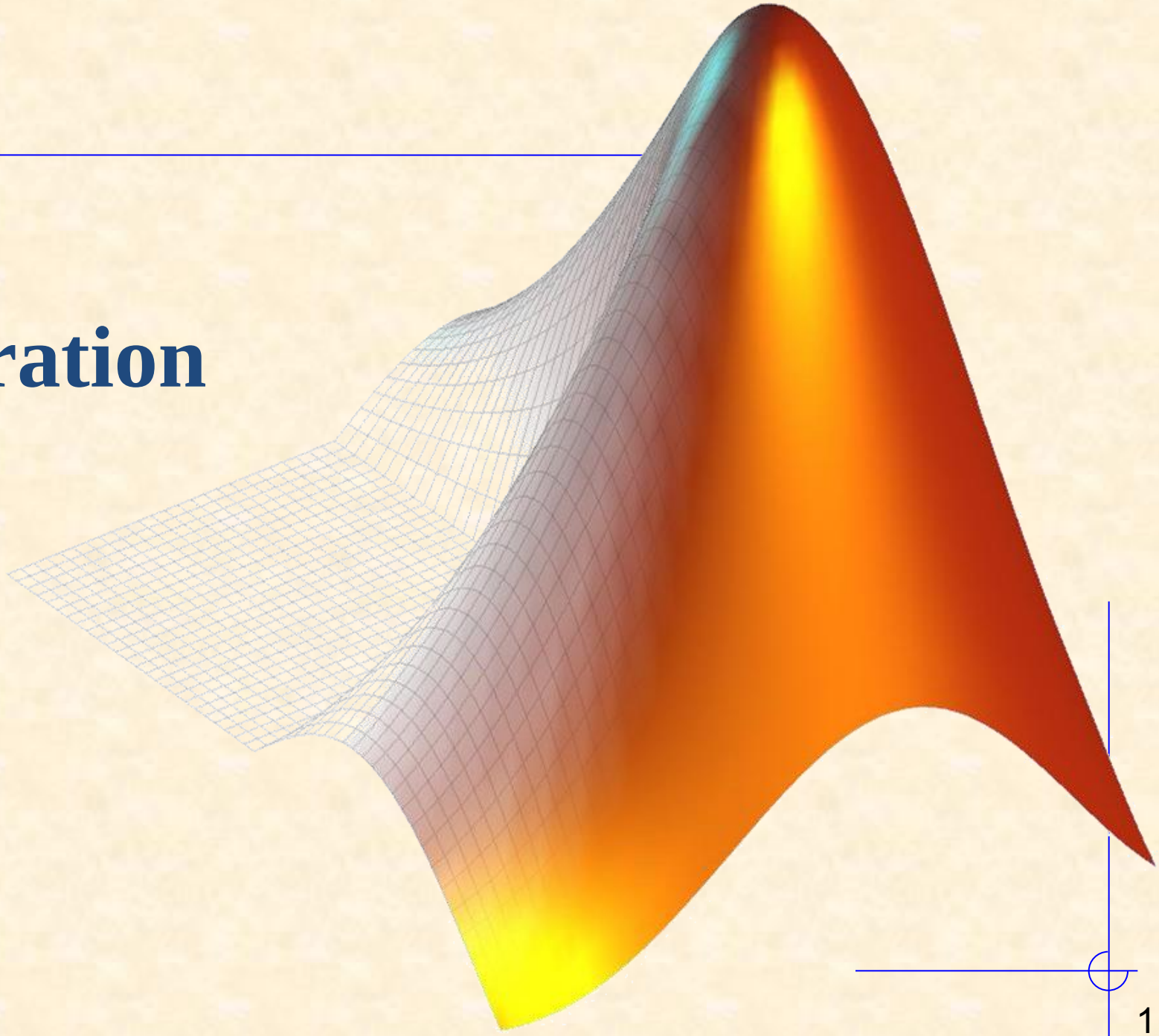
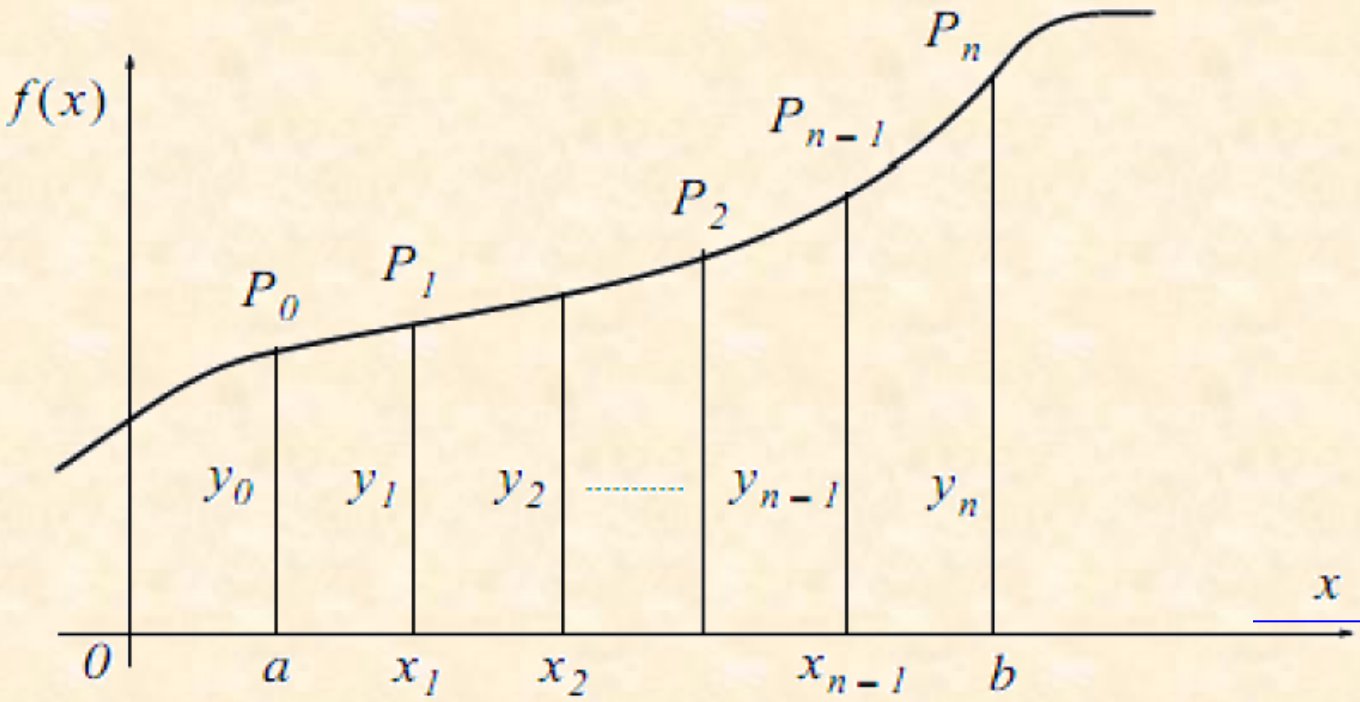
Integration



שיטת טרפז

תהיה הפונקציה y = f(x) בעלת ערכים ממשיים שתלויה במשתנה ממשי באינטרוול $a \le x \le b$
אנו נרצה לחשב את האינטגרל:

$$\int_a^b f(x)\,dx$$



שיטת טרפז

ע"י חילוק האינטרוול $a \le x \le b$ ל n תתי-אינטרוולים, כל אחד באורך $\Delta x = \frac{b-a}{n}$.

אז מספר הנקודות בין $x_0 = a$ לבין $x_n = b$ יהיו:

$$x_1 = a + \Delta x,\ x_2 = a + 2\Delta x,\ \ldots,\ x_{n-1} = a + (n-1)\Delta x$$

לכן, האינטגרל בין a ל b הוא הסכום של האינטגרלים בין a ל x_1, מ x_1 ל x_2 וכך הלאה. השטח הכללי יהיה:

$$\int_a^b f(x)dx = \int_a^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \ldots + \int_{x_{n-1}}^{b} f(x)dx = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x)dx$$

שיטת טרפז

השטח של התת-אינטרוול הראשון ניתן לחישוב ע"י שטח הטרפז $aP_0P_1x_1$ ששווה ל $0.5\cdot(y_0+y_1)\cdot\Delta x$ ועוד השטח של הטרפז $x_1P_1P_2x_2$ ששוה ל $0.5\cdot(y_1+y_2)\cdot\Delta x$, וכך הלאה.

ואז הקירוב ה trapezoidal נהיה :

$$T = \frac{1}{2}(y_0+y_1)\Delta x + \frac{1}{2}(y_1+y_2)\Delta x + \ldots + \frac{1}{2}(y_{n-1}+y_n)\Delta x$$

או:

$$T = \left(\frac{1}{2}y_0 + y_1 + y_2 + \ldots + y_{n-1} + \frac{1}{2}y_n\right)\Delta x$$

Trapezoidal Rule

בעצם מקבלים את הסכימה של שטחי הטרפזים הנוצרים מחיבור ליניארי של נקודות דגימה.

שיטת טרפז

הפונקציה המבצעת אינטגרציה לפי שיטת ה trapezoidal על פני y עם התייחסות ל x, היא:

trapz(x,y)

<u>לדוגמא</u>:
חישוב האינטגרל $\int_0^{2\pi} \cos(x)\,dx$:

```
x = 0:pi/100:pi;
y = cos(x);
z = trapz(x,y)

z =
  1.5301e-016
```

אינטגרציה לפי שיטת Quadrature

ב Matlab ישנן 2 פונקציות המחשבות אינטגרלים לפי Quadrature integration, הפונקציה quad ו quad8.
שתי פונקציות אלו משתמשות בשיטת האינטגרציה adaptive quadrature, מה שאומר שהן יכולות להתמודד עם אי-סדירויות כגון סינגולאריות. כאשר אי-סדירויות כאלו קורות, Matlab יציג הודעת אזהרה אך עדיין יבצע את החישוב עצמו.

פונקצית quad משתמשת בשיטה אדפטיבית של שיטת Simpsons בעוד שפונקצית quad8 משתמשת בשיטת Newton-Cotes 8-panel.
שיטות אלו הן שיטות יותר מדויקת משיטת ה trapeziodal.
כאשר משתמשים בפונקציות אלו אין צורך לקבוע מרווחי דגימה ואת הפונקציה (y=f(x)) עצמה צריך לשמור כ- M-file function.

אינטגרציה לפי שיטת Quadrature

התחביר של הפונקציות quad ושל quad8 הוא:

```
q = quad(fun,a,b,tol)
```

כאשר

fun – הפונקציה שעליה מבוצעת האינטגרציה.
fun היא מצביע לפונקצית m או לפונקציה אנונימית.
a – הגבול התחתון של האינטרוול.
b – הגבול העליון של האינטרוול.
tol – שגיאת האינטגרל (ברירת המחדל היא 1e-6).

אינטגרציה לפי שיטת Quadrature

חישוב האינטגרל:

$$\int_0^{\frac{\pi}{2}} \left(\cos(x)+\sin(2x)\right)dx$$

ע"י הגדרת פונקציה רגילה:

```
function y = f(x)
y = cos(x)+sin(2*x);
end
```

```
Q = quad(@f,0,pi/2)

Q =
   2.0000
```

אינטגרל כפול

אינטגרל כפול מוגדר באופן הבא:

$$\int_{y\min}^{y\max}\int_{x\min}^{x\max} f(x,y)dxdy$$

התחביר של הפונקציה ב Matlab הוא:

```
q = dblquad(fun,xmin,xmax,ymin,ymax,tol)
```

כאשר

fun – הפונקציה שעליה מבוצעת האינטגרציה.

xmin ו xmax – הגבולות של האינטרוול של x.

ymin ו ymax – הגבולות של האינטרוול של y.

tol – שגיאת האינטגרל (ברירת המחדל היא 1e-6).

אינטגרל כפול

חישוב האינטגרל:

$$\int_{\pi}^{2\pi}\int_{0}^{\pi}\left(y\cdot\sin\left(x\right)+x\cdot\sin\left(y\right)\right)dxdy$$

ע"י פונקציה אנונימית:

```
F = @(x,y)y*sin(x)+x*cos(y);
Q = dblquad(F,pi,2*pi,0,pi);

Q =

   -9.8696
```

אינטגרל משולש

אינטגרל משולש מוגדר באופן הבא:

$$\int_{x\min}^{x\max}\int_{y\min}^{y\max}\int_{z\min}^{z\max} f(x,y,z)dxdydz$$

התחביר של הפונקציה ב Matlab הוא:

triplequad(fun,xmin,xmax,ymin,ymax,zmin,zmax,tol)

כאשר

fun – הפונקציה שעליה מבוצעת האינטגרציה.

xmin ו xmax – הגבולות של האינטרוול של x.

ymin ו ymax – הגבולות של האינטרוול של y.

zmin ו zmax – הגבולות של האינטרוול של z.

tol – שגיאת האינטגרל (ברירת המחדל היא 1e-6).

אינטגרל משולש

חישוב האינטגרל:

$$\int_0^{\pi}\int_0^{1}\int_{-1}^{1}\left(y\cdot\sin\left(x\right)+z\cdot\cos\left(x\right)\right)dxdydz$$

ע"י פונקציה אנונימית:

```
F = @(x,y,z)y*sin(x)+z*cos(x);
Q = triplequad(F,0,pi,0,1,-1,1);

Q =

    2.0000
```

תרגילים

1. פונקצית השגיאה (error function) היא:

$$erf\left(x\right)=\frac{2}{\sqrt{\pi}}\int_{0}^{x}e^{-t^{2}}dt$$

חישוב ערך פונקצית השגיאה עבור x=2.5 עבור 10 דגימות בשיטת ה trapeziodal.

```
t = linspace(0,2.5,10);
y = exp(-t.^2);
erf = trapz(t,y)

erf =

    0.8858
```

תרגילים

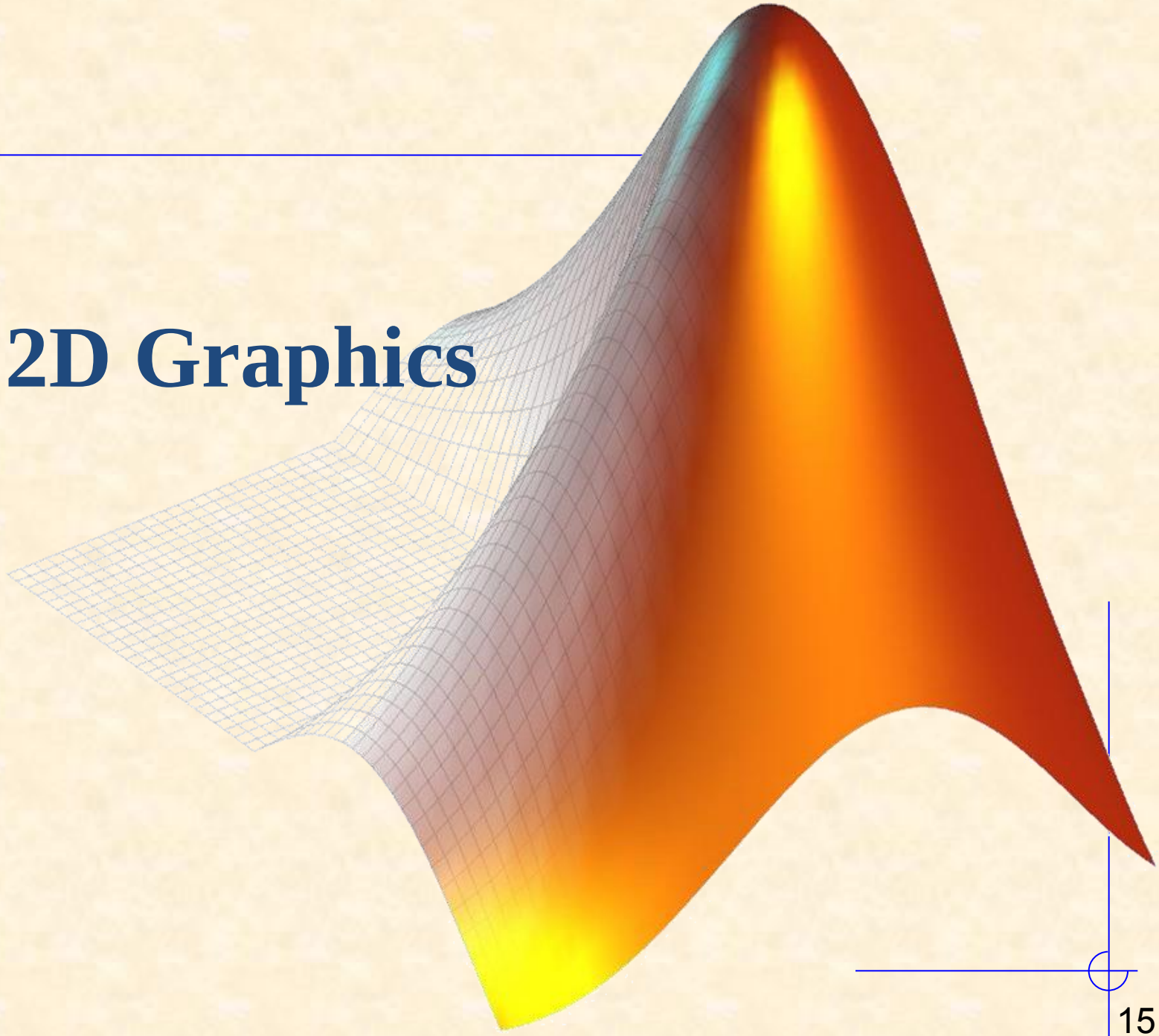
2. חישוב האינטגרל:

$$\int_0^1 \frac{x \cdot \sin(4x)}{\sqrt{1-\left(\frac{x}{2\pi}\right)^2}}\,dx$$

```
function y = f(x)
y =  x.*sin(4*x)./(sqrt(1 - (x/(2*pi)).^2));
end

Q = quad(@f,0,1)
Q =
    0.1158
```

Basic 2D Graphics

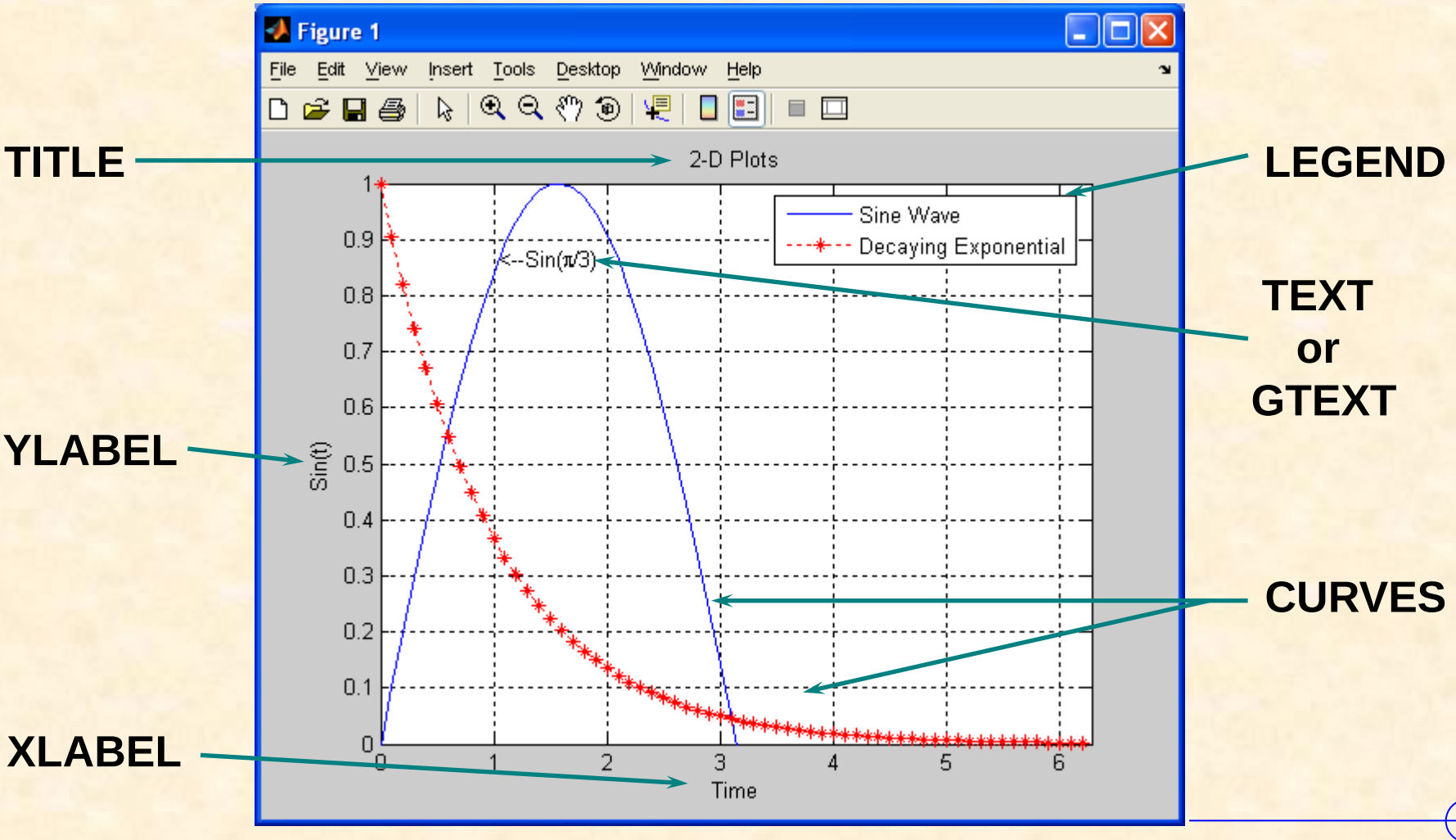


2-D Plotting

- Specify x-data and/or y-data
- Specify color, line style and marker symbol
  (Default values used if not specified)
- Syntax:
  - Plotting a single line:
  - ```
    plot(xdata, ydata, 'color_linestyle_marker')
    ```
  - Plotting multiple lines:

```
plot(x1, y1, 'clm1', x2, y2, 'clm2', ...)
```
```

Graphics Example – a final product:



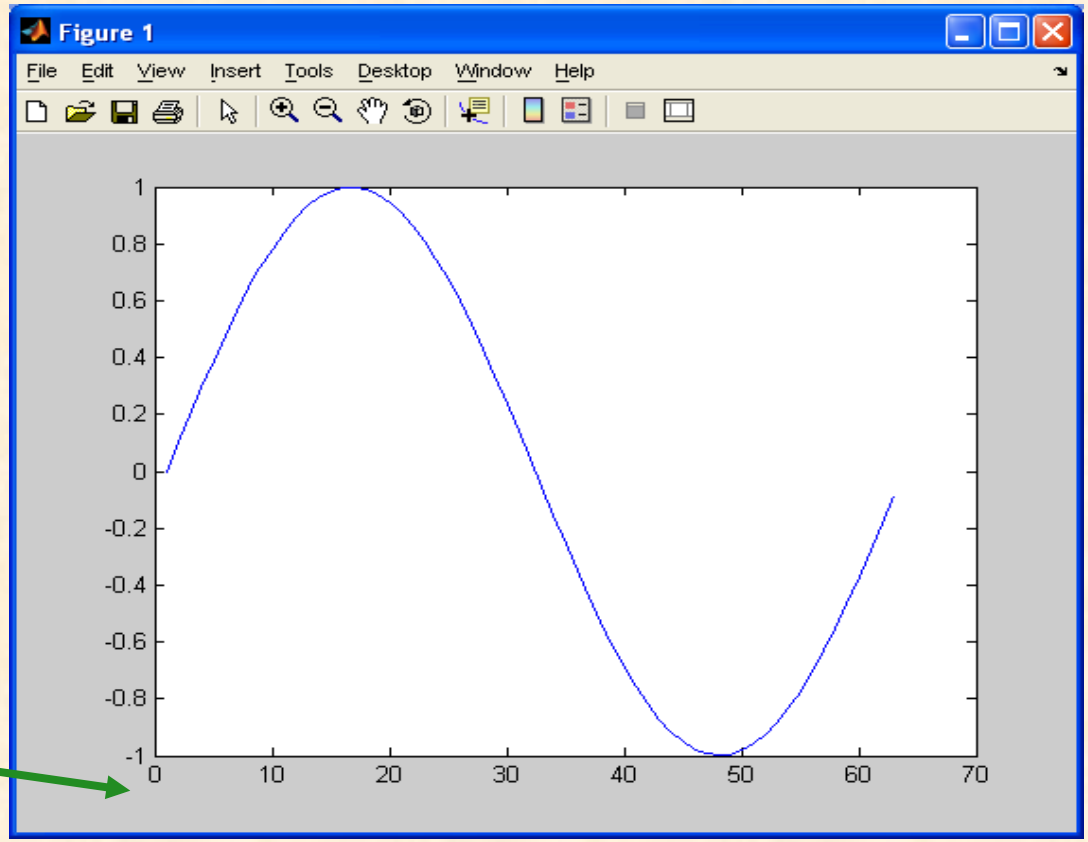
2-D Plotting – *simplest example*

Create a Blue Sine Wave

```
>> x = 0:.1:2*pi;

>> y = sin(x);

>> plot(y)
```



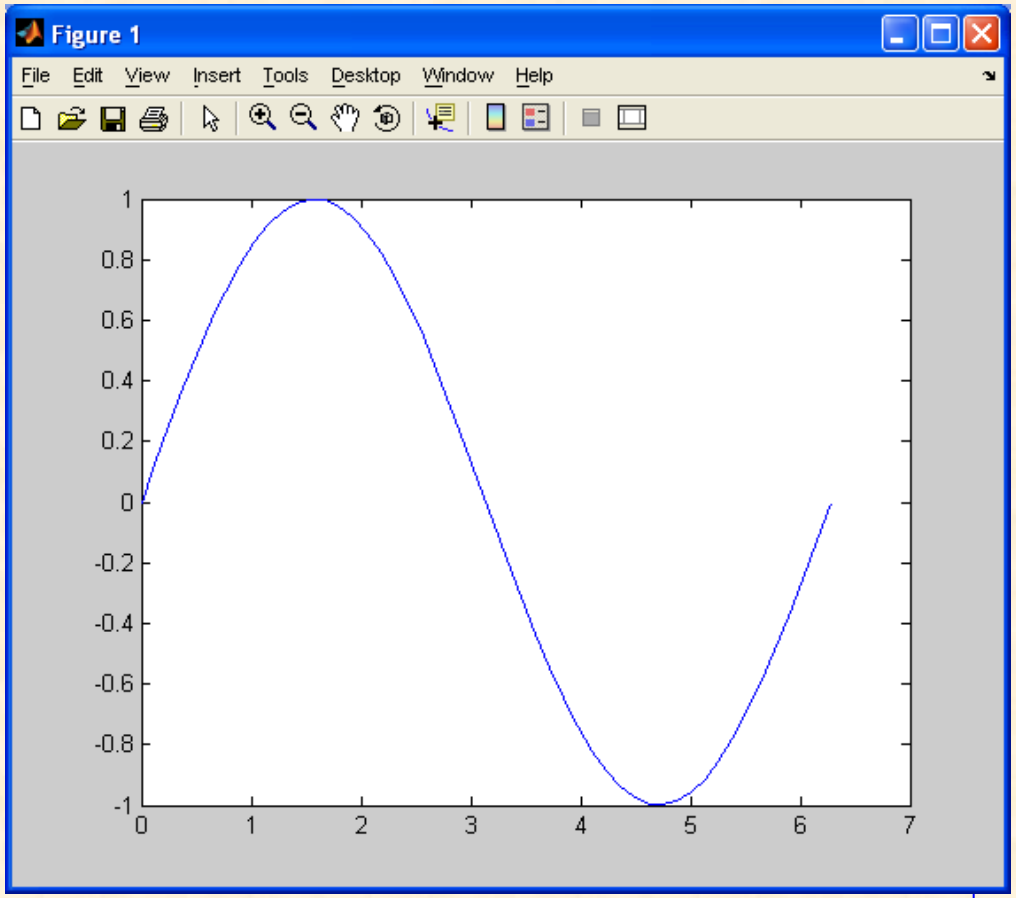
If x axis isn't specified, MATLAB uses consecutive numbers

