimuth
  - elevation

  Example use: view(azimuth, elevation)

- Default viewing controls are: -37.5 degrees in azimuth and 30 degrees in elevation
- Try the traffic file changing a few times the viewing angle

Rotating Interactively a 3D Plot

- Use the rotate3d command to view interactively the 3D plots (good for quantitative data analysis)
- The zoom command does not work in 3D

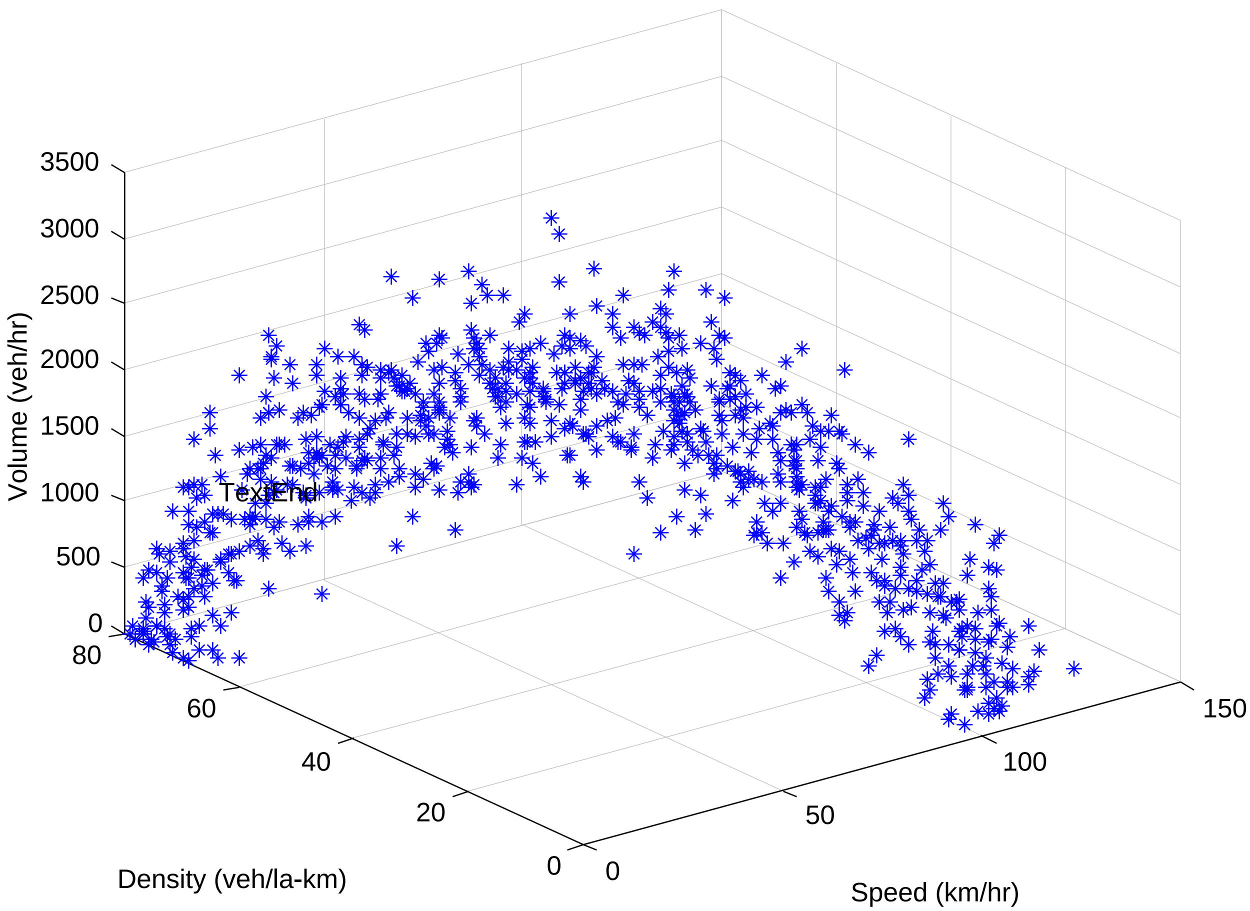
```
>> plot3d(x,y,z)
>> rotate3d
>>
```

- Try rotating the traffic characteristics file using rotate3d

Retrieve the data file called: traffic flow data from our syllabus web site

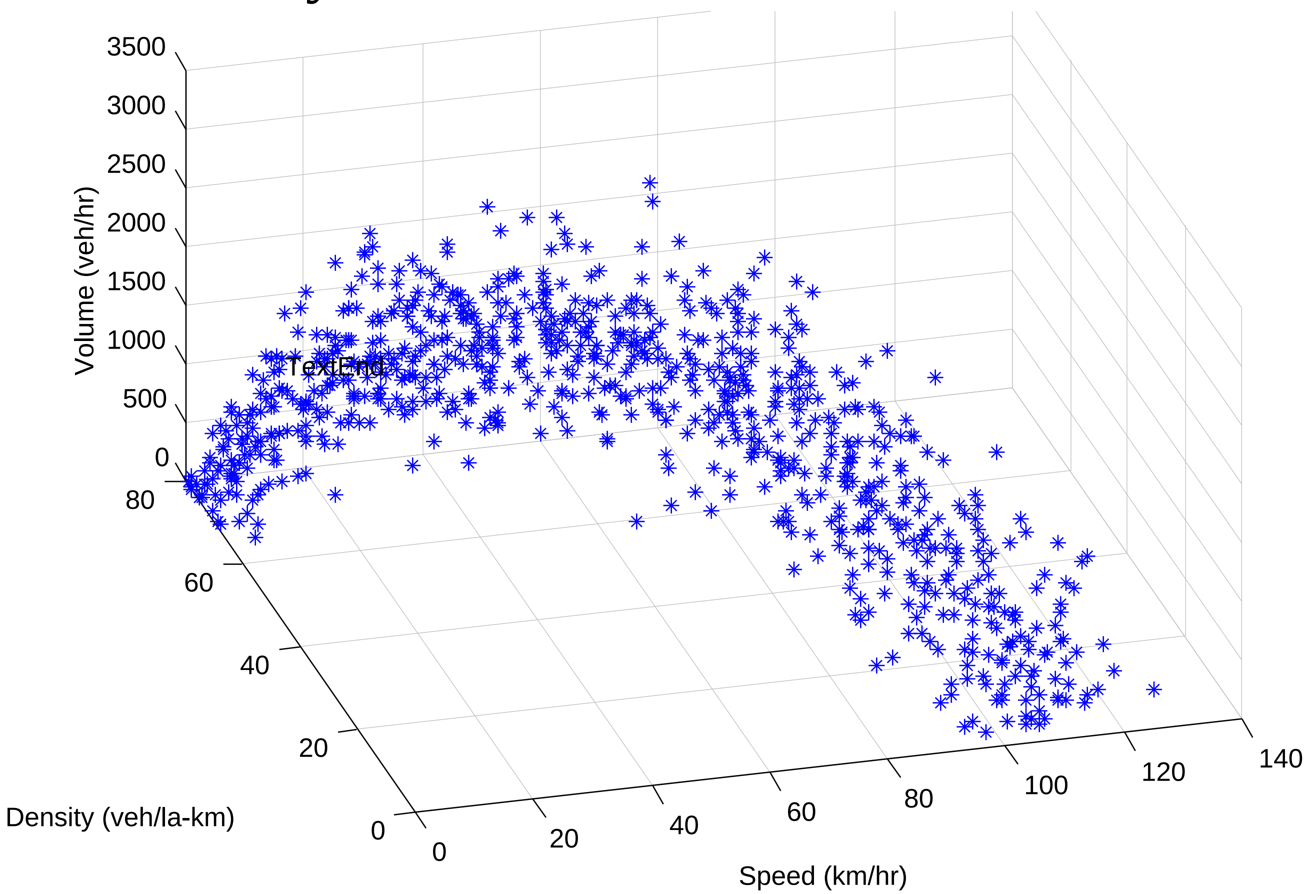
Sample 3D Plot (plot3 function)

plot3(density,speed,volume,'*')



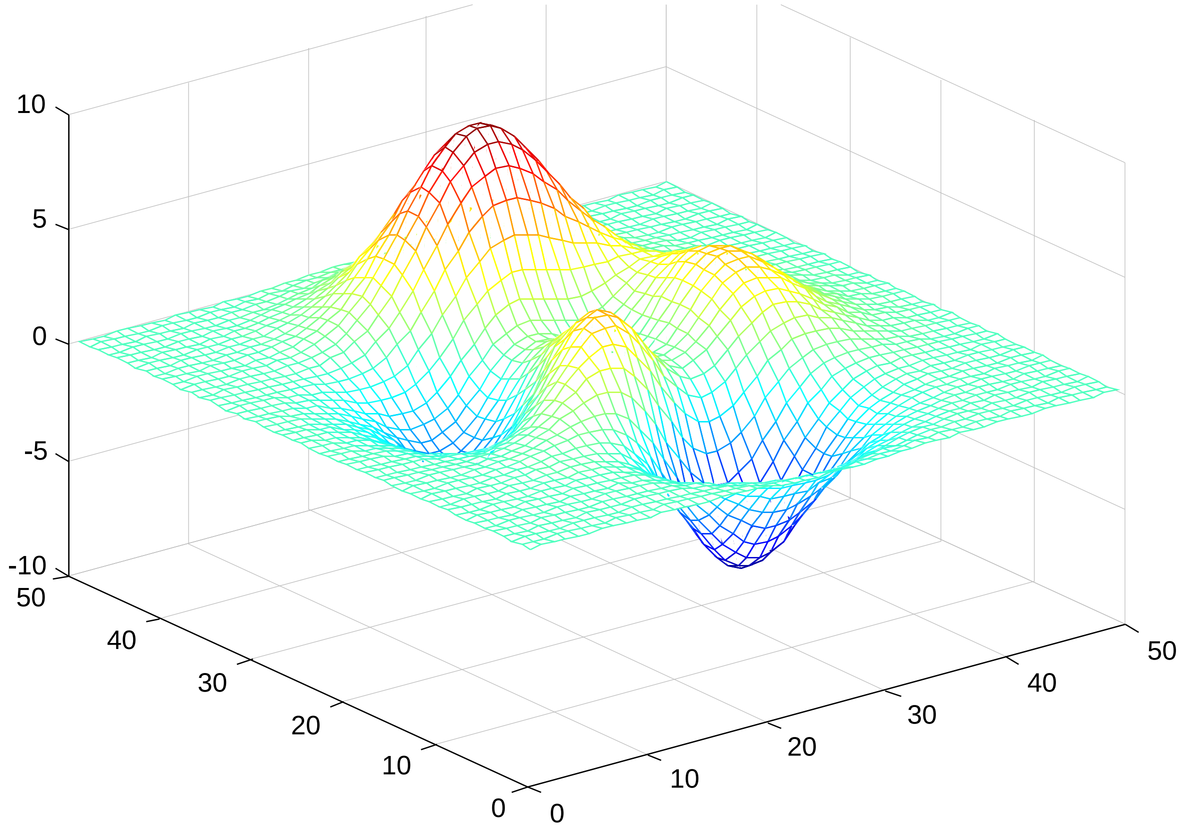
Rotate Command

The rotate3d command is useful to visualize 3D data interactively



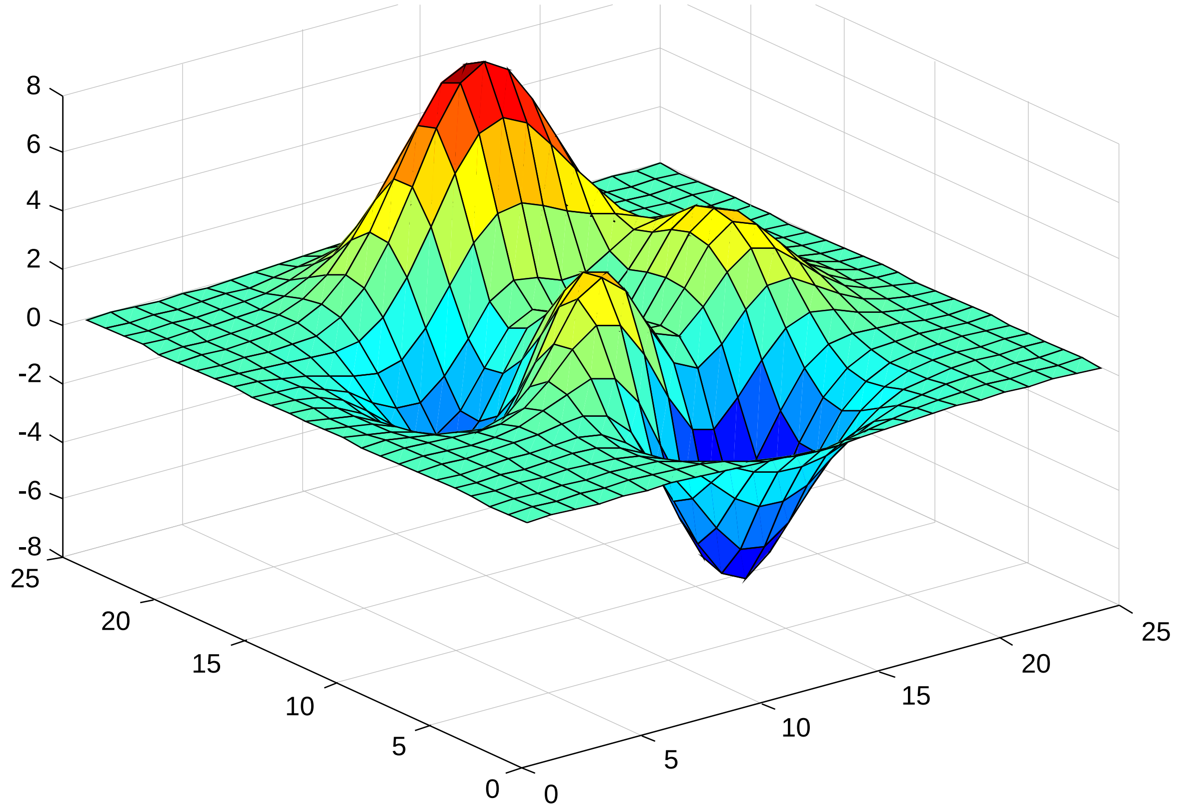
Sample 3D Graphics (mesh)