2-D Plotting – *with x axis*:

Create a Blue Sine Wave

with appropriate x-axis

```
>> x=linspace(0,2*pi,50);
>> y = sin(x);
>> plot(x,y)
```



Note values on x axis:

2-D Plotting – *multiple lines*

Create a blue sine wave and green cosine

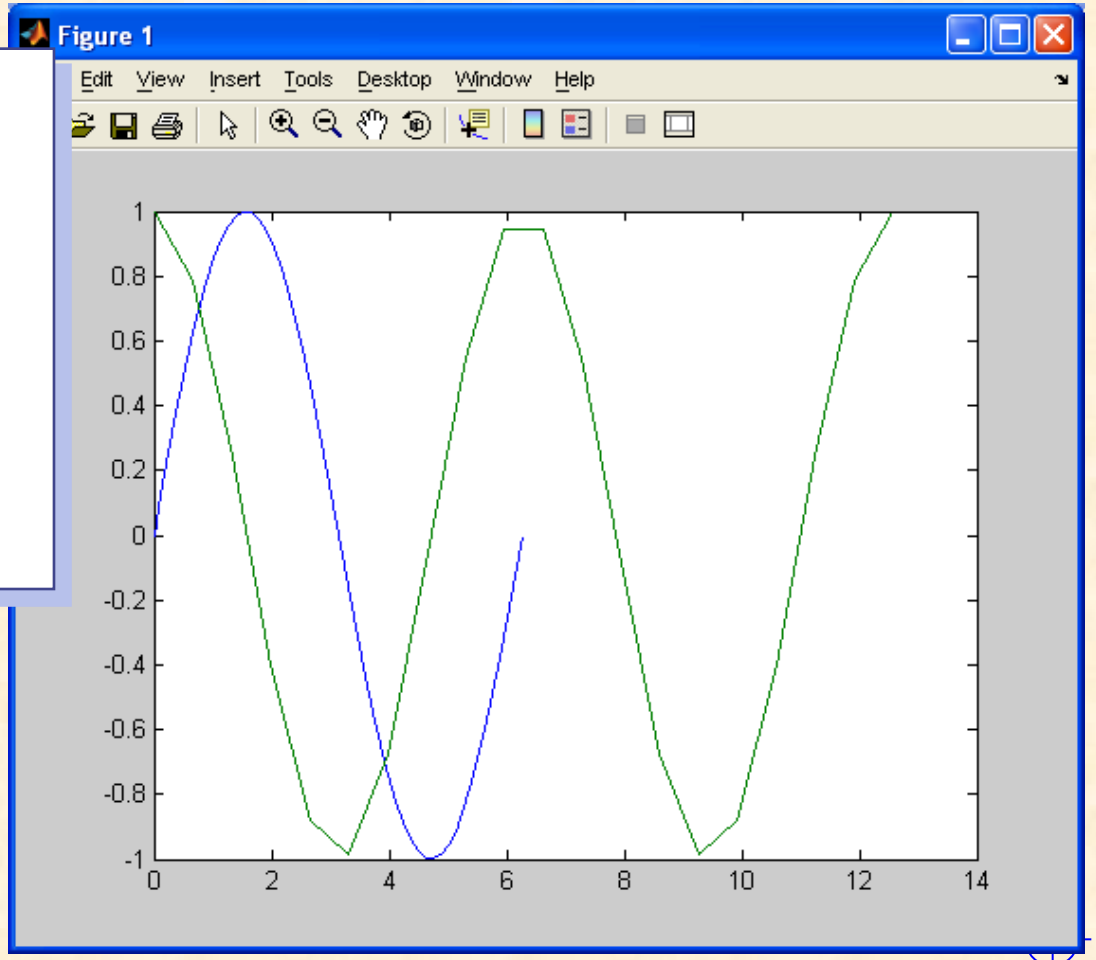
```
>> x=linspace(0,2*pi,50);

>> y = sin(x);

>> xx=linspace(0,4*pi,20);

>> yy=cos(xx);

>> plot(x,y,xx,yy)
```



"broken" green line due to small number of points

x and y are same length

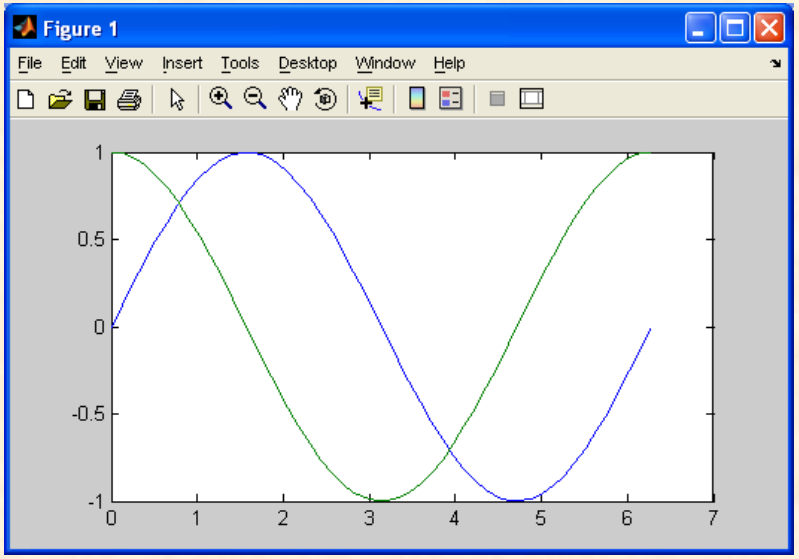
xx and yy are same length

x and xx can be different

Two more ways to draw multiple lines

- The same syntax as we showed before can be applied when x and y are matrices:
  - First case: `x` is a vector and **y is a matrix**
    Example: we want to plot both `sin` and `cos` over the *same* interval -

```
>> x=linspace(0,2*pi,50);
>> y = [sin(x)', cos(x)'];
>> plot(x,y)
```



Note:

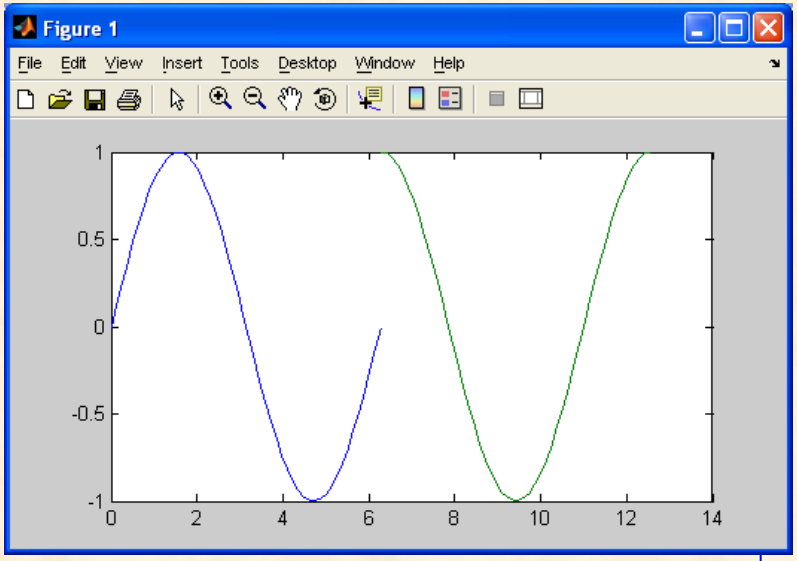
- **All Columns** of y are plotted vs. same x

Multiple lines

- Second case: **x is a matrix** and **y is a matrix**

  Example: we want to plot both sin and cos over the *different* intervals -

```
>> x=linspace(0,2*pi,50)';
>> x=[x,x+2*pi];
>> y = [sin(x(:,1)),cos(x(:,2))];
>> plot(x,y)
```

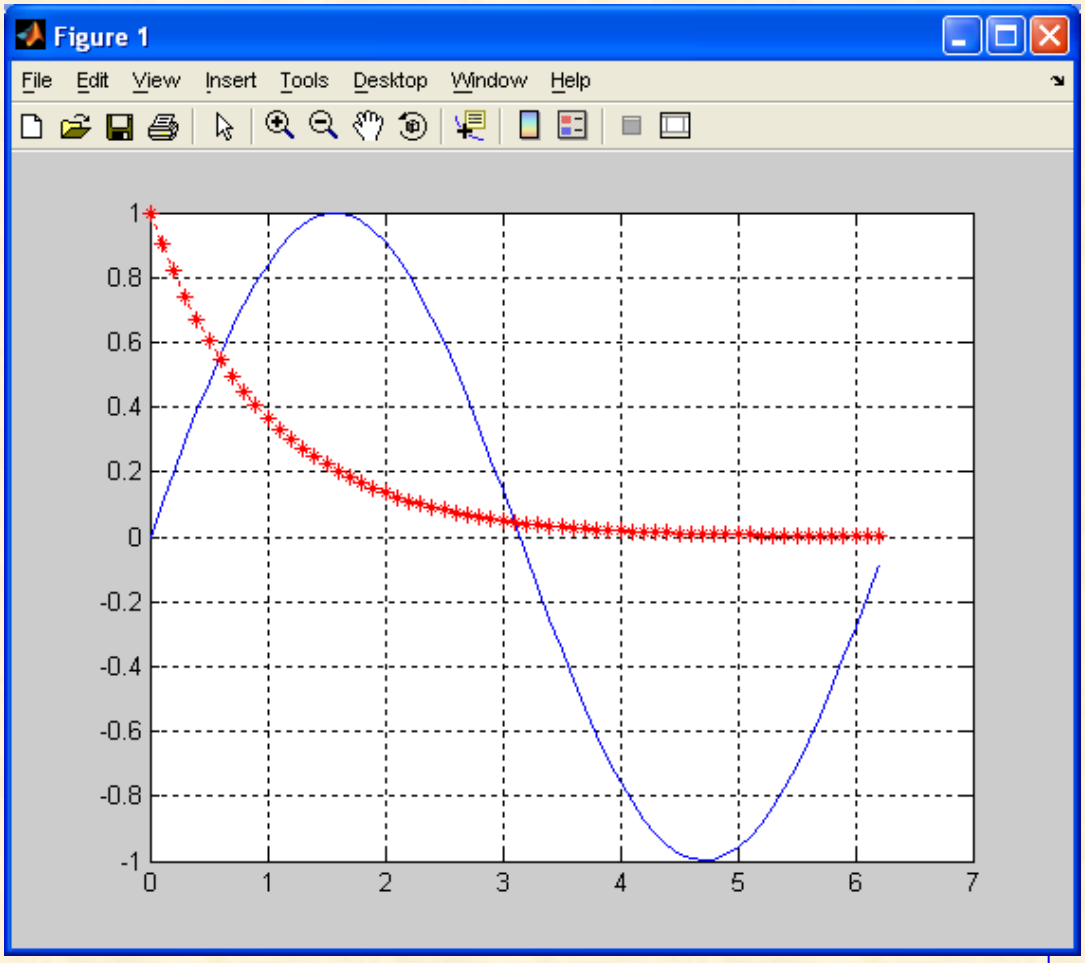


- Notes:
  - **Each Column of y is plotted vs. the *corresponding* column of x**

Adding Additional Plots to a Figure

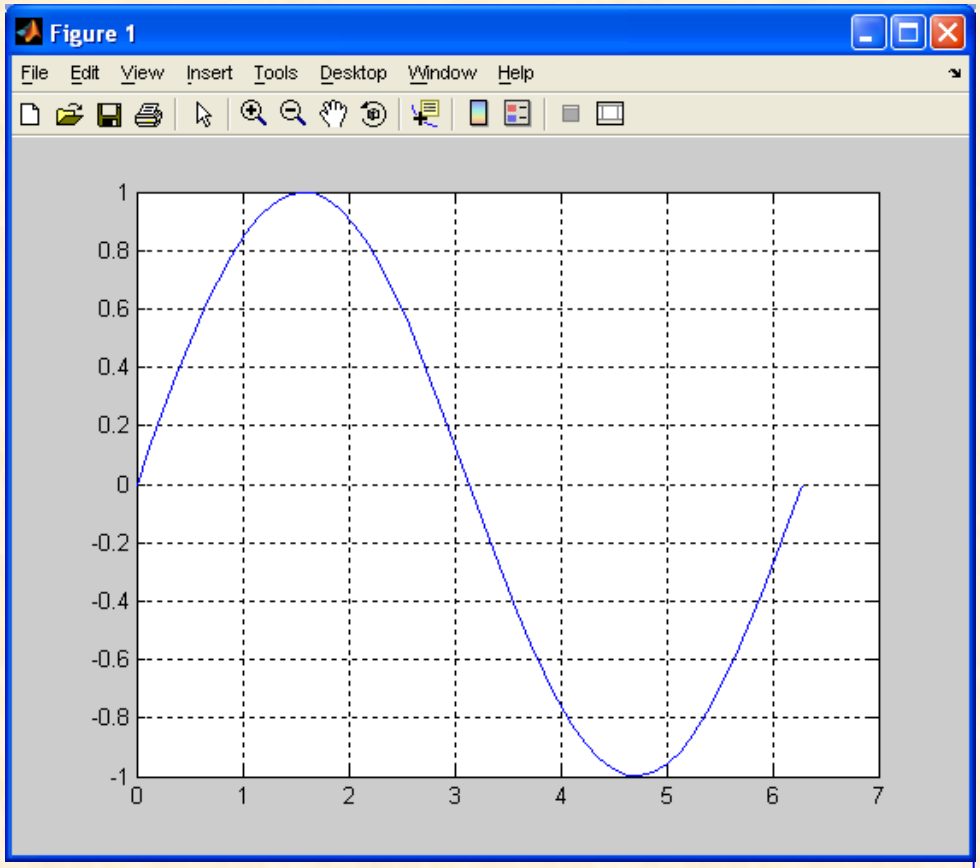
- HOLD ON holds the current plot
- HOLD OFF releases hold on current plot
- HOLD toggles the hold state

```
>> x = 0:.1:2*pi;
>> y = sin(x);
>> plot(x,y)
>> grid on
>> hold on
>> plot(x,exp(-x),'r:*')
```



Adding a Grid

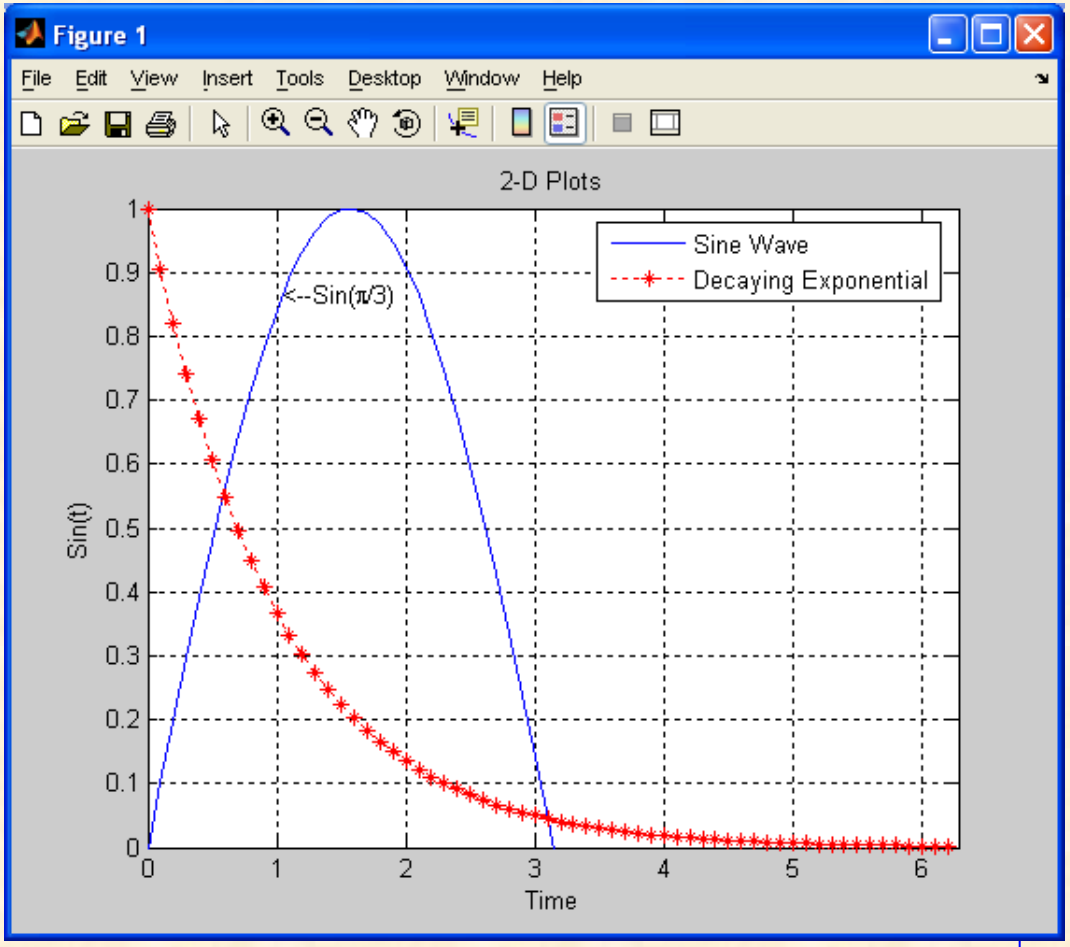
- GRID ON creates a grid on the current figure
- GRID OFF turns off the grid from the current figure
- GRID toggles the grid state




```
>> grid on
```

Graph Annotation

```
>> title('2-D Plots')
>> xlabel('Time')
>> ylabel('Sin(t)')
>> text(pi/3,sin(pi/3), ...
   '<--Sin(\pi/3)')
>> legend('Sine Wave', ...
   'Decaying Exponential')
```



Some useful symbols:

\alpha

\beta

… other Greek letters

\infty

\div

\aleph

\neq

\copyright

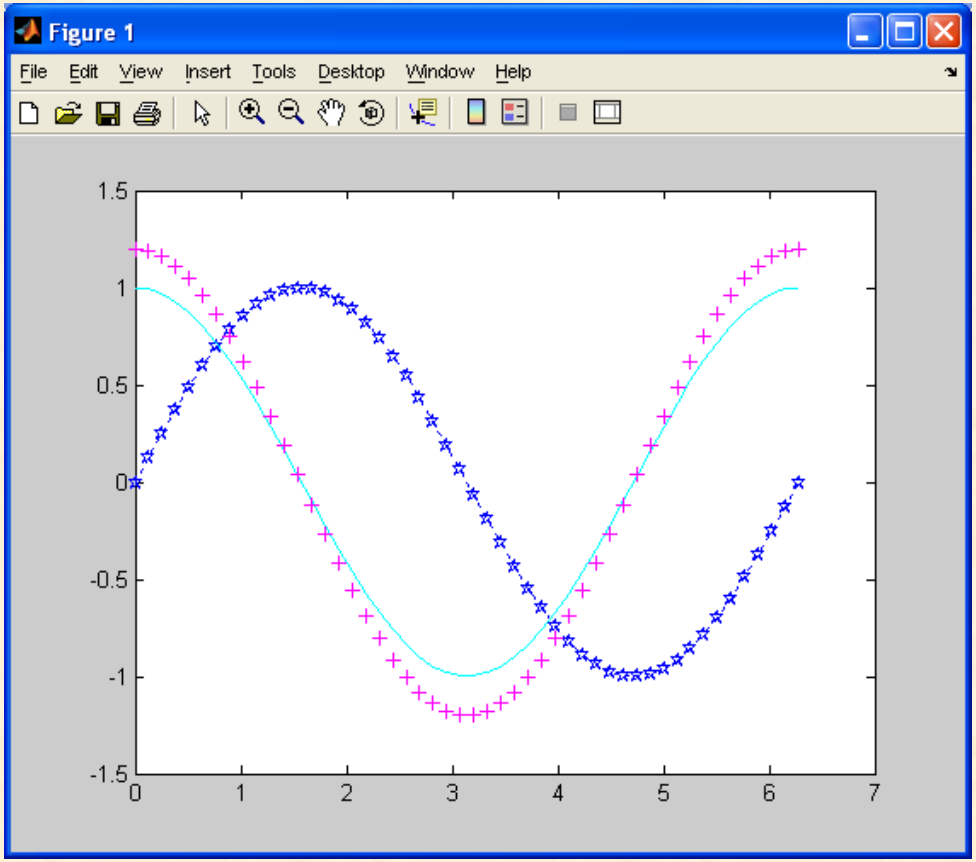
Line styles, markers and colors

Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

b	blue		.	point		-	solid
g	green		o	circle		:	dotted
r	red		x	x-mark		-.	dashdot
c	cyan		+	plus		--	dashed
m	magenta		*	star			
y	yellow		s	square			
k	black		d	diamond			
			v	triangle (down)			
			^	triangle (up)			
			<	triangle (left)			
			>	triangle (right)			
			p	pentagram			
			h	hexagram			

2-D Plotting – with various lines etc.

```
>> x = linspace(0,2*pi,50);
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,'b:p',x,z,'c-',x,1.2*z,'m+')
```