% Mesh Plot of Peaks
z=peaks(50);
mesh(z);



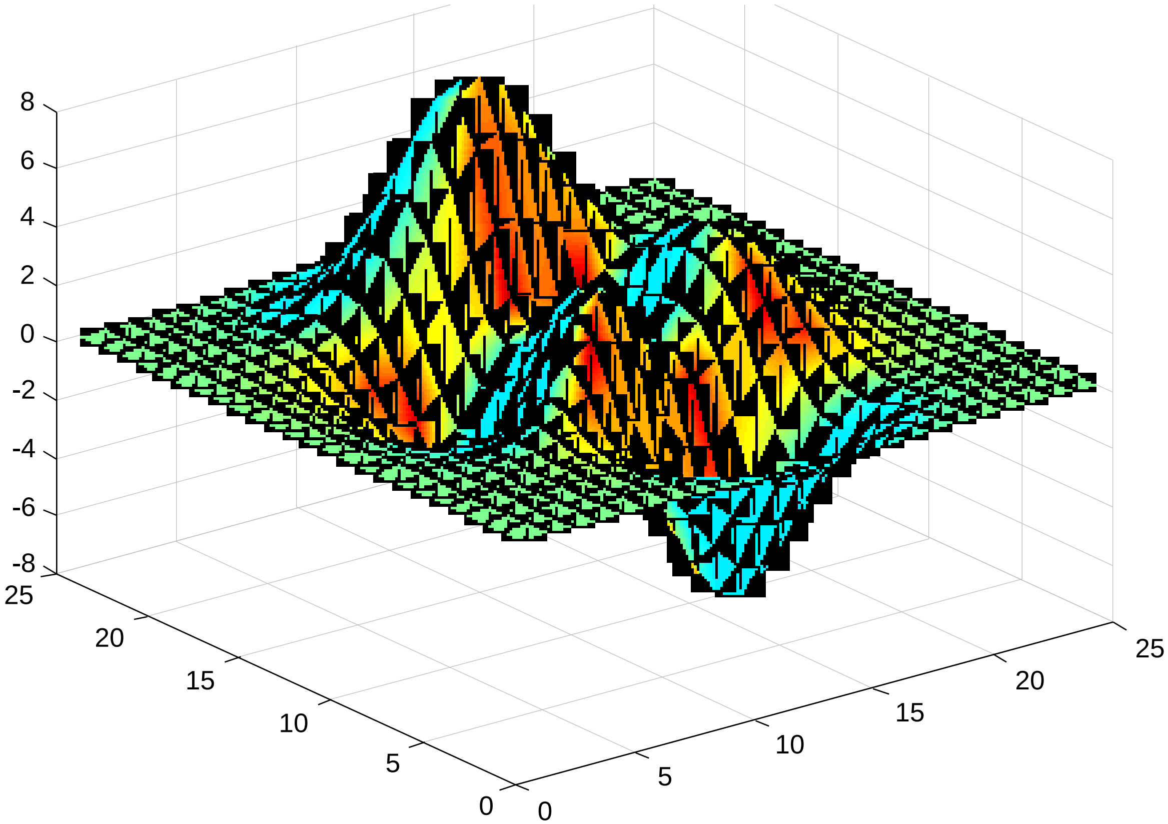
Sample 3D Graphics (surf)

z=peaks(25);
surf(z);
colormap(jet); ;