The `axis` command

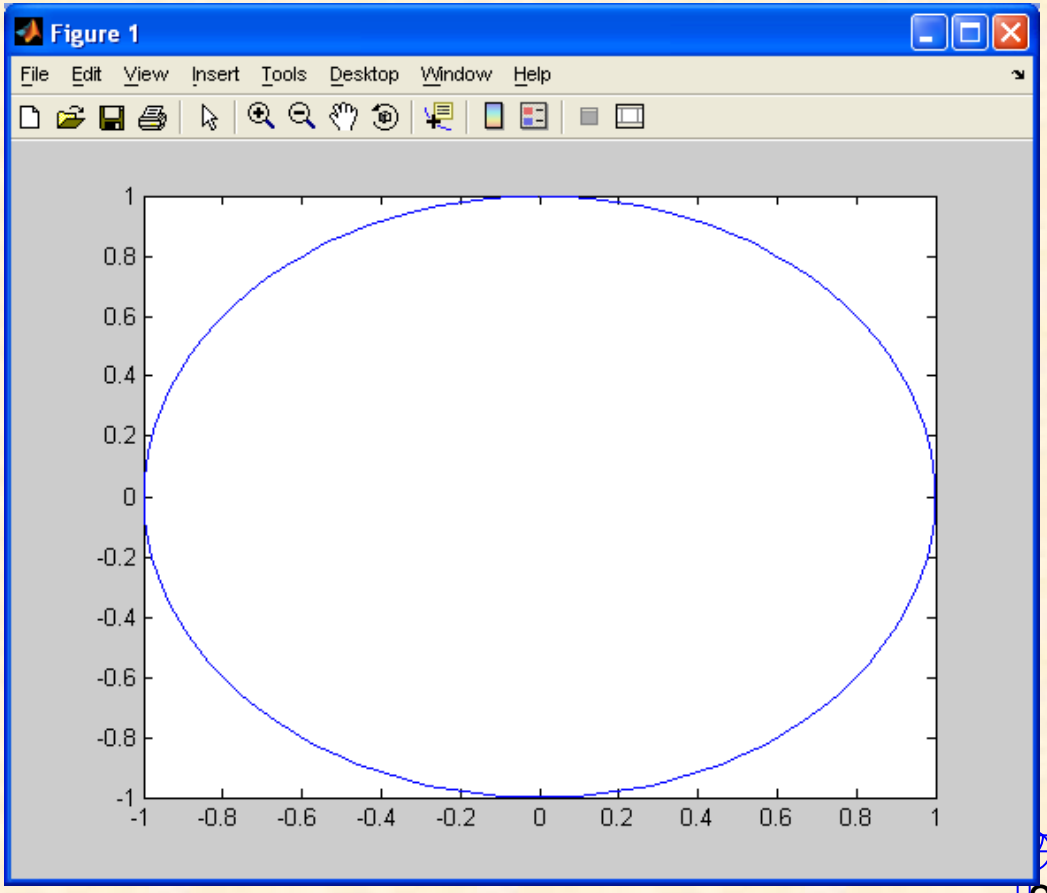
Controls many aspects of the figure:

`axis([xmin xmax ymin ymax])`	Set axis limits on the current plot
`V=axis`	Return a row vector containing the current axis limits
`axis auto`	Return axis scaling to automatic defaults
`axis manual`	Freeze axis scaling so that if hold is on, subsequent plots use the same limits
`axis tight`	Set the axis limits to the range of the plotted data
`axis fill`	Set limits and aspect ration to fill the allotted space
`axis ij`	Matrix mode: vertical axis increases from top to bottom
`axis xy`	Cartesian mode: vertical axis increases from bottom to top
`axis equal`	Set aspect ration so that equal tick mark increments on each axis are equal in size
`axis image`	Set axis limits appropriate for displaying an image
`axis square`	Make the axis box square in size
`axis normal`	Restore the current axis box to full size
`axis off`	Turn off all axis labeling, tick marks and background
`axis on`	Turn on all axis labeling, tick marks and background

Lets experiment with `axis`:

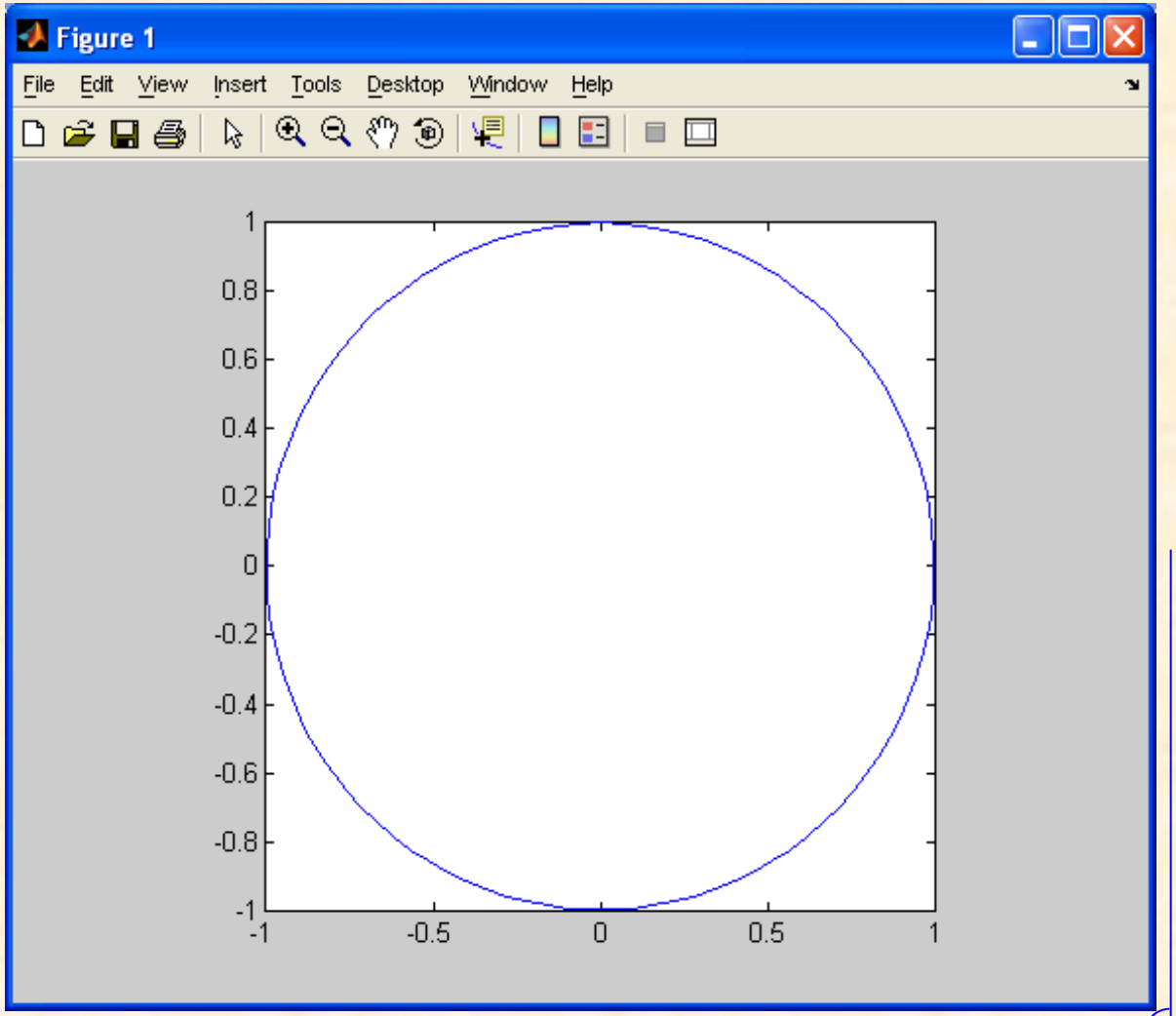
```
>> % plotting a circle:
>> x = linspace(0,2*pi,100);
>> y=sin(x);
>> z=cos(x);
>> plot(y,z)
```

- **OUCH!**
- **How can `axis` help us?**
  - **`Square`…**
  - **`Equal`…**



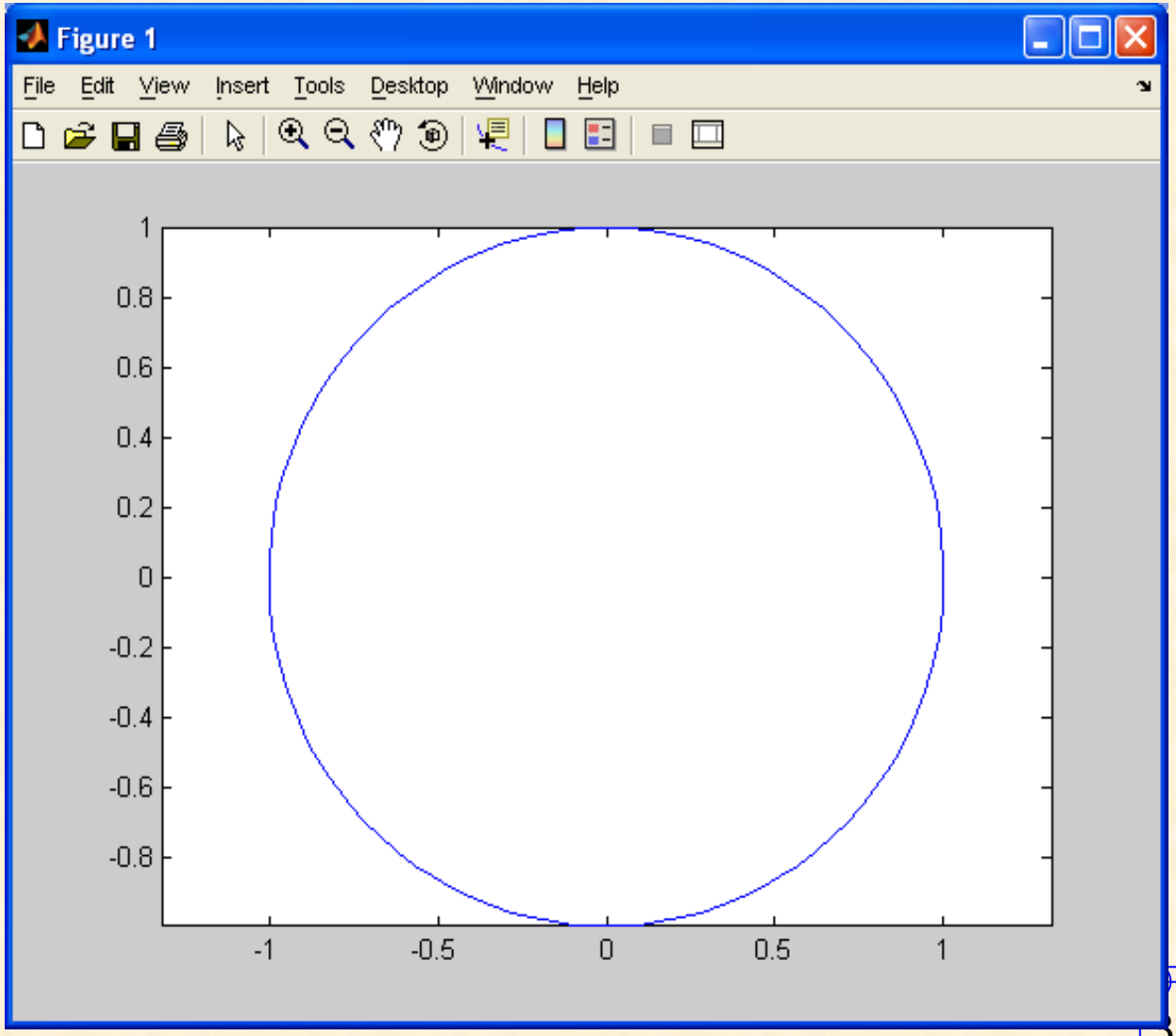
Squaring the axis

```
>> axis square
```



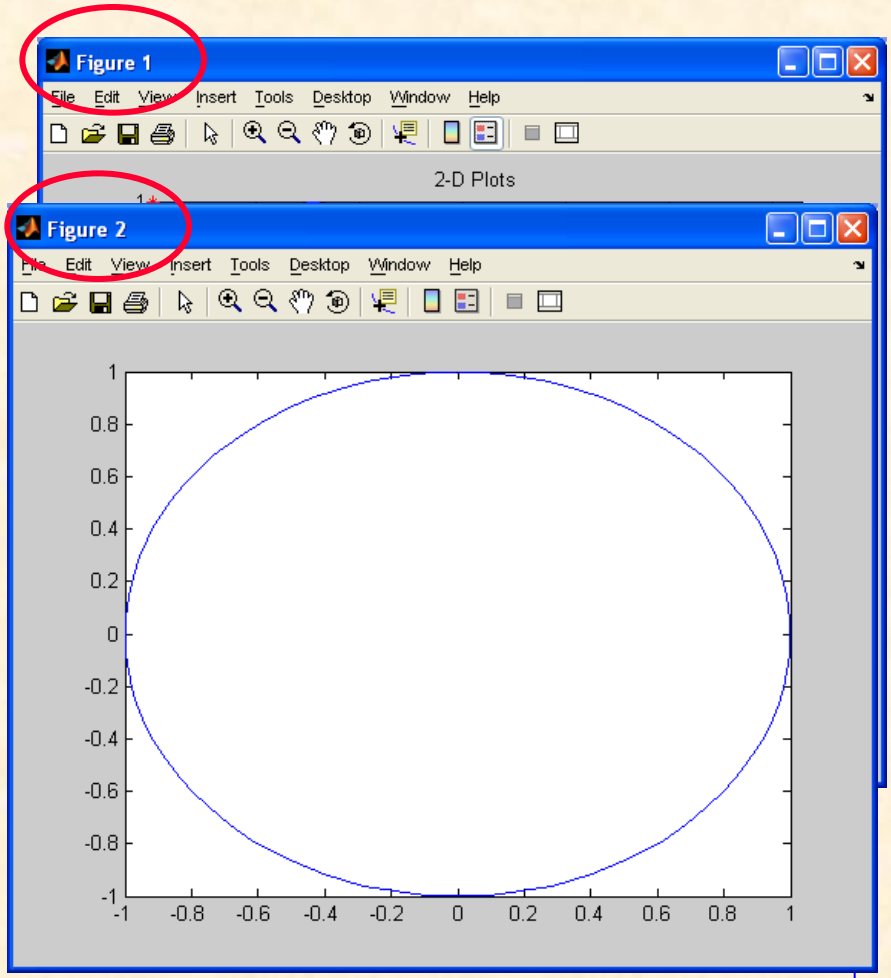
Making the aspect ratio uniform:

```
>> axis equal
```



Multiple figures:

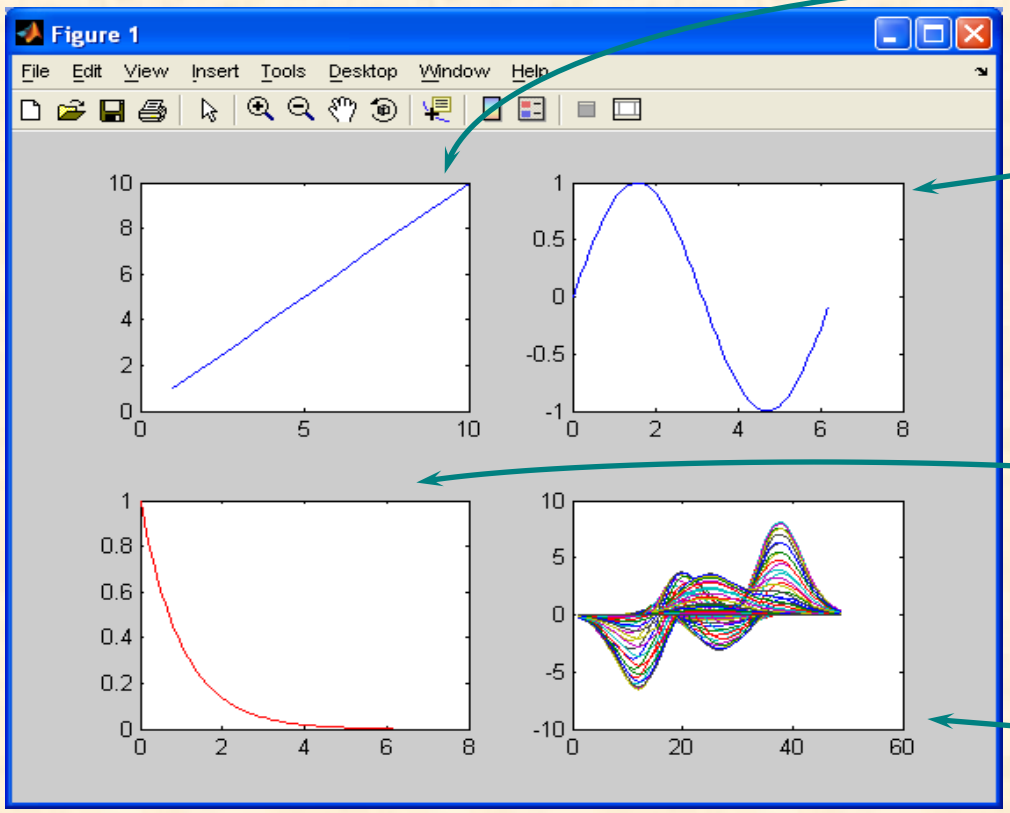
- Multiple figure windows can be opened
- `figure` command opens a new one
- To make a window active or current:
  - Click on it with the mouse
  - Or – type `figure(n)`
- To close a figure window
  - Click on the x in the corner
  - Or – type `close(n)`
- `close all` – closes all figure windows
- `clf` – erases the contents of the figure



Subplots

SUBPLOT- display multiple axes in the same figure window

```
subplot(#rows, #cols, index)
```

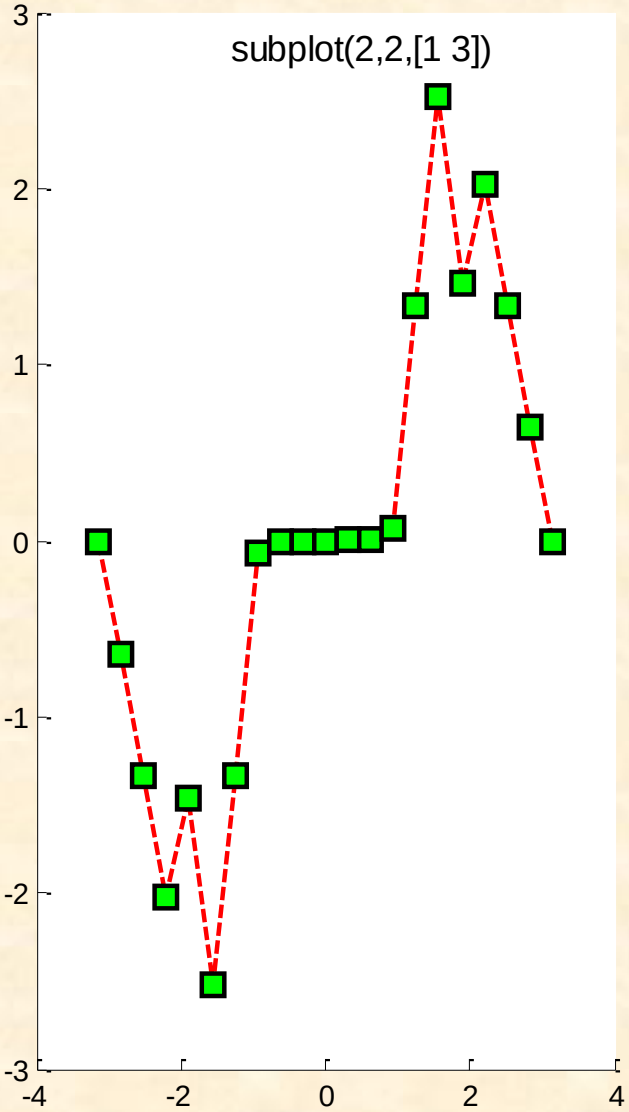


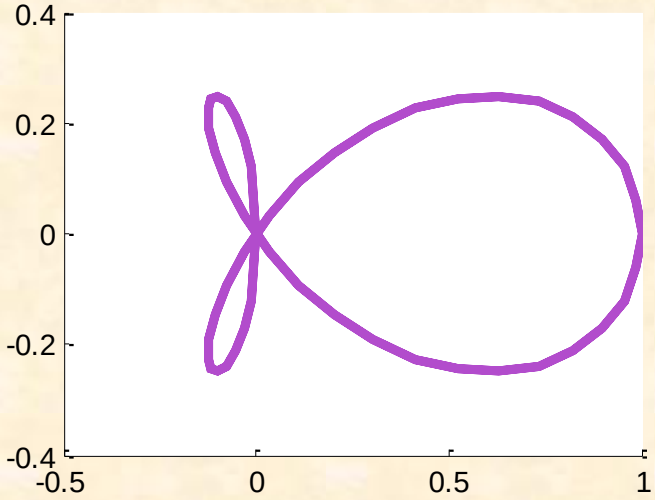

```
>> subplot(2,2,1);
>> plot(1:10)
>> subplot(2,2,2)
>> x = 0:.1:2*pi;
>> plot(x,sin(x))
>> subplot(2,2,3)
>> x = 0:.1:2*pi;
>> plot(x,exp(-x),'r')
>> subplot(2,2,4)
>> plot(peaks)
```

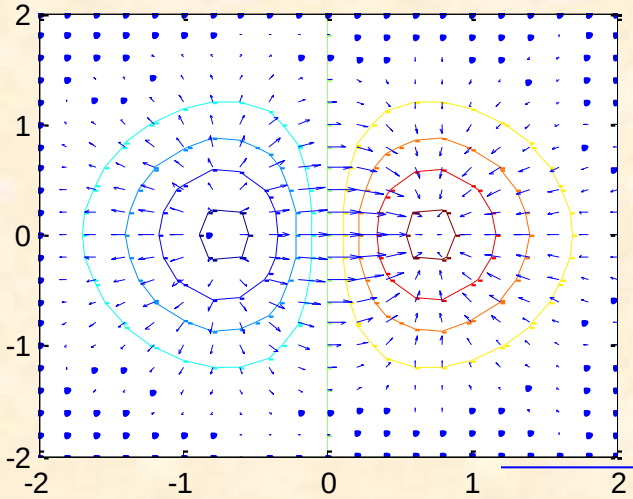
Subplots - example

```
clear all; clc;
figure
subplot(2,2,[1 3])
hold on
text(.5,2.8,'subplot(2,2,[1 3])',
'FontSize',12,'HorizontalAlignment','center')
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--
rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','
MarkerSize',10)
subplot(2,2,2)
hold on
t = 0:pi/50:2*pi;
h = polar(t,cos(t).*cos(2*t));
set(h,'LineWidth',3,'Color',[.7,.3,.8])
subplot(2,2,4)
[X,Y] = meshgrid(-2:.2:2);
Z = X.*exp(-X.^2 - Y.^2);
[DX,DY] = gradient(Z,.2,.2);
contour(X,Y,Z)
hold on
quiver(X,Y,DX,DY)
```

Subplots - example

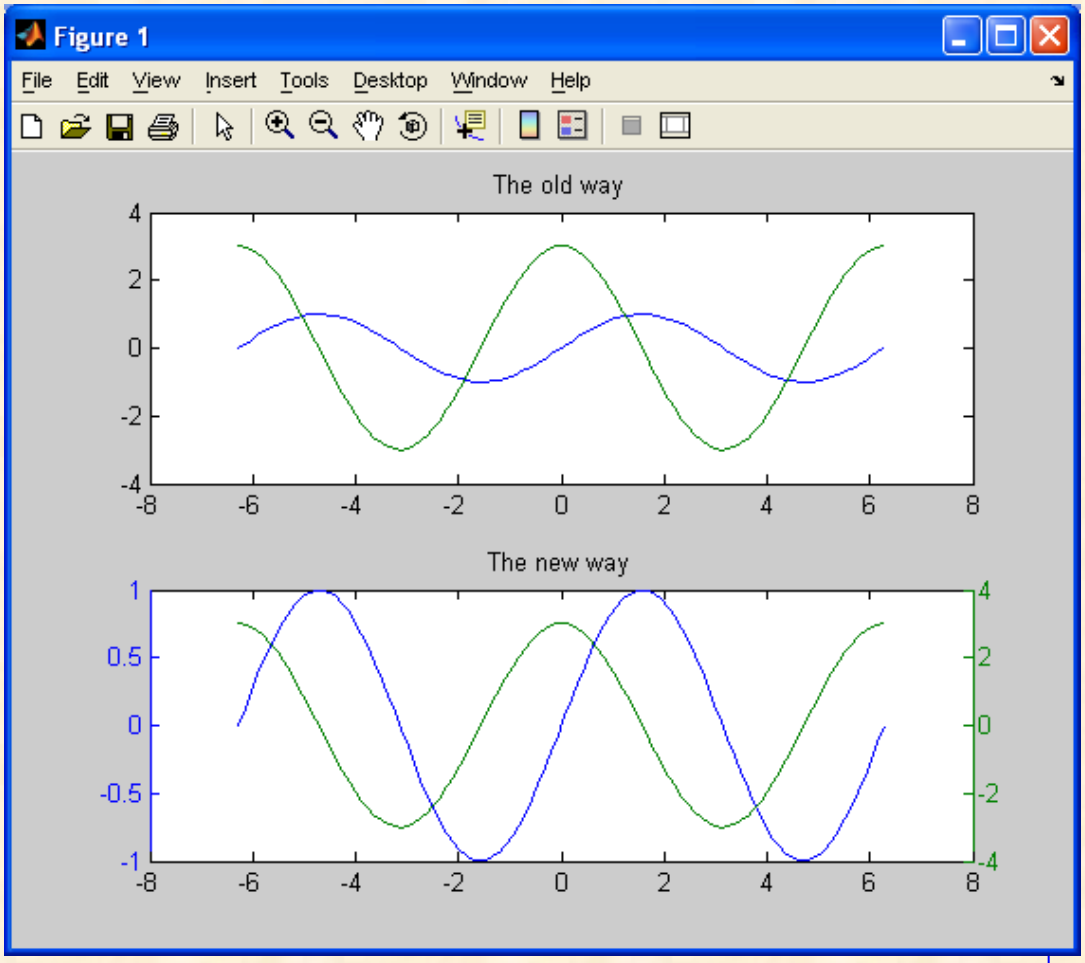






2 separate y axes: plotyy

```
>> x=linspace(-2*pi,2*pi,100);
>> y=sin(x);
>> z=3*cos(x);
>> subplot(2,1,1)
>> plot (x,y,x,z)
>> title('The old way')
>> subplot(2,1,2)
>> plotyy(x,y,x,z)
>> title('The new way')
```



Easy plotting

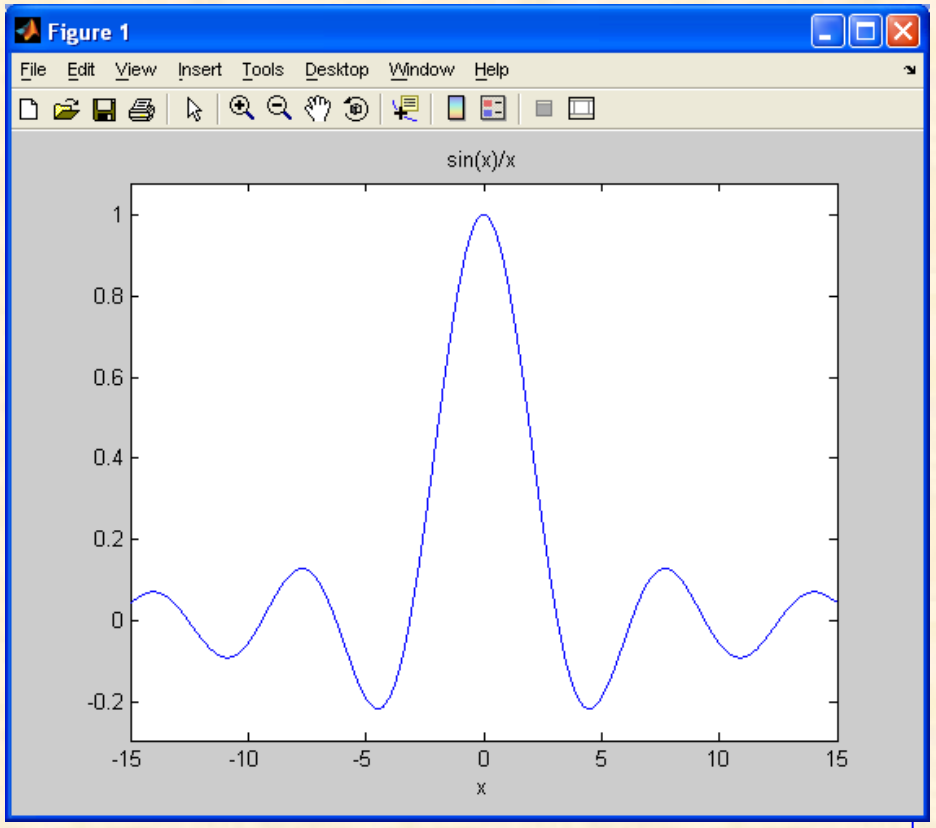
`ezplot, ezpolar`

Interpret a symbolic vector

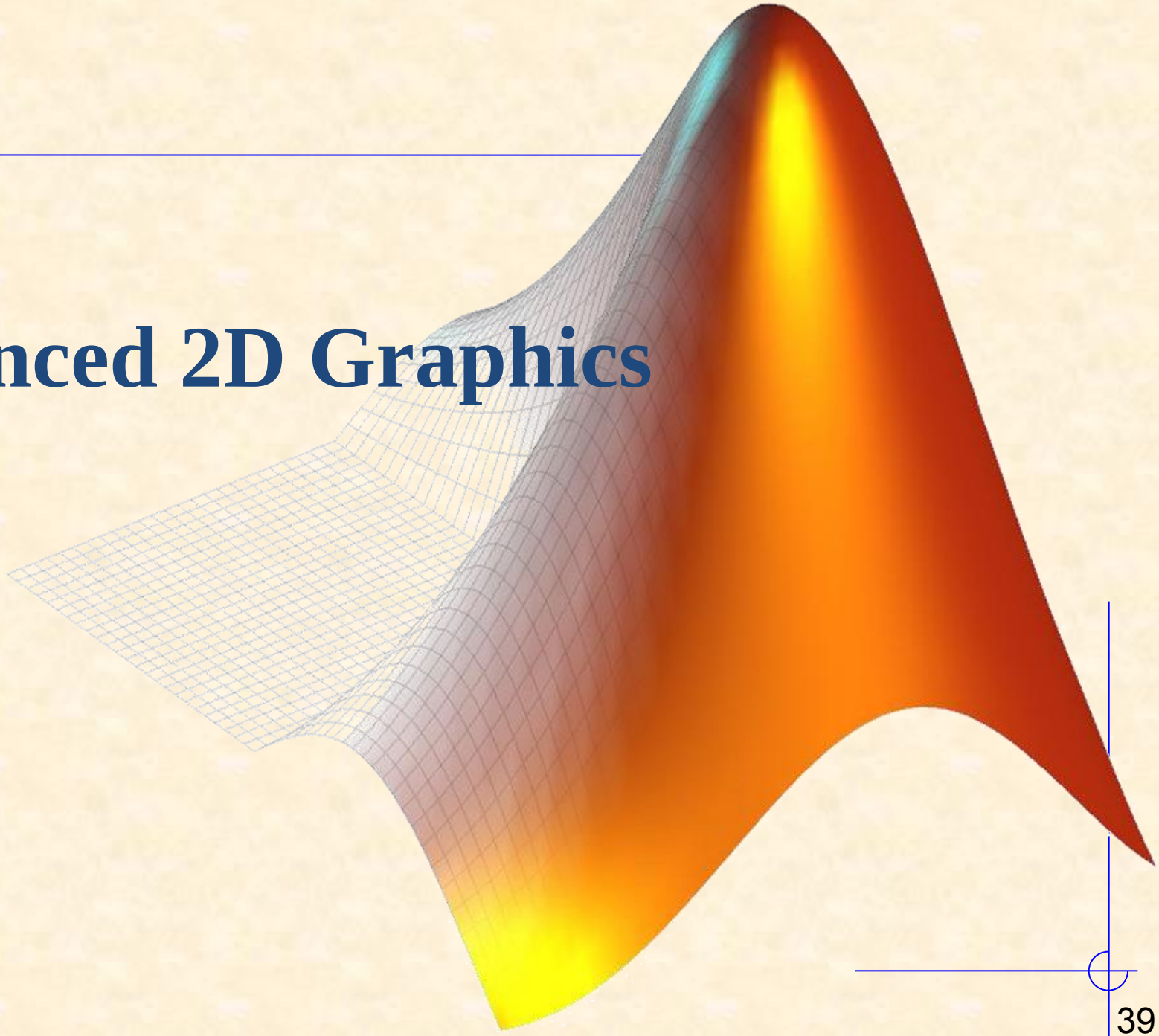
```
>> fstr='sin(x)/x';

>> ezplot(fstr,[-15,15])

>> title(fstr)
```



Advanced 2D Graphics



Area plots

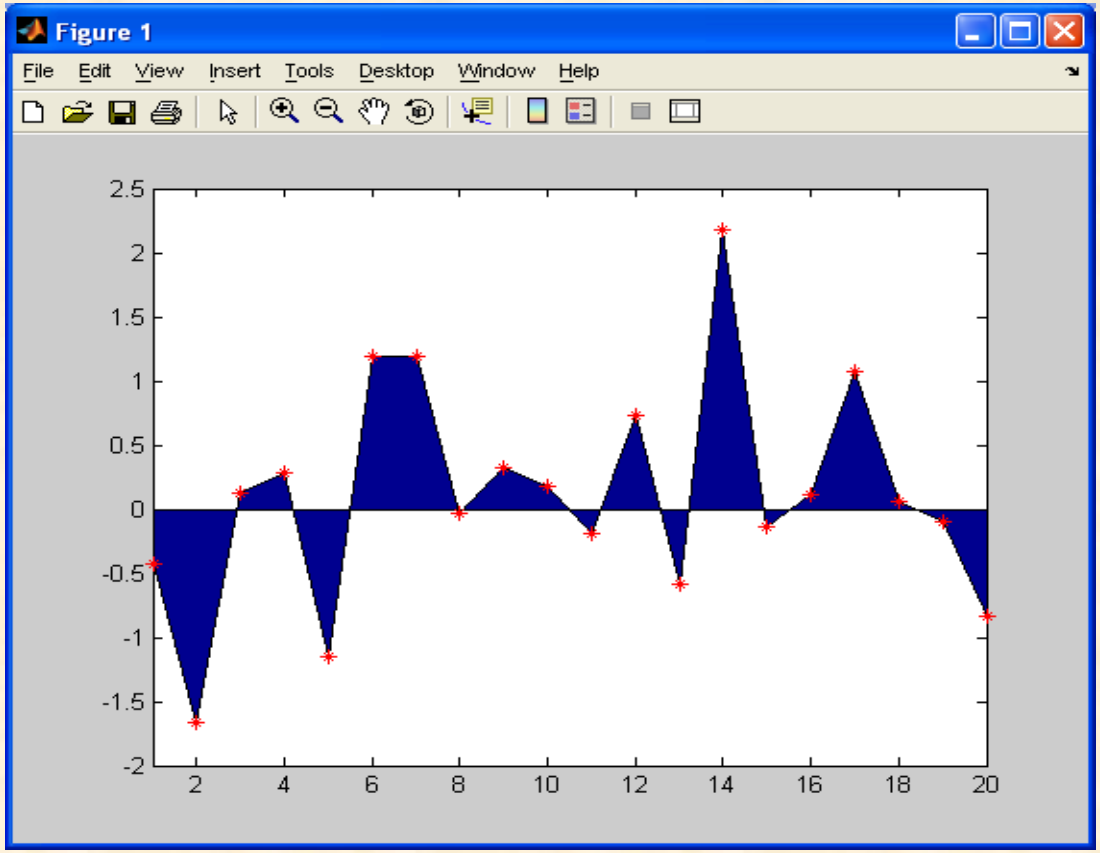
- Simply using `area` with a vector argument creates a plot with filled areas between the line and the x axis:

```
>> a=randn(1,20);

>> area(a)

>> hold on

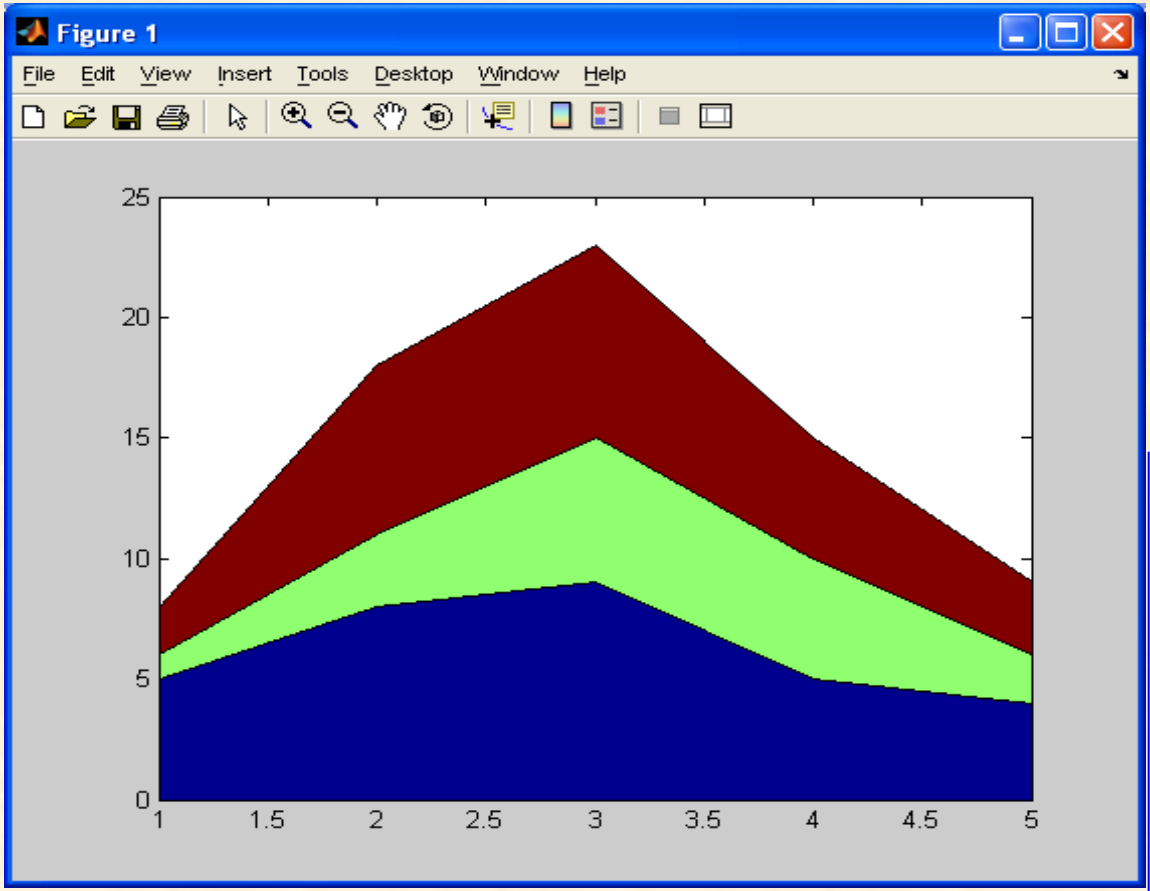
>> plot(a,'*r')
```



Area plots

- Area plots can be stacked:

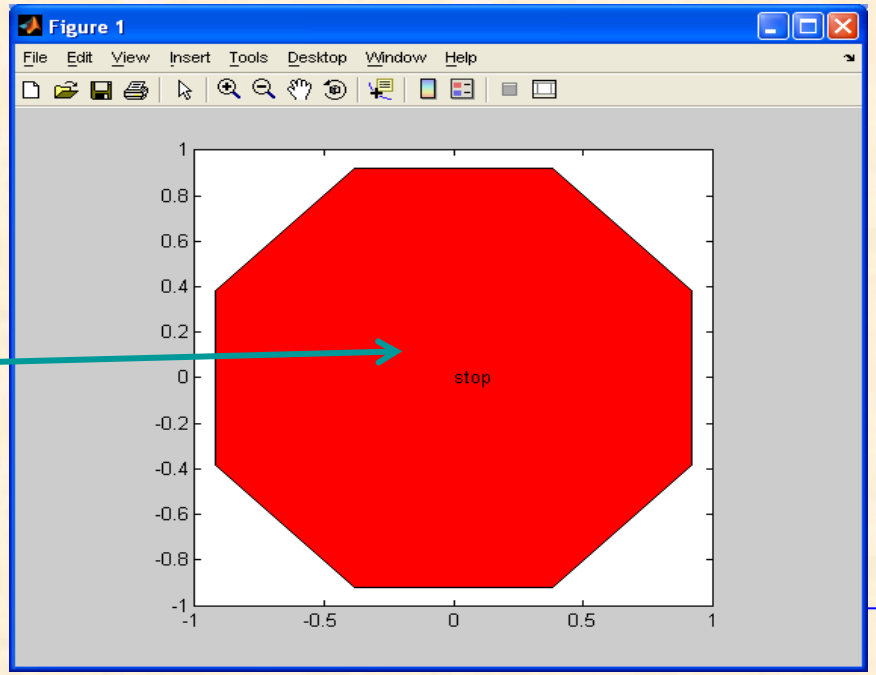
```
>> y = [5 1 2
       8 3 7
       9 6 8
       5 5 5
       4 2 3];
>> area(y)
```



Filled polygons

- The `fill` function creates filled polygons:
  `fill(x,y,'c')` fills a polygon defined by two column vectors – each x(i),y(i) pair defines a vertex
- When x and y are matrices of the same dimension, separate columns define separate polygons

```
>> t=(1:2:15)'*pi/8;
>> x=cos(t);
>> y=sin(t);
>> fill(x,y,'r')
>> axis square
>> text(0,0,'stop');
```



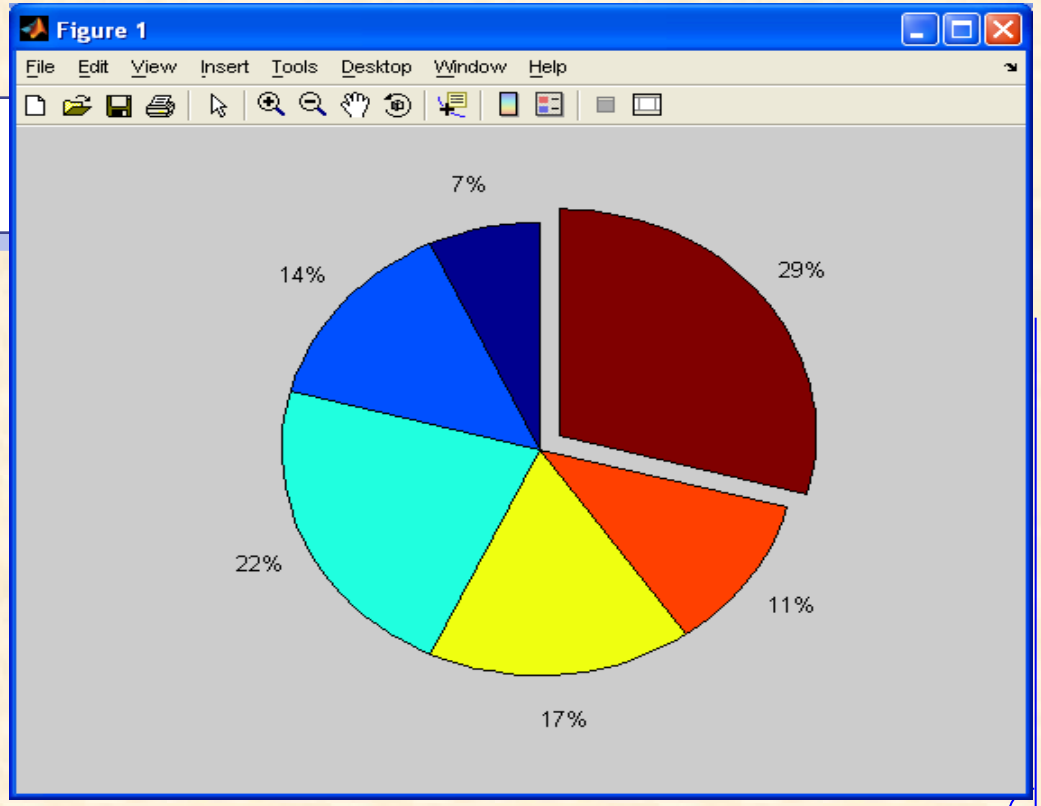
Pie charts

`pie(a,b)` creates a pie chart:

a is a vector of values

b is an optional logical vector describing the slices to be pulled out

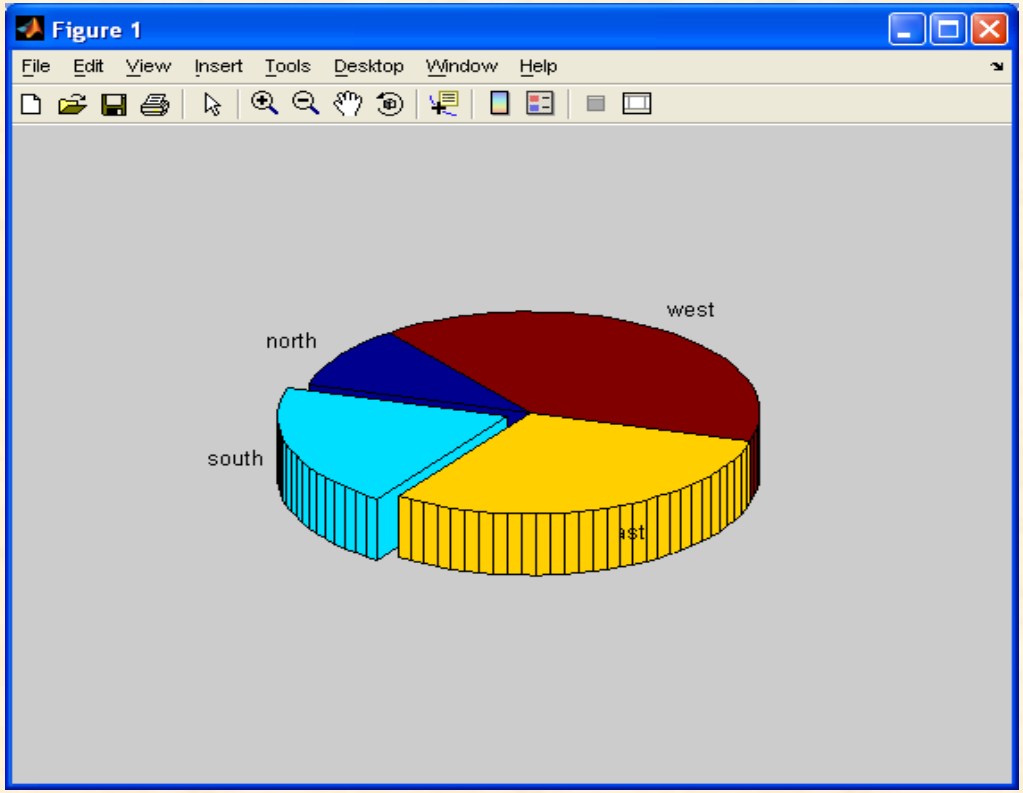
```
>> a=[.5 1 1.6 1.2 .8 2.1];
>> pie(a,a==max(a))
```



Pie charts – 3D rendering

- `pie3` takes the same arguments, but renders in 3 dimensions

```
>> a=[1 2 3 4];
>> pie3(a, [0 1 0 0 ],{'north', 'south','east','west'})
```

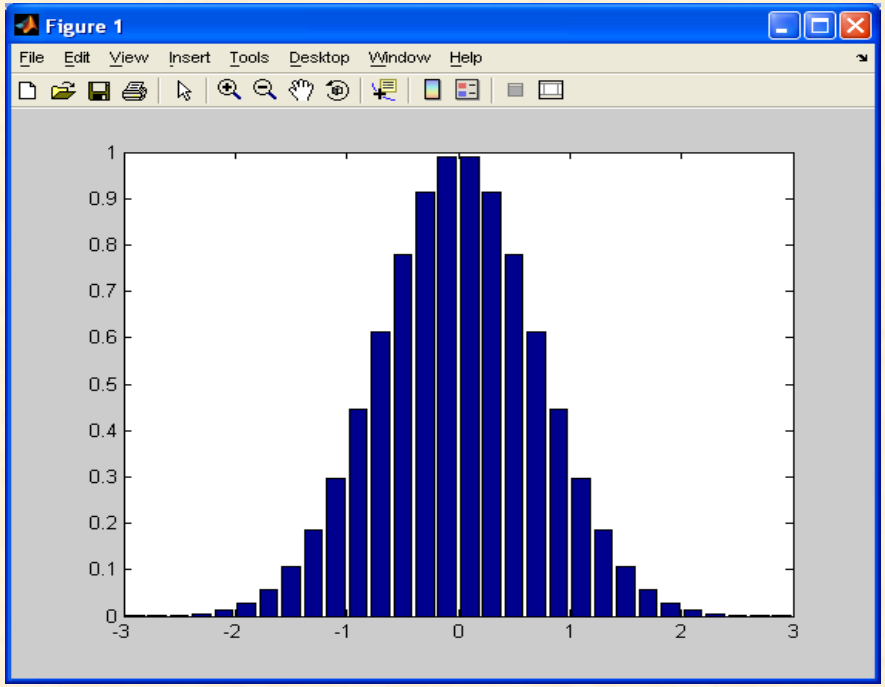


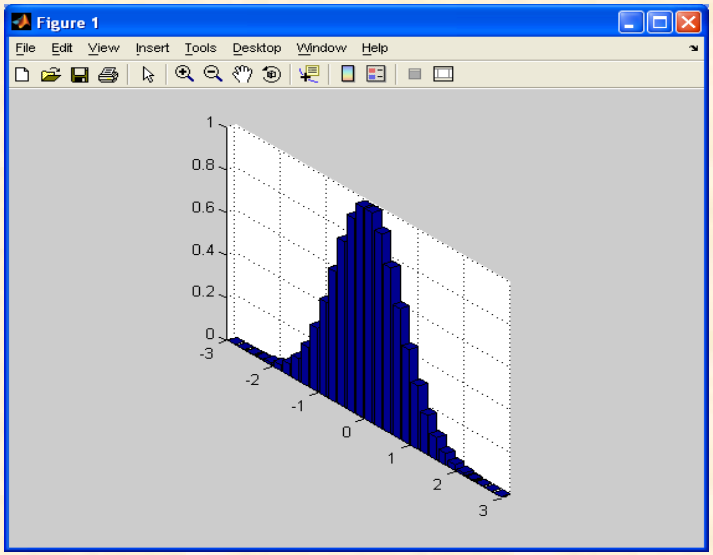
Bar plots

- Bar plots can be created in grouped or stacked form, in 2 and 3 dimensions

- *Simple* bar plots:

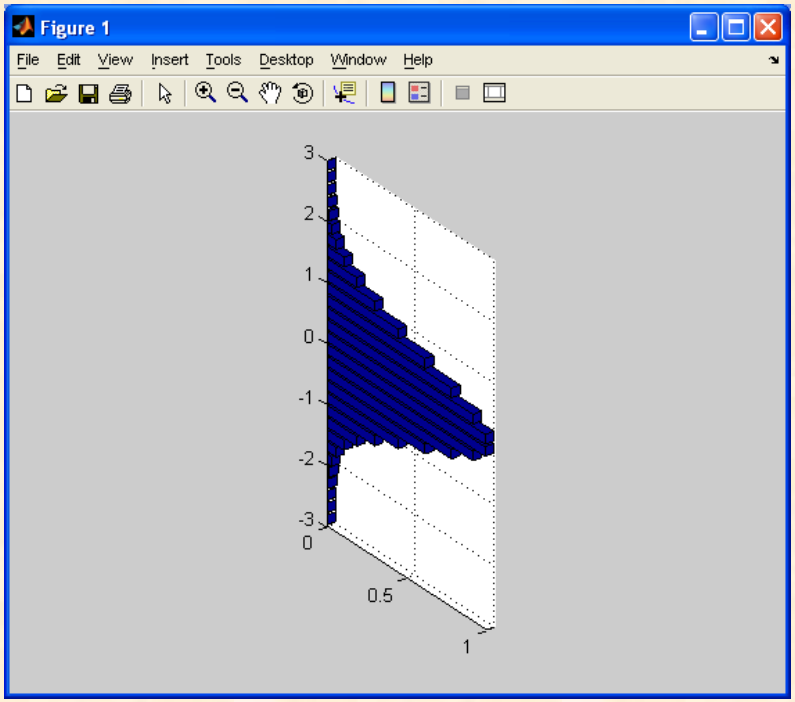
```
>> x=-2.9:.2:2.9;
>> y=exp(-x.^2);
>> bar(x,y)
>> bar3(x,y)  %3D
```

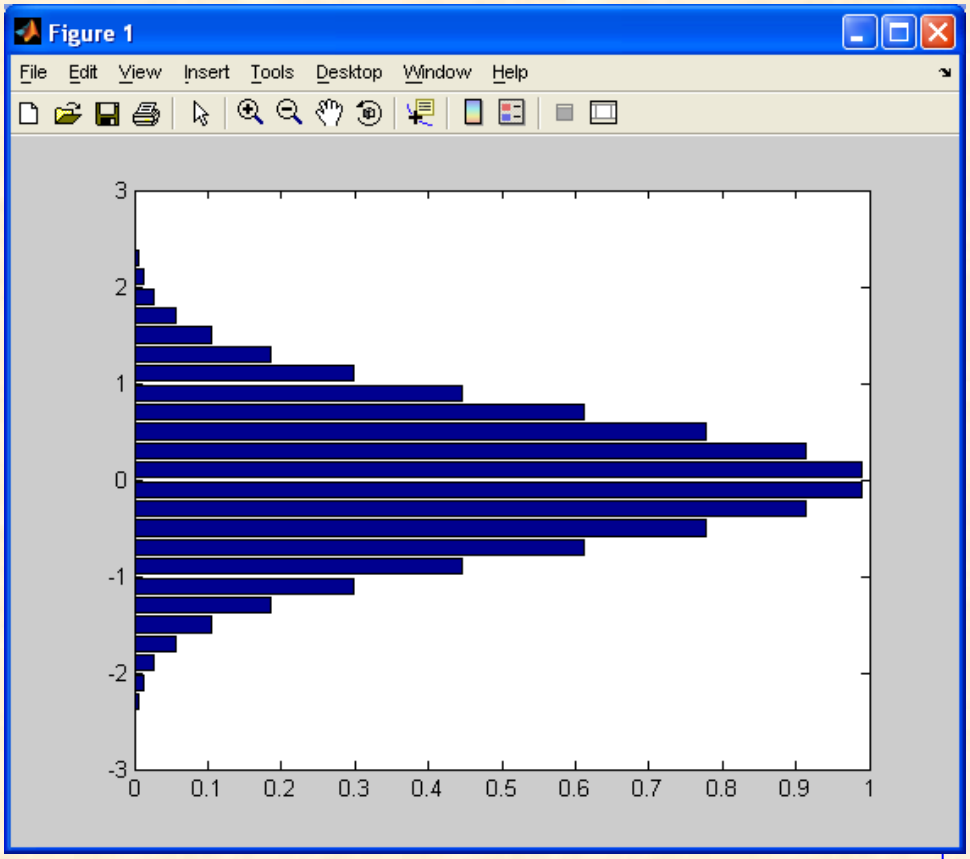




Horizontal bar plots

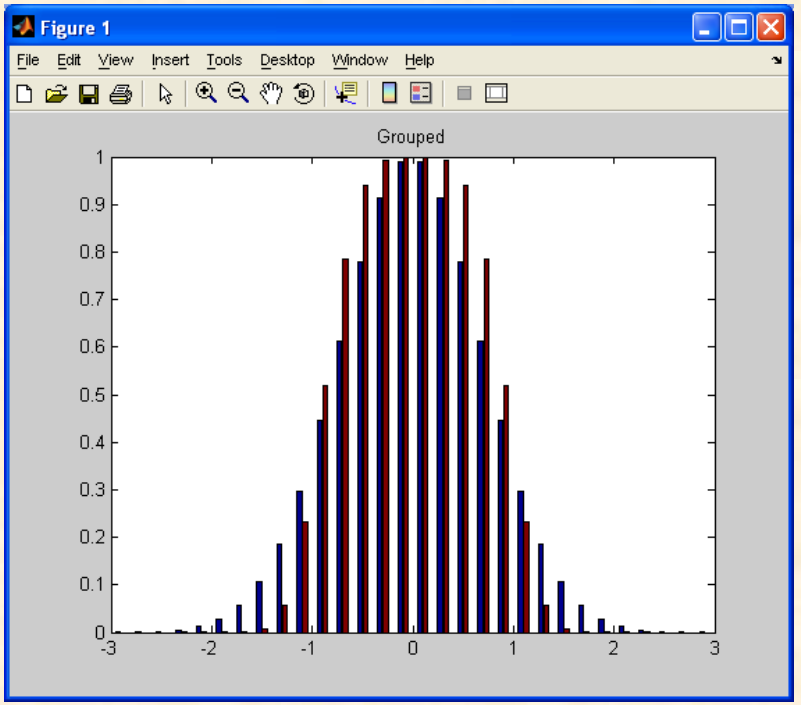
```
>> x=-2.9:.2:2.9;
>> y=exp(-x.^2);
>> barh(x,y)
>> bar3h(x,y) %3D
```

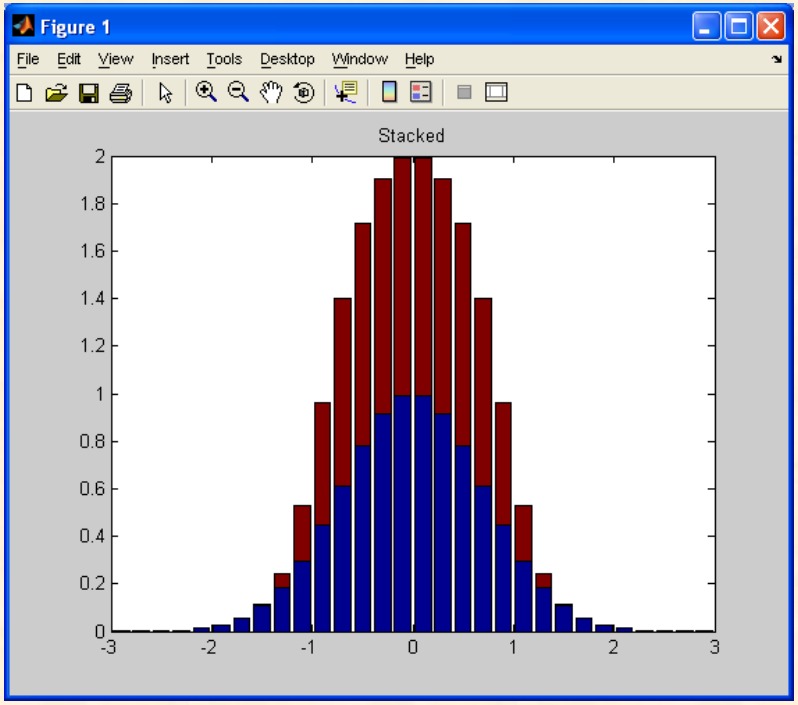




Grouped / stacked bar plots:

```
>> x=-2.9:.2:2.9;
>> y1=exp(-x.^2);
>> y2=exp(-x.^4);
>> bar(x,[y1',y2'],'grouped')
>> bar(x,[y1',y2'],'stacked')
```

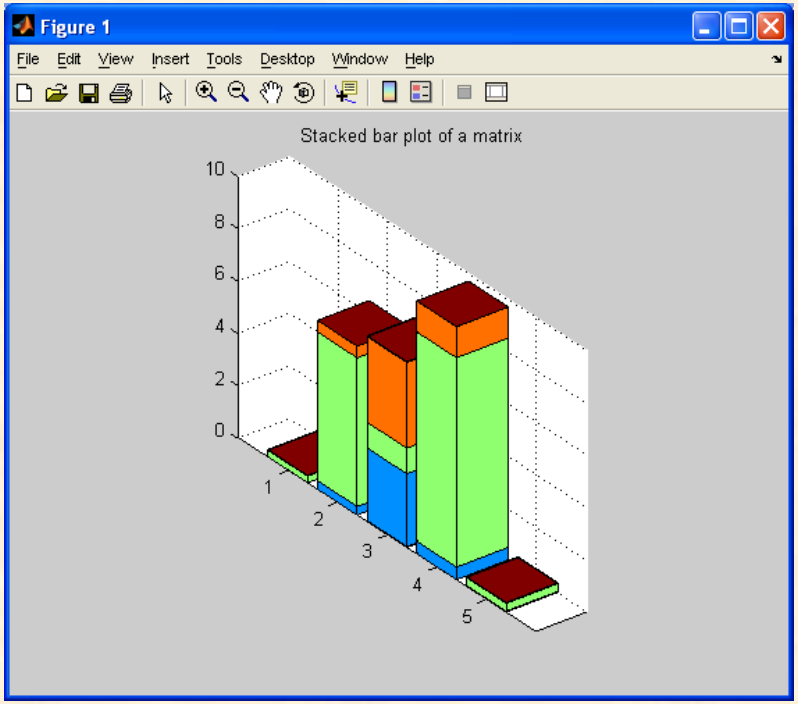


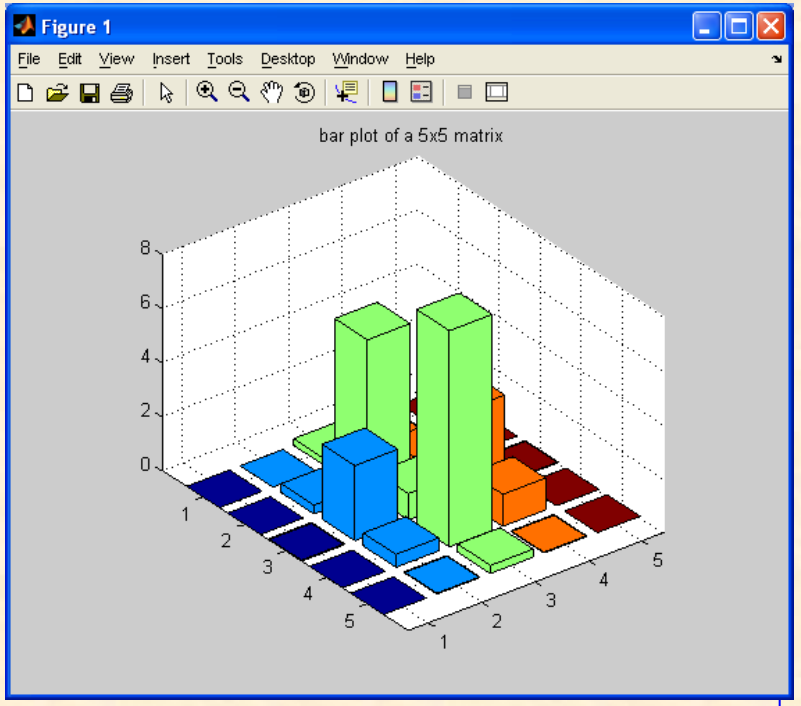


3-D bar plots:

- If not ‘grouped’ or ‘stacked’ then the default is a 3-D plot:

```
>> bar3(abs(peaks(5)))
>> bar3(abs(peaks(5)),'stacked')
```





Note: **`peaks` is a sample function of two variables – the command `peaks(5)` gives a 5x5 matrix.**

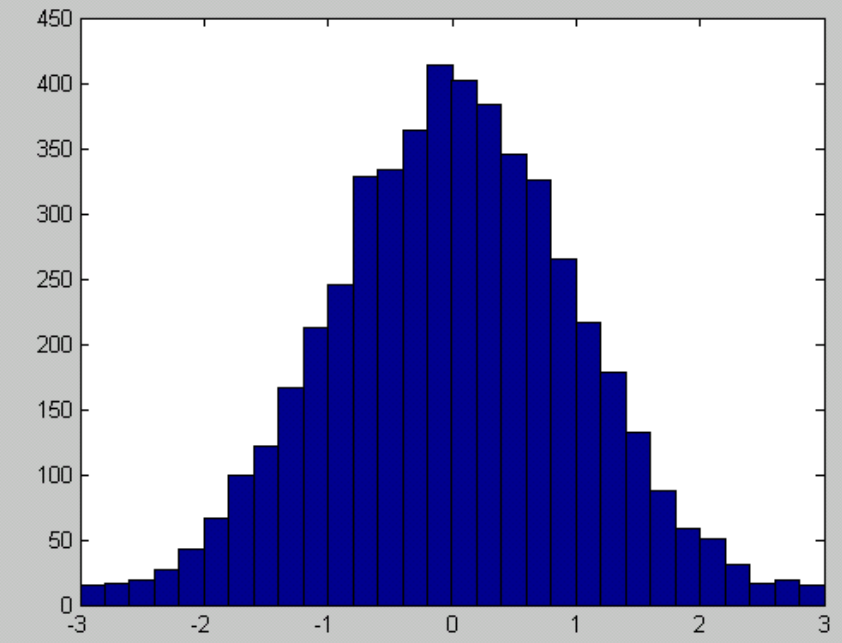
Histograms

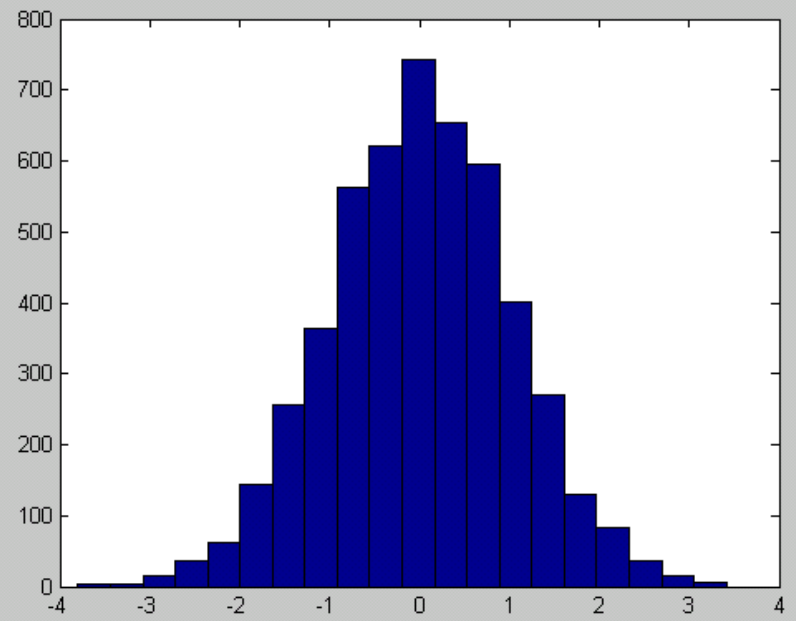
- Histograms illustrate the distribution of values in a vector

hist (y) - uses 10 bins

hist (y, n) - uses n bins

hist (y, x) - uses the bins whose center values are specified in the vector x





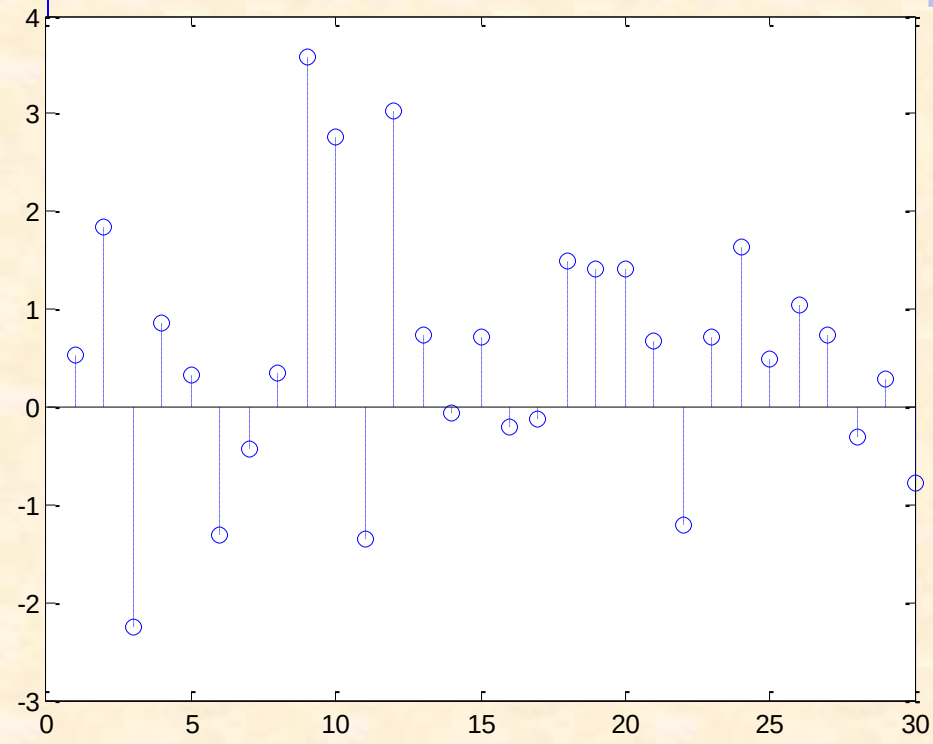

```
>> y=randn(5000,1);
>> hist(y,20)
>> hist(y,-2.9:0.2:2.9)
```

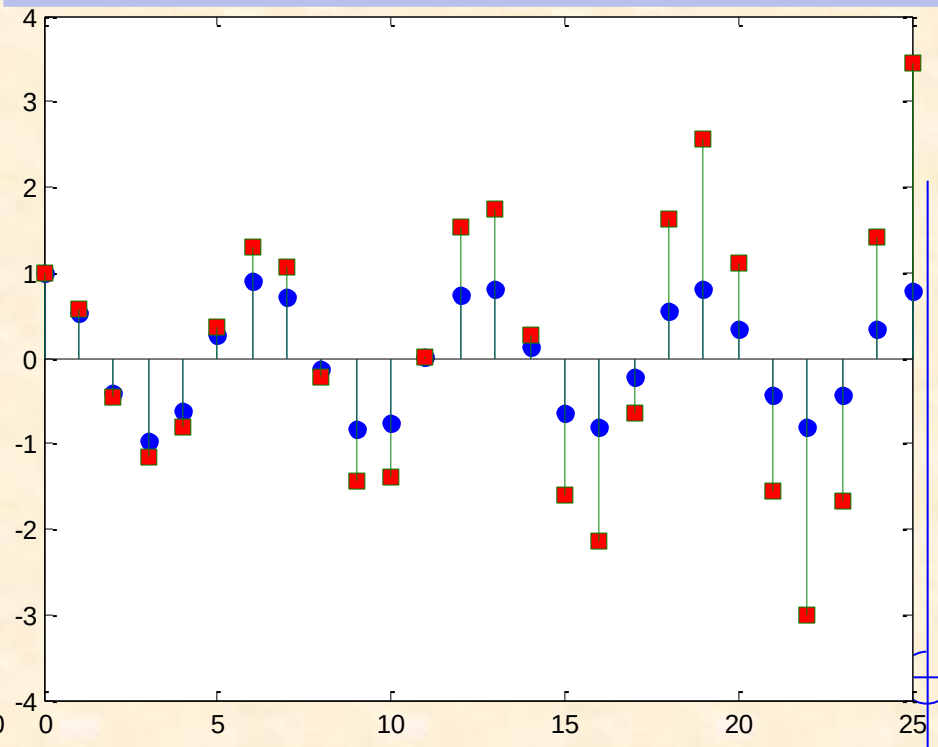
Stem plots

Stem plots are convenient for sampled data **`stem(x,y,'linespecs')`**

```
>> a=randn(30,1);
>> stem(a,':')
```

```
x = 0:25;
y = [exp(-0.01*x).*cos(x);exp(.05*x).*cos(x)]';
h = stem(x,y);
set(h(1),'MarkerFaceColor','blue')
set(h(2),'MarkerFaceColor','red','Marker','square')
```

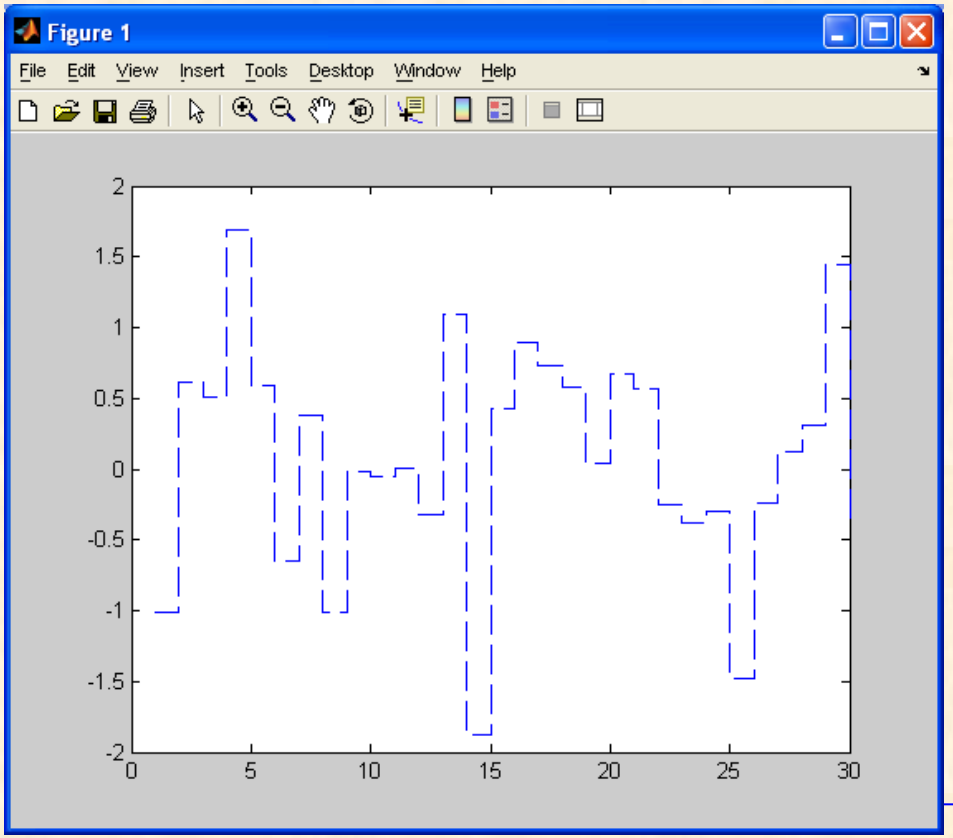




Stair plots

- **Staircase plots draw ZOH (Zero Order Hold) of sampled data**
  - Format is `stairs(x,y,'linespec')` as in the plot command
  - Multiple lines are not allowed
-

```
>> a=randn(30,1);
>> stairs(a,'--')
```

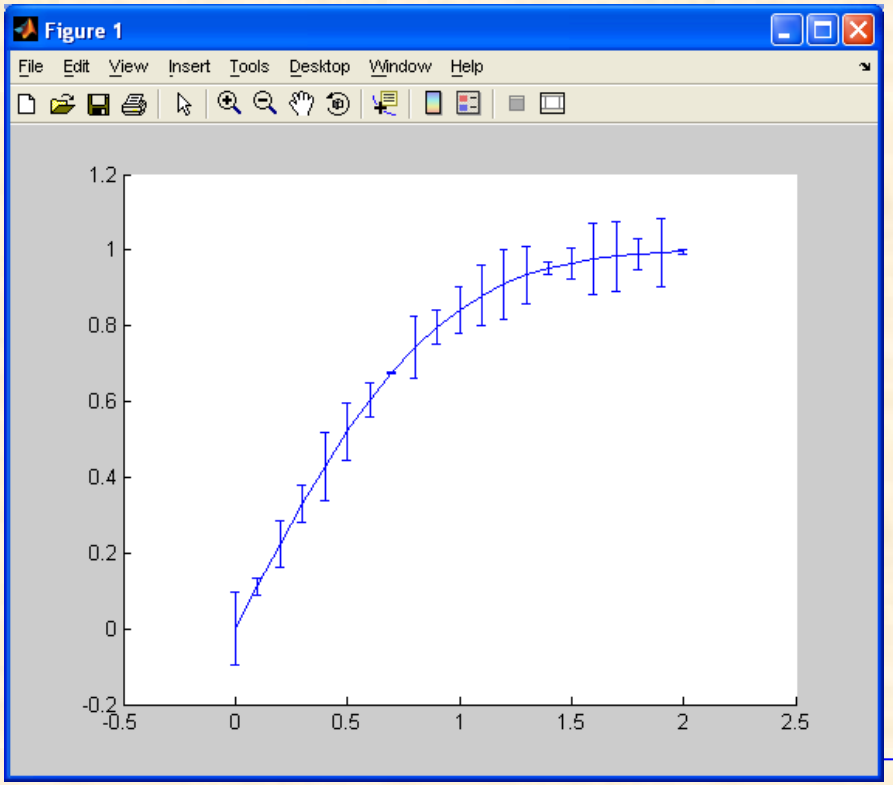


Error bars

`errorbar` plots a *line* together with error bars

- Format is
  `errorbar(x,y,l,u)` – nonsymmetric error bars
  `errorbar(x,y,e)` – symmetric error bars
- Multiple lines are allowed, using multiple columns for each argument
-

```
>> x=linspace(0,2,21);
>> y=erf(x);
>> e=rand(size(x))/10;
>> errorbar(x,y,e)
```