Sample 3D Graphics (surfl)

```
z=peaks(25);
surfl(z);
shading interp; colormap(jet);;
```

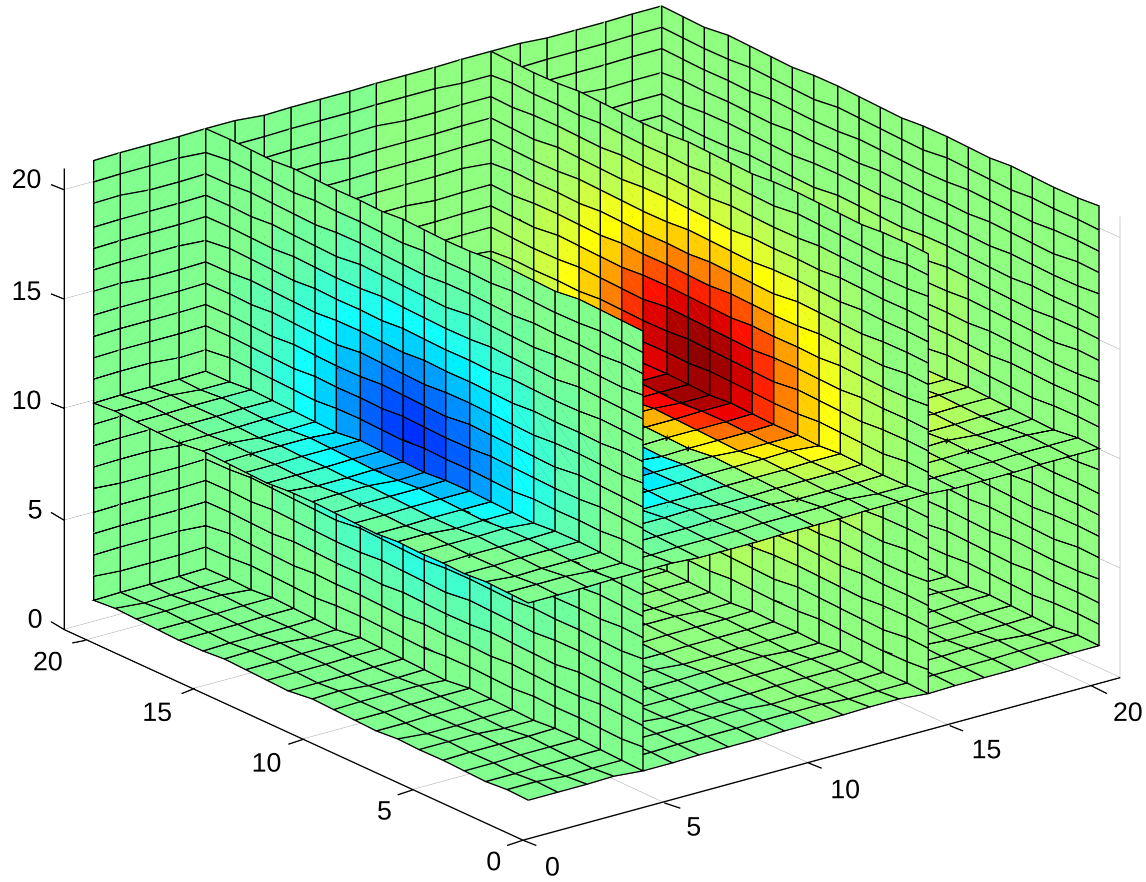


Sample 3D Graphics (slice)

Slice 3D plots visualize the internal structure of set of numbers as gradients

```
[x,y,z] = meshgrid(-2:.2:2,-2:.2:2,-2:.2:2);
 v = x .* exp(-x.^2 - y.^2 - z.^2);
 slice(v,[5 15 21],21,[1 10])
 axis([0 21 0 21 0 21]);
 colormap(jet)
```

Slice Plot of Pseudo-Gaussian Function



Sample 3D Graphics (stem3)

stem3 - plots stems in three dimensions as shown below