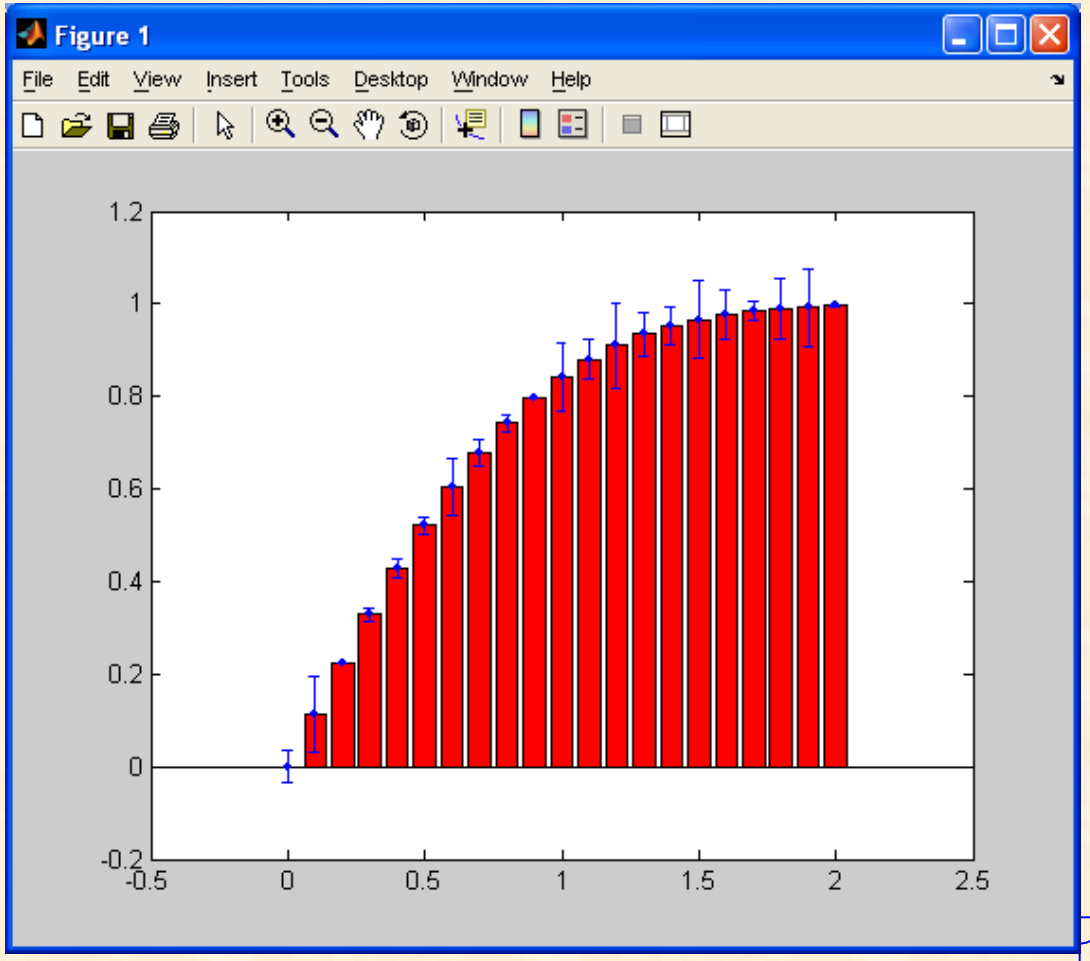


Error bars – cont.

To plot a *bar graph* with error bars:

- Plot bars
- Hold
- Use `errorbar` – marking dots only

```
>> x=linspace(0,2,21);
>> y=erf(x);
>> e=rand(size(x))/10;
>> bar(x,y,'r');
>> hold on
>> errorbar(x,y,e,'.')
```

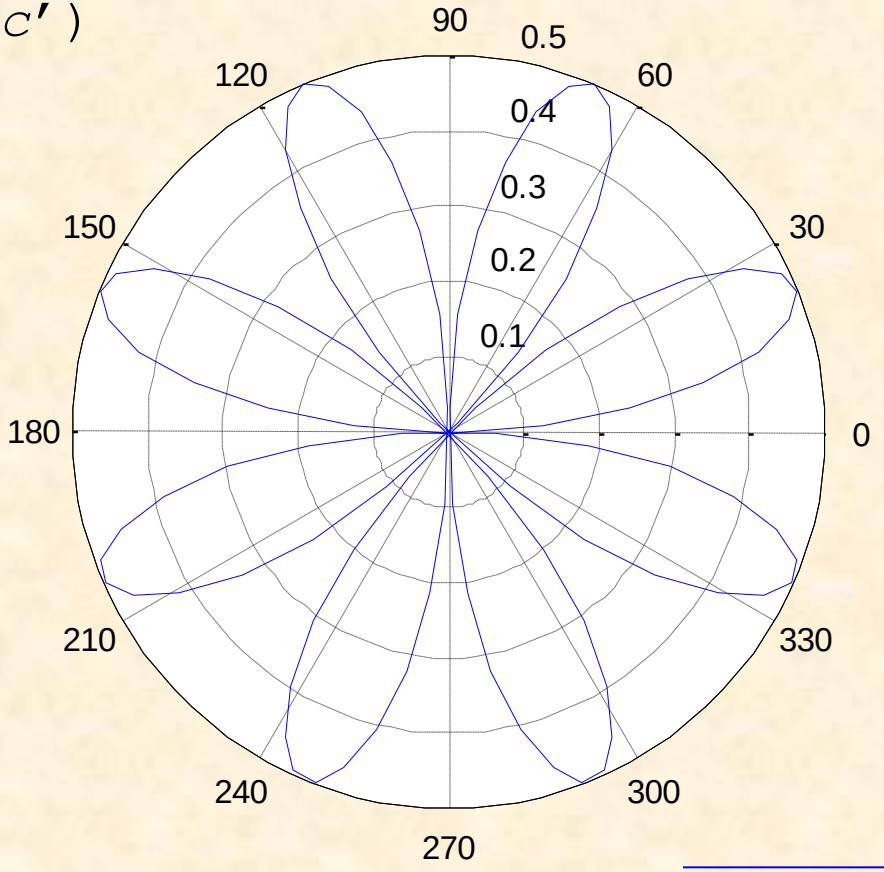


Polar plots

- `polar` does simple plotting in polar coordinates
- Format:

`polar(ang,r,'linespec')`

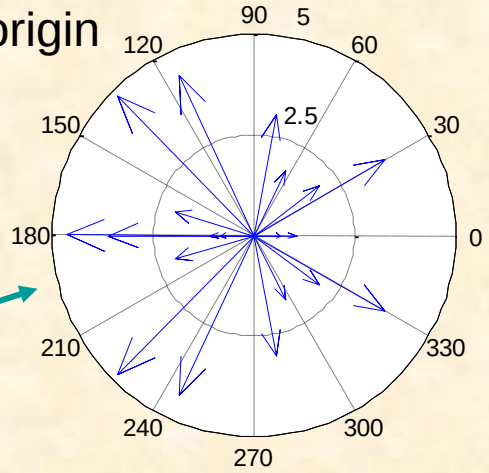
```
>> ang=linspace(0,2*pi);
>> r=sin(2*ang).*cos(2*ang);
>> polar(ang,r)
```

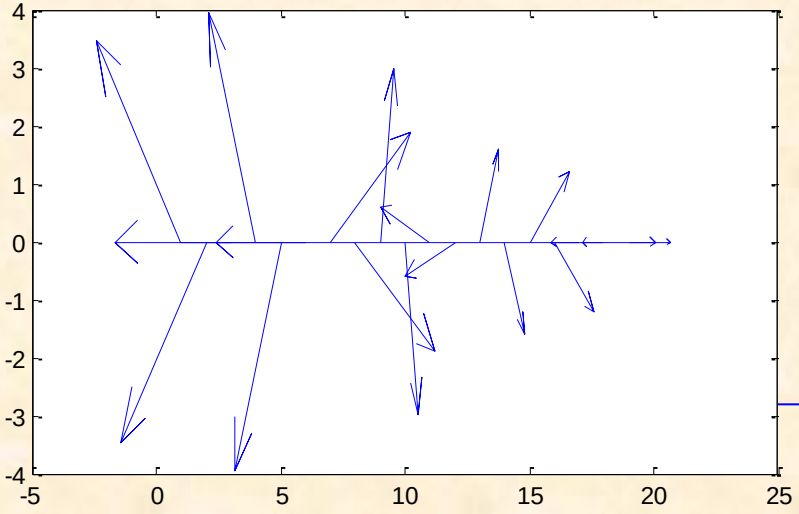


Plotting complex data

- Complex numbers can be plotted as vectors
  `compass(z)` – as arrows emanating from the origin
  `feather(z)` – arrows on the x axis

```
>> z=eig(randn(20));
>> subplot(2,1,1)
>> compass(z)
>> subplot(2,1,2)
>> feather(z)
```

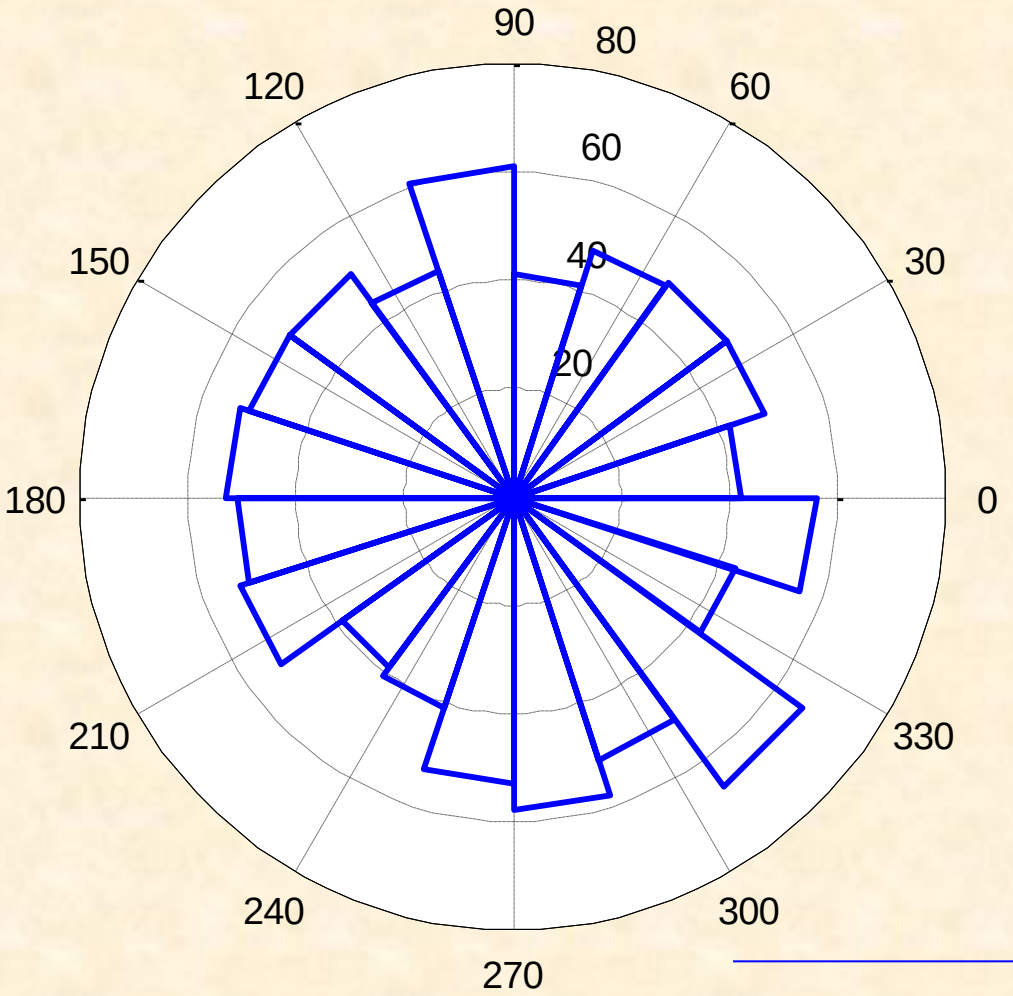




Plotting angle histograms

- `rose` **is similar to** `hist` **– only the histogram is polar and the range is** `0` **to** `2*pi`
- **Default is 20 bins**

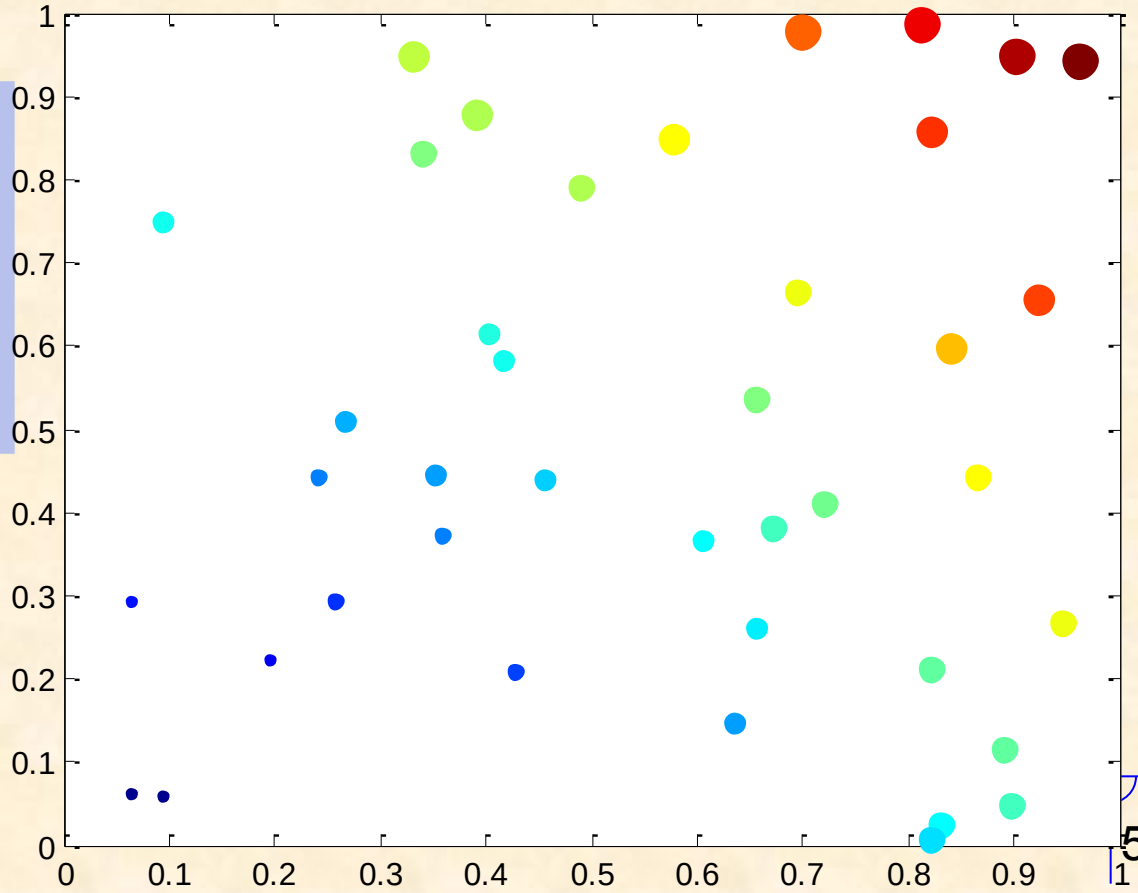
```
>> v=randn(1000,1)*pi;
>> rose(v)
```



Scatter plots:

- `scatter` **plots circles at data points, where the circle size and color can be specified**
- **The areas of each circle is in points^2, where 'points' are the units used to measure font sizes, etc.**

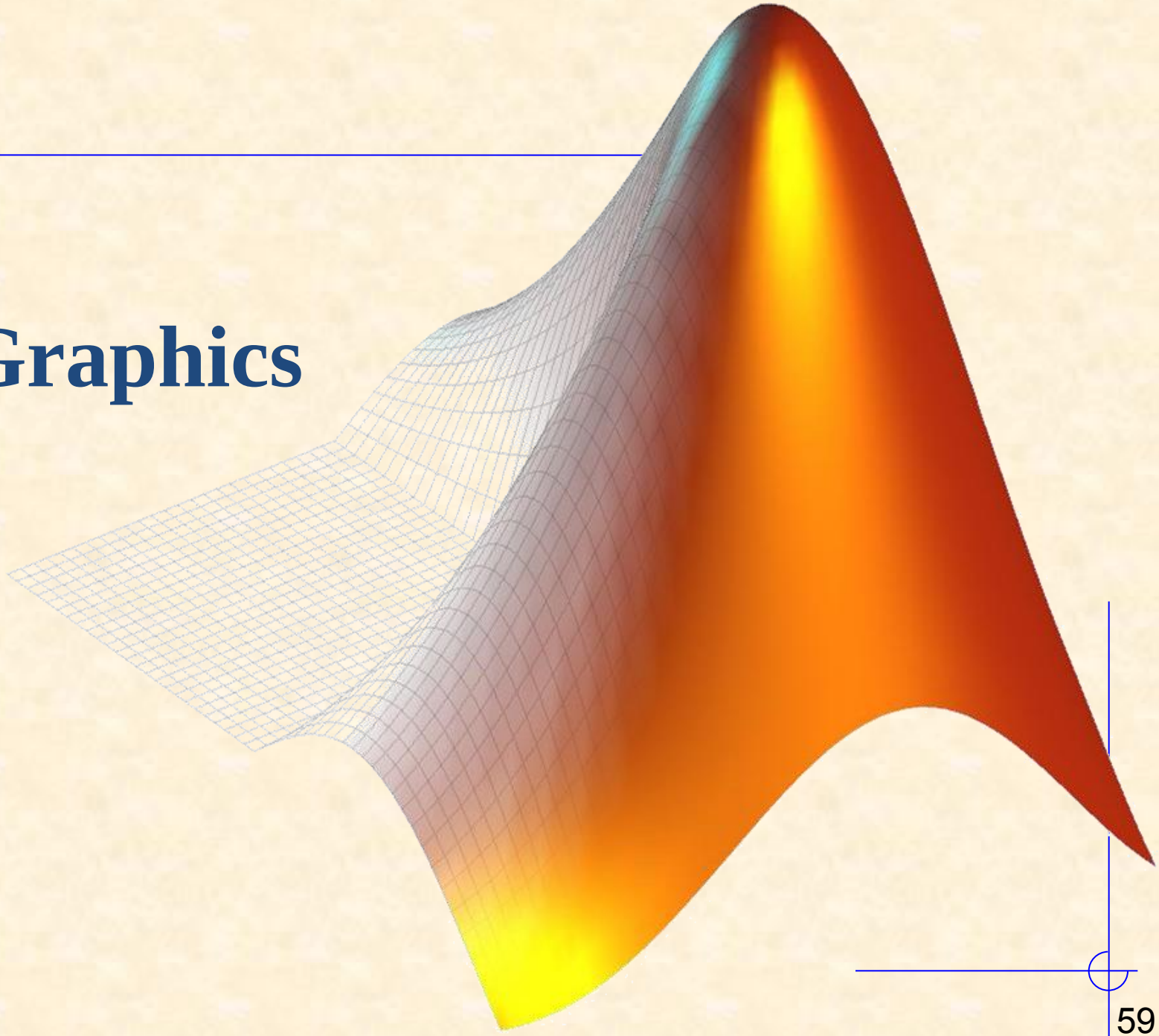
```
x=rand(40,1);
y=rand(40,1);
area=70*(x+y.^2);
color=(x.^2+y);
scatter(x,y,area,color,'filled')
box on
```



Additional plot-related commands

`xlim`	X-axis limits
`ylim`	Y-axis limits
`zlim`	Z-axis limits
`daspect`	Set and get aspect ratio
`pbaspect`	Set and get plot box aspect ratio
`text`	Place text on plot
`quiver`	Quiver or velocity plot
`fplot`	Plot a function
`pareto`	Pareto chart
`plotmatrix`	Scatter plot matrix
`ribbon`	Linear plot with 2-D lines as ribbons

3D Graphics



Line plots in 3D – `plot3`

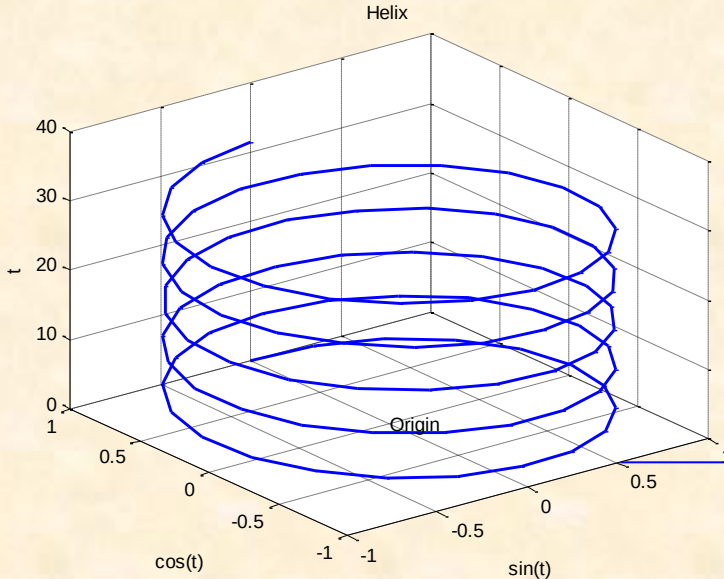
- The simplest case of 3D plotting is plotting lines, specified similarly to 2D
  - Each point on a line is specified by 3 values: x,y,z
- **`plot3`** syntax:
  - Plotting single line:

```
plot3(xdata,ydata,zdata 'color_linestyle_marker')
```

  - Plotting multiple lines:

```
plot3(x1, y1, z1,'clm1', x2, y2, z2,'clm2', ...)
```

```
t = linspace(0,10*pi);
plot3(sin(t),cos(t),t)
xlabel('sin(t)')
ylabel('cos(t)')
zlabel('t')
text(0,0,0,'Origin')
grid on
title('Figure 26.1: Helix')
```



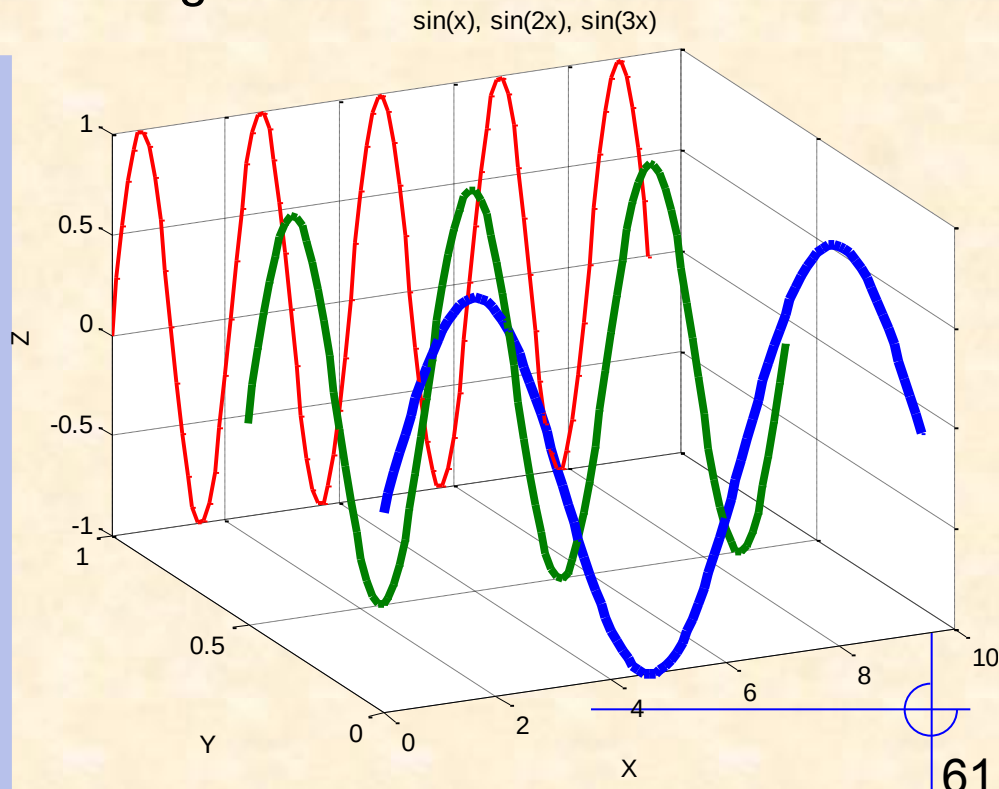
Line plots in 3D – `plot3`

All the basic features of 2D plots are present here too:

- `axis` command gives axis limits as in 2D
- `zlabel` command adds a label to z-axis
- `grid` command adds a grid to each of 3 planes
- Multiple plots can be performed using `hold`

```
x = linspace(0,3*pi); % x-axis data
z1 = sin(x);          % plot in x-z plane
z2 = sin(2*x);
z3 = sin(3*x);
y1 = zeros(size(x)); % giving each curve
y3 = ones(size(x));  % different y-axis
y2 = y3/2;           % values

plot3(x,y1,z1,x,y2,z2,x,y3,z3)
grid on
xlabel('X'), ylabel('Y'), zlabel('Z')
title(' sin(x), sin(2x), sin(3x)')
```



Scalar functions of 2 variables

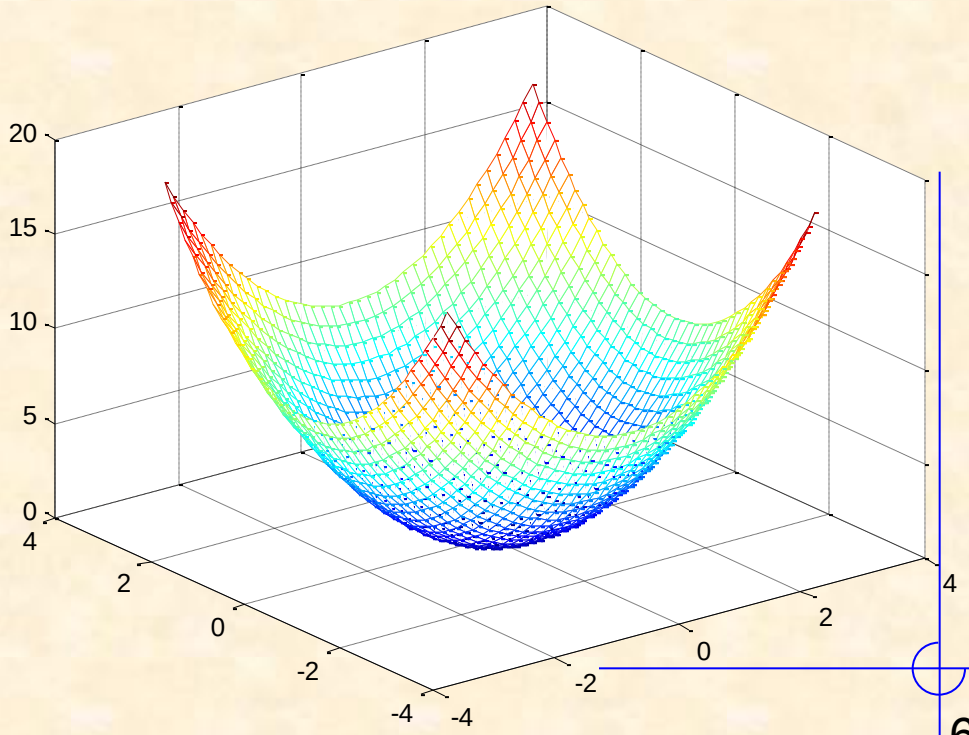
- Here we wish to visualize a function of the type: $z = (x + y)^2$
- In MATLAB, z is represented by a matrix and the variables x and y will be represented by vectors
- To compute z using array operations we need arrays of x and y values in proper orientation
  - These can be created using

`meshgrid`

For example ……..preparing:

- We can draw $z = x^2 + y^2$
- Using:

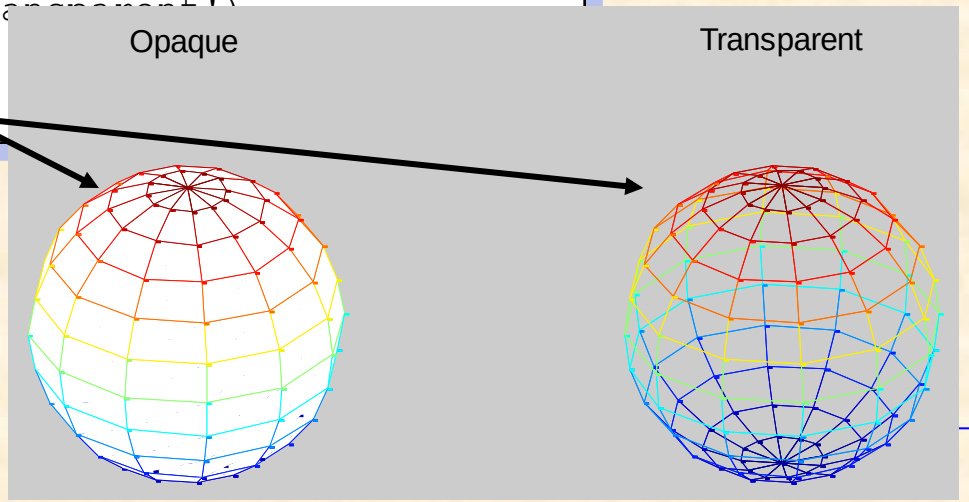
```
[X,Y]=meshgrid(-3:0.2:3)
Z=X.^2+Y.^2;
mesh(X,Y,Z)
```



Mesh plots

- By default the surface is opaque
- This can be switched on and off using:
  - `hidden on`
  - `hidden off`

```
[X,Y,Z]=sphere(12);
subplot(1,2,1)
mesh(X,Y,Z), title('Figure 26.5a: Opaque')
hidden on
axis square off
subplot(1,2,2)
mesh(X,Y,Z), title('Figure 26.5b: Transparent')
hidden off
axis square off
```



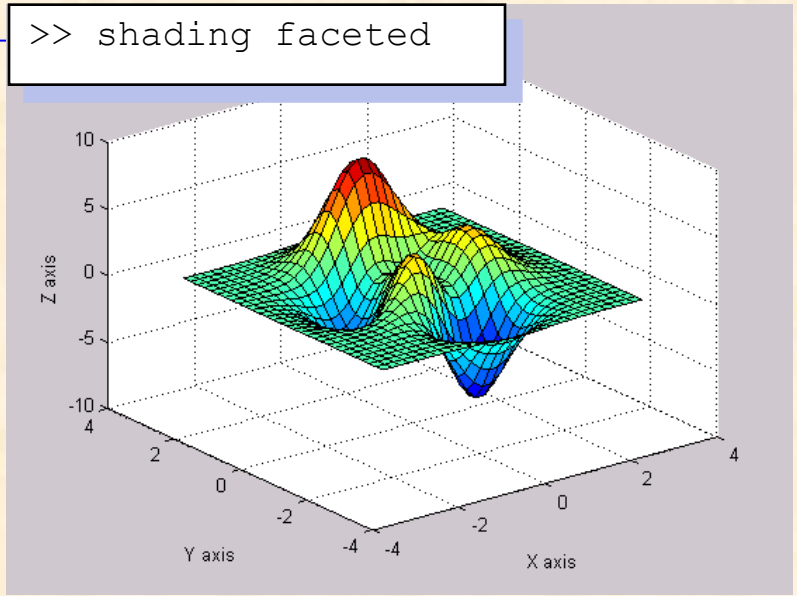
Surface plots - surf

- A **surface** plot is similar to a mesh plot – only the spaces between the lines (called *patches*) are filled in
- Three methods of shading the plot are available:
  - Faceted: the mesh appears together with a single color in each patch
  - Flat: the same as faceted, without the mesh lines
  - Interpolated: color is interpolated smoothly across patches, and mesh lines are also absent (computation intensive!)
- The plot can be switched between each scheme after the surf command has been given, using:
  - `shading faceted` [*this is the default*]
  - `shading flat`
  - `shading interp`

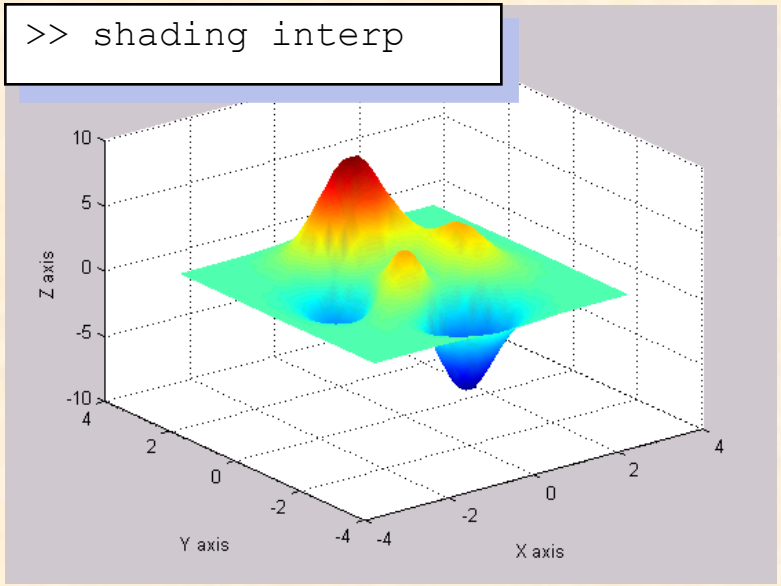
Surface plots - surf

```
[X,Y,Z]=peaks(30);
surf(X,Y,Z)
xlabel('X axis')
ylabel('Y axis')
zlabel('Z axis')
```

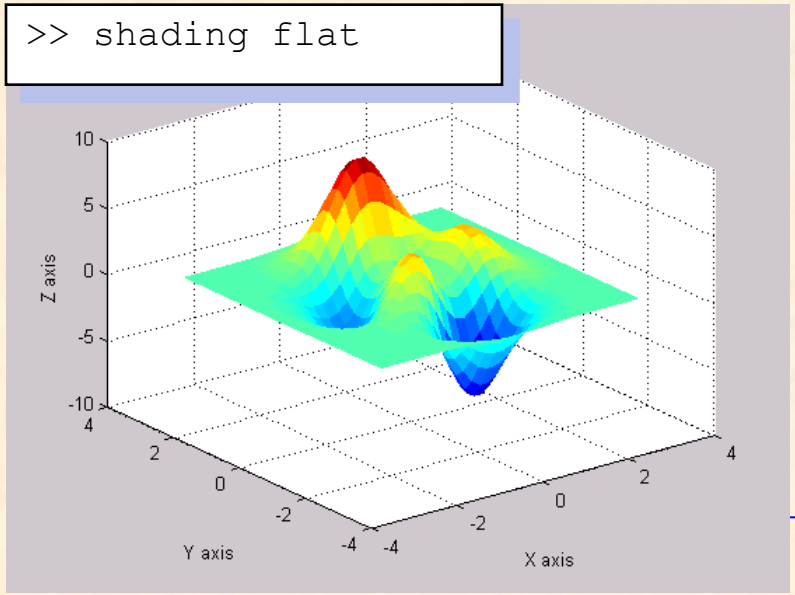
```
>> shading faceted
```




```
>> shading interp
```



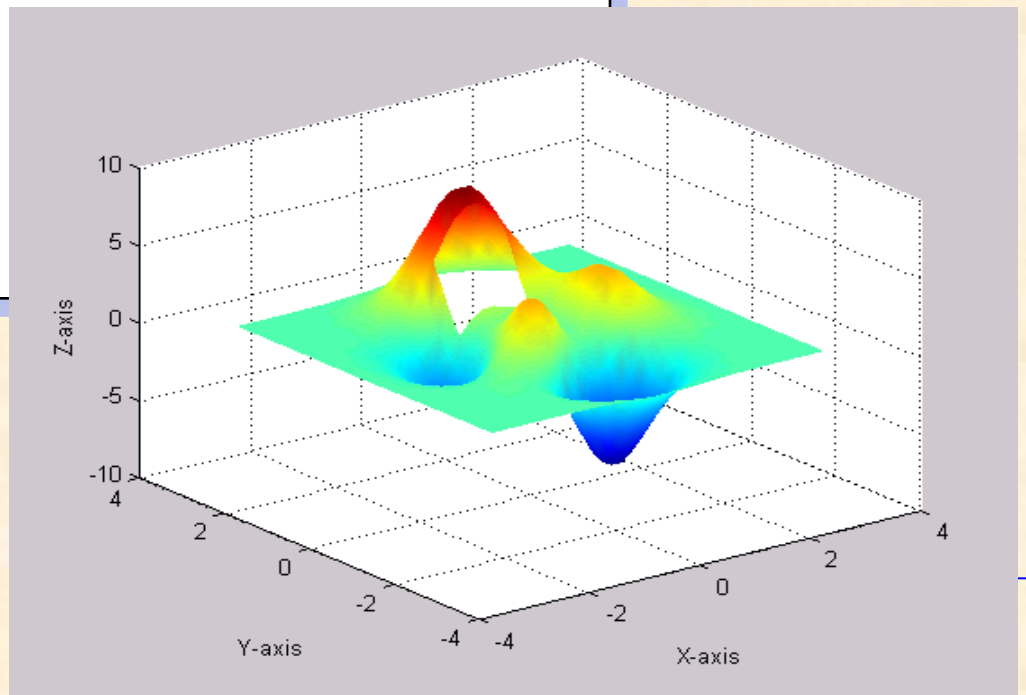

```
>> shading flat
```



Surface plots - surf

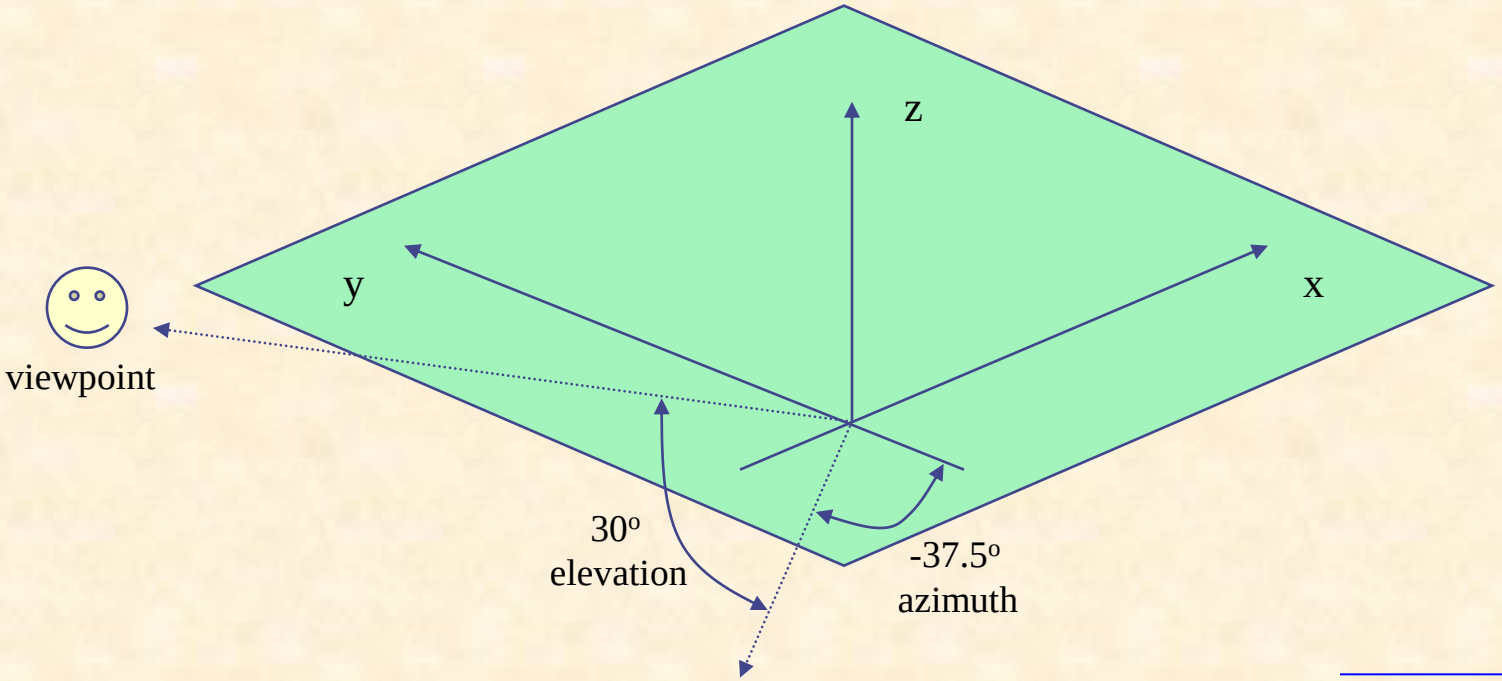
- "holes" can be inserted into the surface by replacing values in the z matrix with `NaN` values.

```
[X,Y,Z] = peaks(30);
x = X(1,:);          % vector of x axis
y = Y(:,1);          % vector of y axis
i = find(y>.8 & y<1.2);  % find y axis indices of hole
j = find(x>-.6 & x<.5);  % find x axis indices of hole
Z(i,j) = nan;        % set values at hole indices to NaNs
surf(X,Y,Z)
xlabel('X-axis')
ylabel('Y-axis')
zlabel('Z-axis')
shading interp
```



The viewpoint

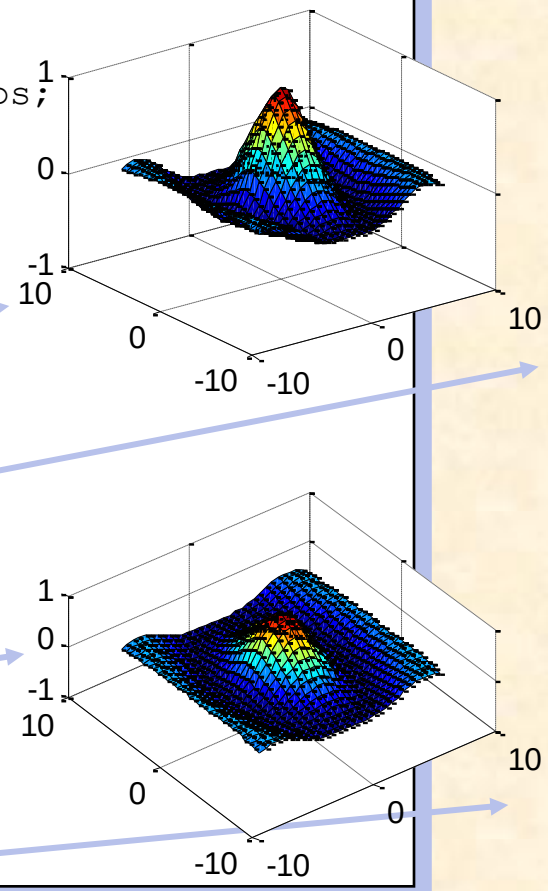
- The **default viewpoint** is:
  - Looking down at the z=0 plane at 30 degrees (**elevation**)
  - Looking at the x=0 plane at –37.5 degrees (**azimuth**)

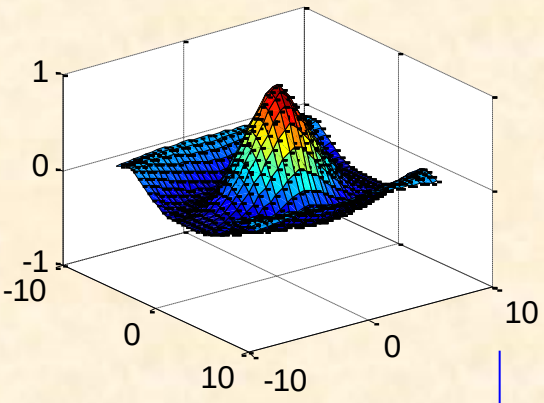


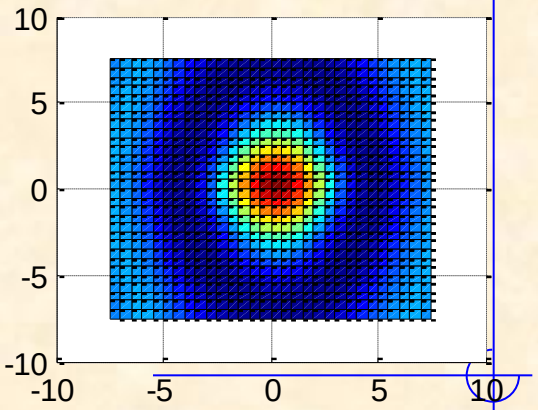
Changing the viewpoint

- The `view` function changes the viewpoint, by specifying azimuth and elevation in degrees: `view(az,el)` or

```
x = -7.5:.5:7.5; y = x;    % create a data set
[X,Y] = meshgrid(x,y);
R = sqrt(X.^2+abs(Y).^1.5)+eps;
Z = sin(R)./R;
subplot(2,2,1)
surf(X,Y,Z)
view(-37.5,30)
subplot(2,2,2)
surf(X,Y,Z)
view(-37.5+90,30)
subplot(2,2,3)
surf(X,Y,Z)
view(-37.5,60)
subplot(2,2,4)
surf(X,Y,Z)
view(0,90)
```



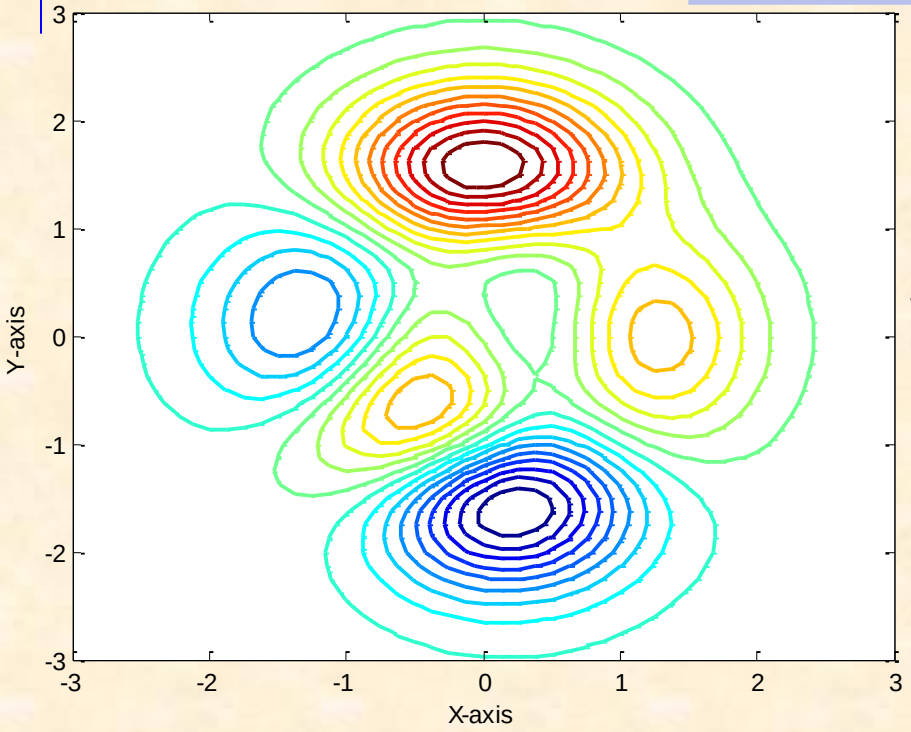


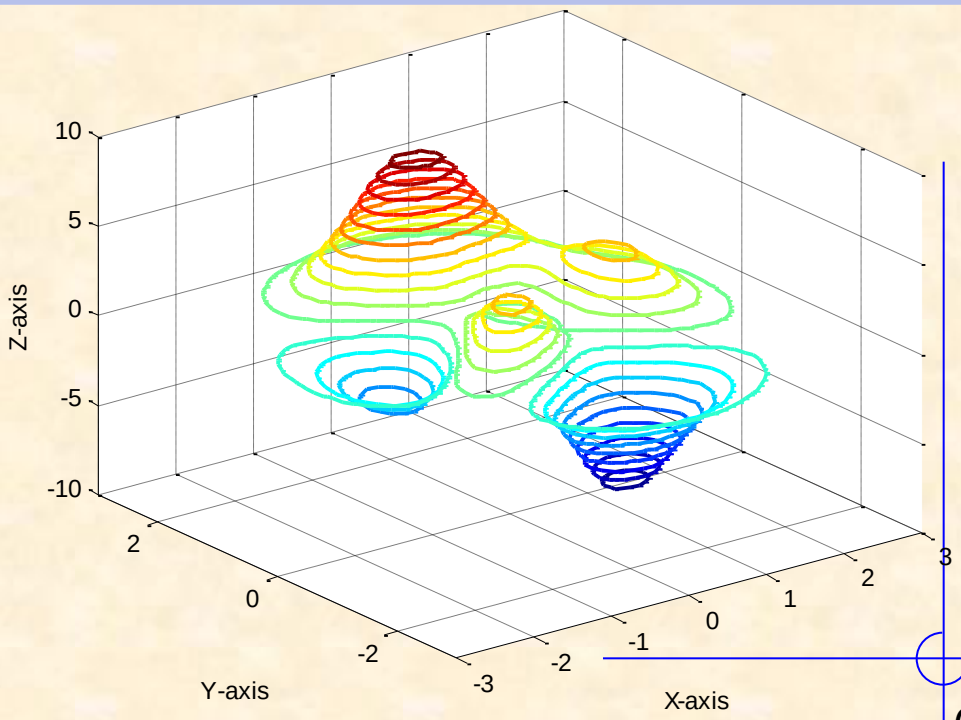


Contour plots

- contour plots show lines of constant elevation – like topographical maps
  - `contour` – in 2D
  - `contour3` – in 3D

```
[X,Y,Z] = peaks;
contour(X,Y,Z,20) % generate 20 2-D contour lines
xlabel('X-axis'), ylabel('Y-axis')
contour3(X,Y,Z,20)  % the same contour plot in 3-D
xlabel('X-axis'), ylabel('Y-axis'), zlabel('Z-axis')
```

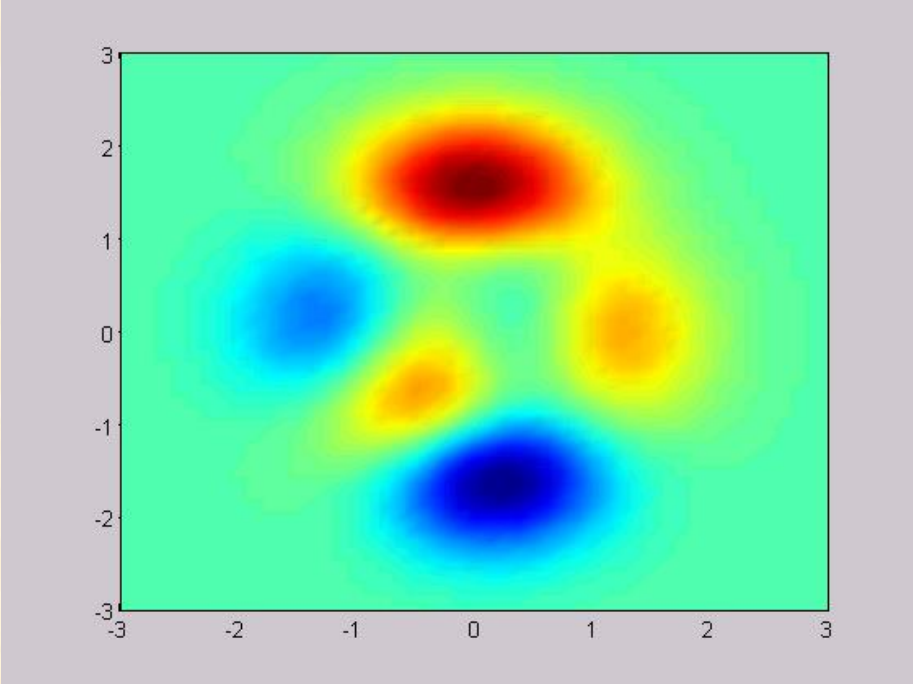




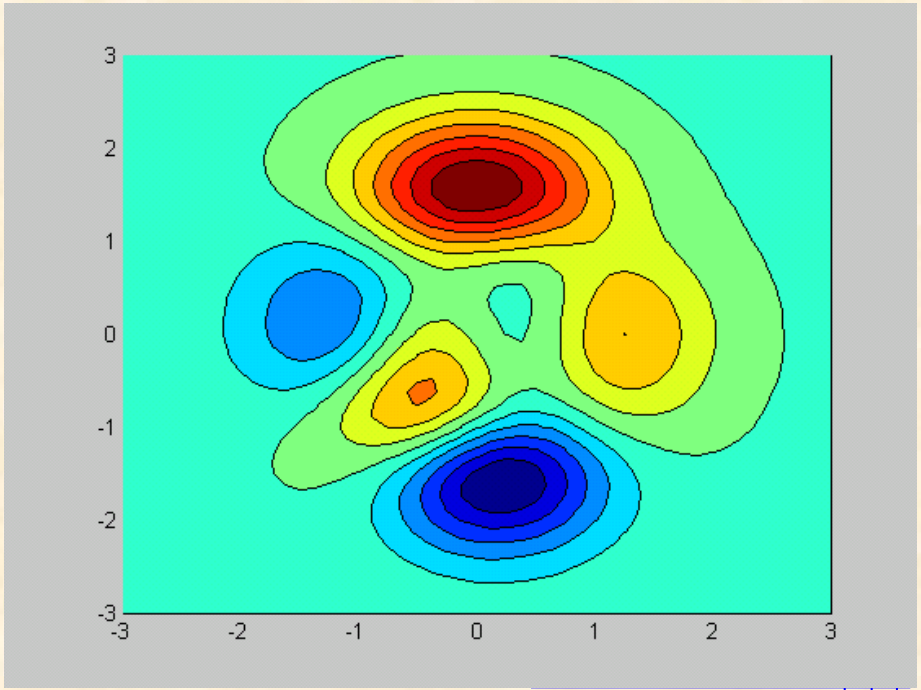
Contour plots

- `pcolor` - represents height with color instead of elevation lines
- `contourf` – creates both color and elevation lines

```
pcolor(X,Y,Z)
shading interp
```



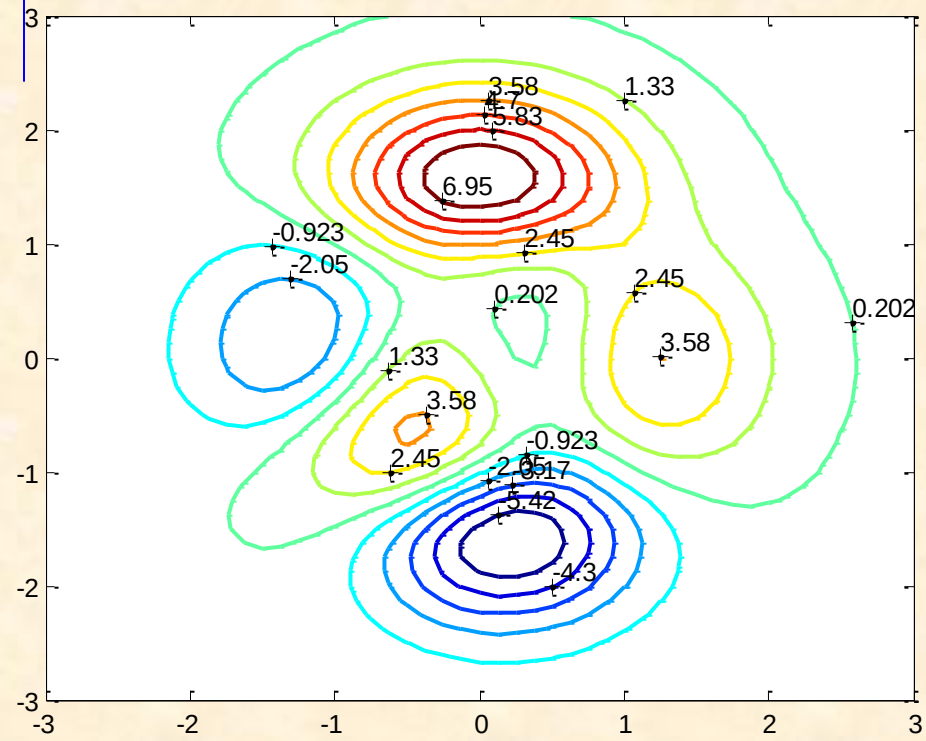

```
contourf(X,Y,Z,12)
```



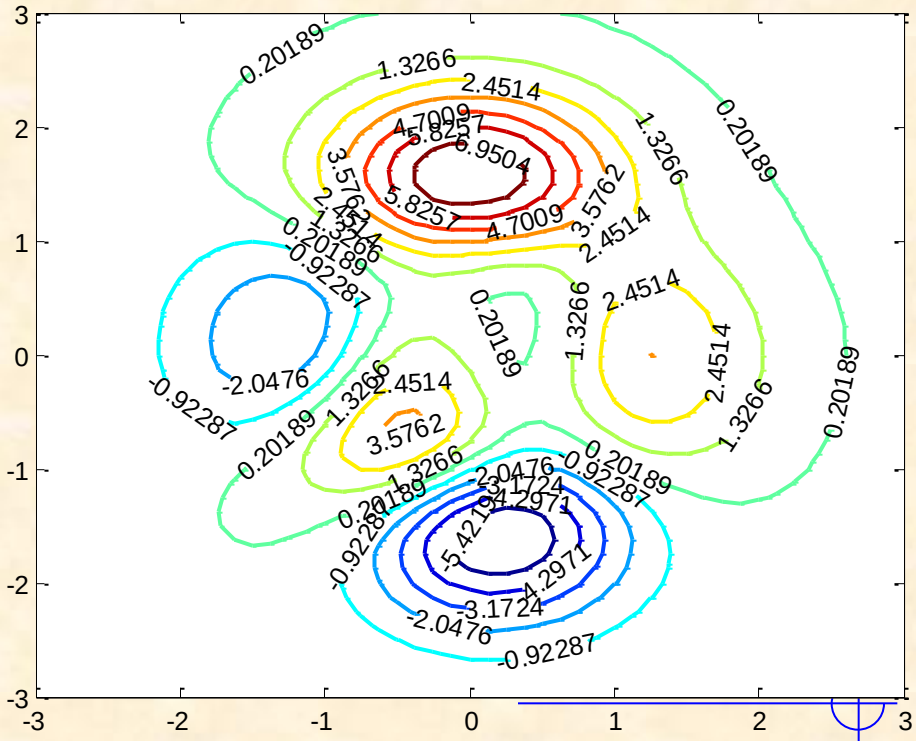
Contour plots

- Contour lines can be labeled, inline or no, using `clabel`:

```
C=contour(X,Y,Z,12);
clabel(C)
```



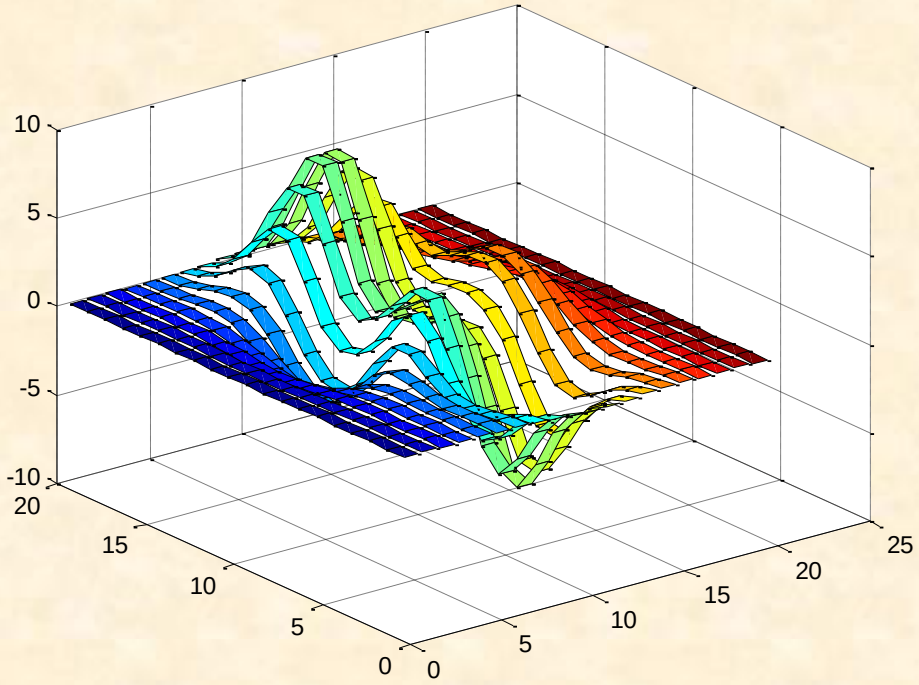

```
[C,h]=contour(X,Y,Z,12);
clabel(C,h)
```



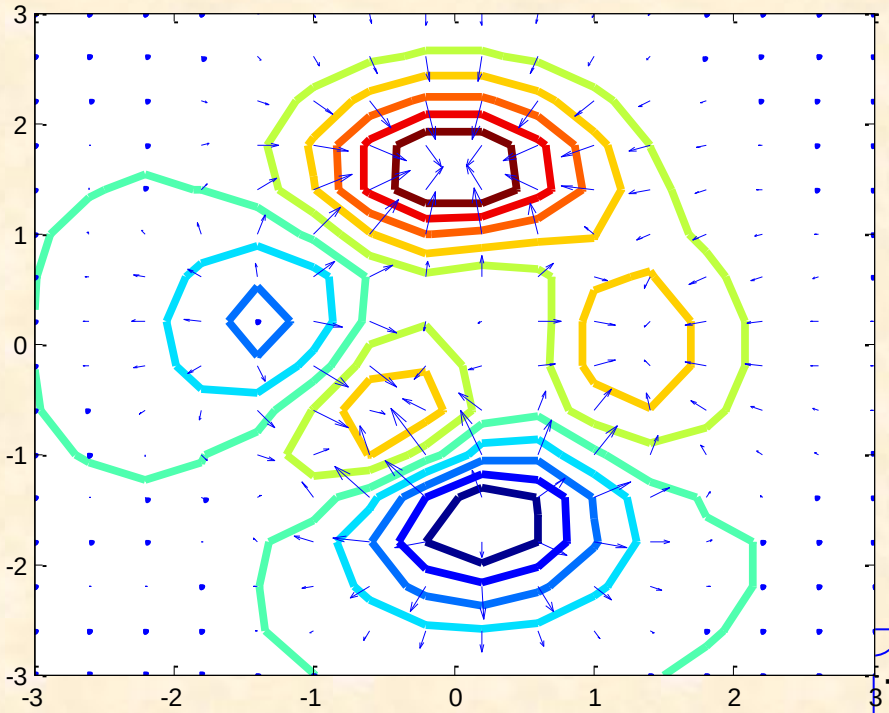
Specialized 3D plots

- `ribbon` plots columns as separate ribbons
- `quiver(x,y,dx,dy)` draws vectors (dx,dy) at points (x,y)

```
Z=peaks(20)
ribbon(Z)
```




```
[X,Y,Z] = peaks(16);
[DX,DY] = gradient(Z,.5,.5);
contour(X,Y,Z,10)
hold on
quiver(X,Y,DX,DY)
```

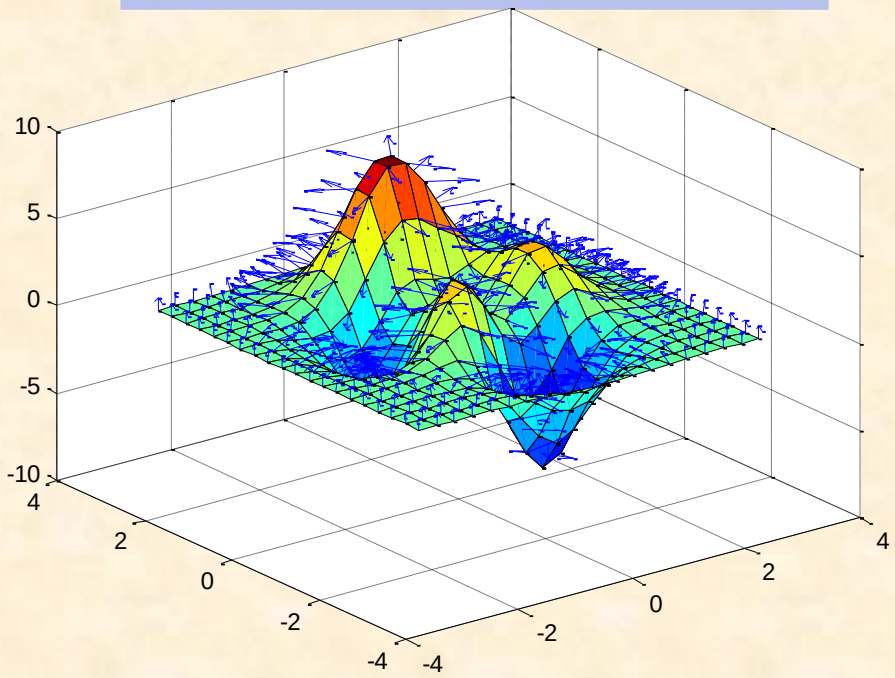


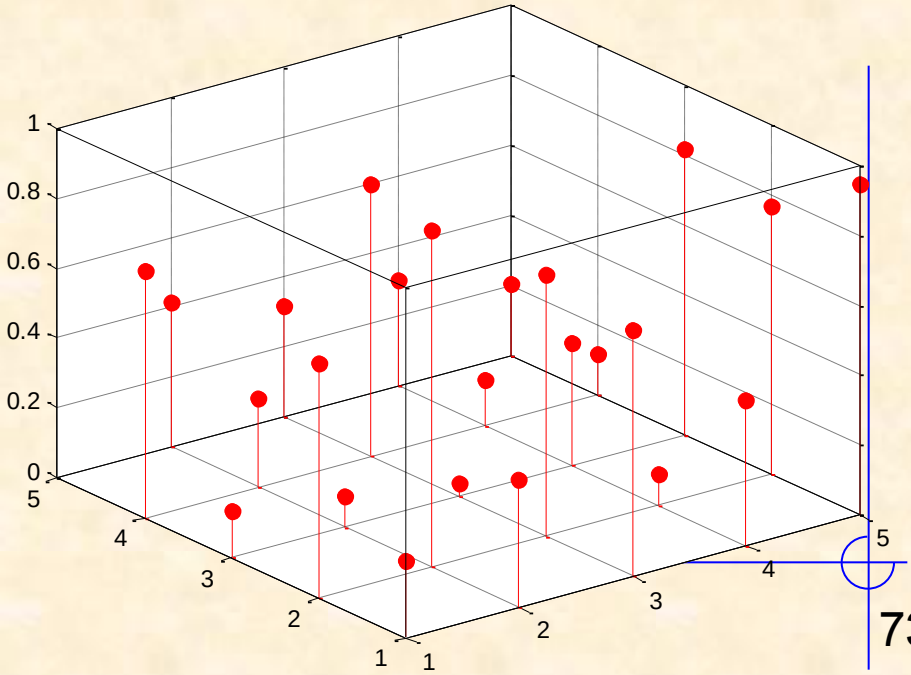
Specialized 3D plots

- `quiver3(x,y,z,Nx,Ny,Nz)` draws vectors (`Nx,Ny,Nz`) at points (x,y,z)
- `stem3` is a stem plot in 3D

```
[X,Y,Z] = peaks(20);
[Nx,Ny,Nz] = surfnorm(X,Y,Z);
surf(X,Y,Z)
hold on
quiver3(X,Y,Z,Nx,Ny,Nz)
```

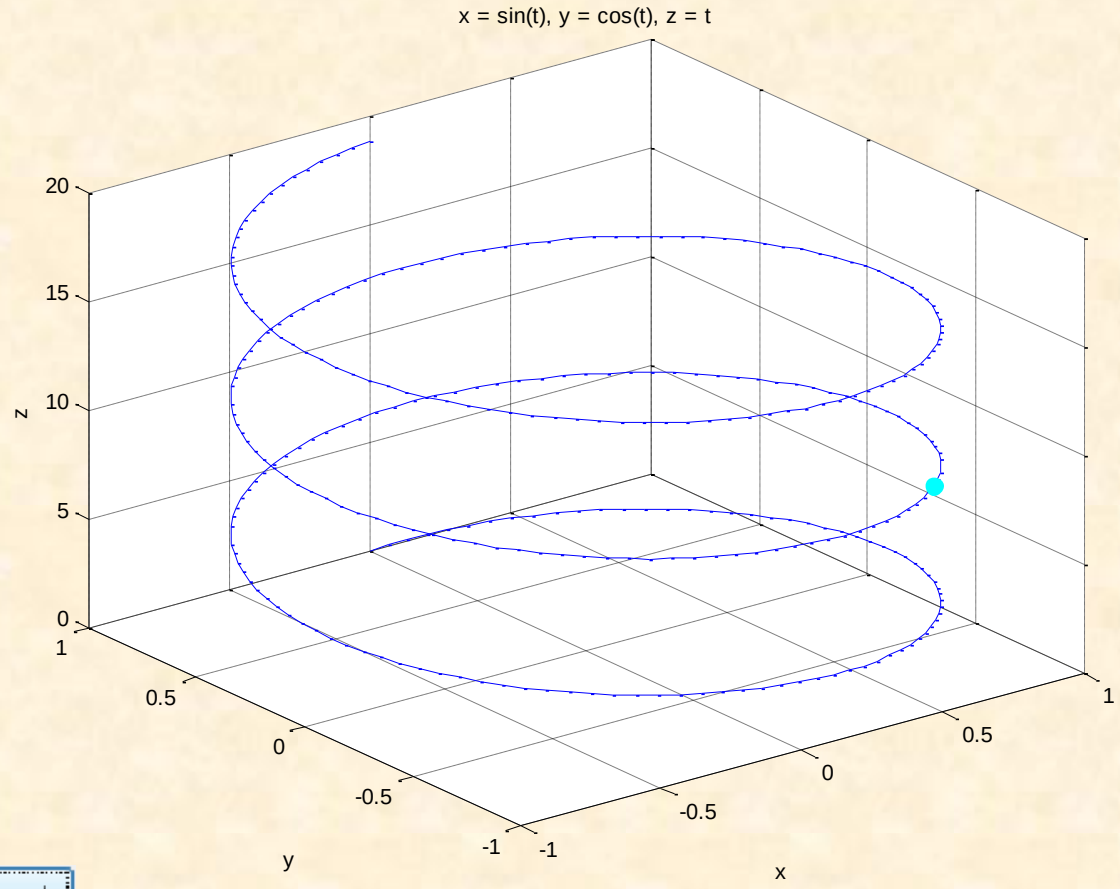
```
Z=rand(5);
stem3(Z,'ro','filled')
```





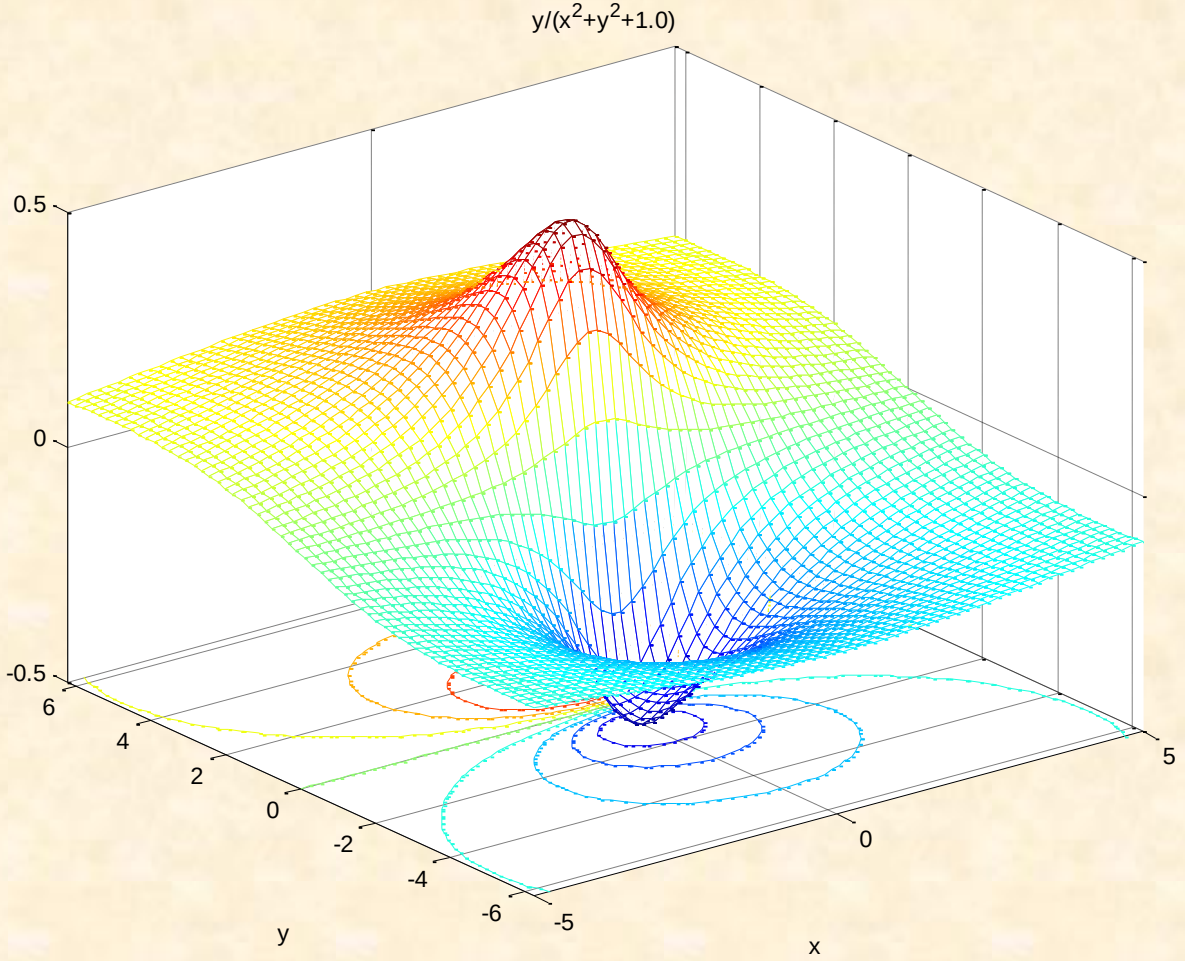
Easy plots

```
syms t;
ezplot3(sin(t), cos(t),t,[0,6*pi],'animate')
```



Easy plots

```
syms x y
ezmeshc(y/(1 + x^2 + y^2),[-5,5,-2*pi,2*pi])
```



Easy plots

```
syms x y
ezsurf(real(atan(x+i*y)))
```