# 3D Graphs in MatLab

## The plot3 function

- The `plot` function in the previous lecture extends into 3D via `plot3`. All the basic features of 2D plotting extend naturally into 3D. Pairs of arguments, for example coordinates $(x, y)$, now become triplets of arguments, $(x, y, z)$.

- The `axis` extends into 3D by adding the $z$ axis limits:

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

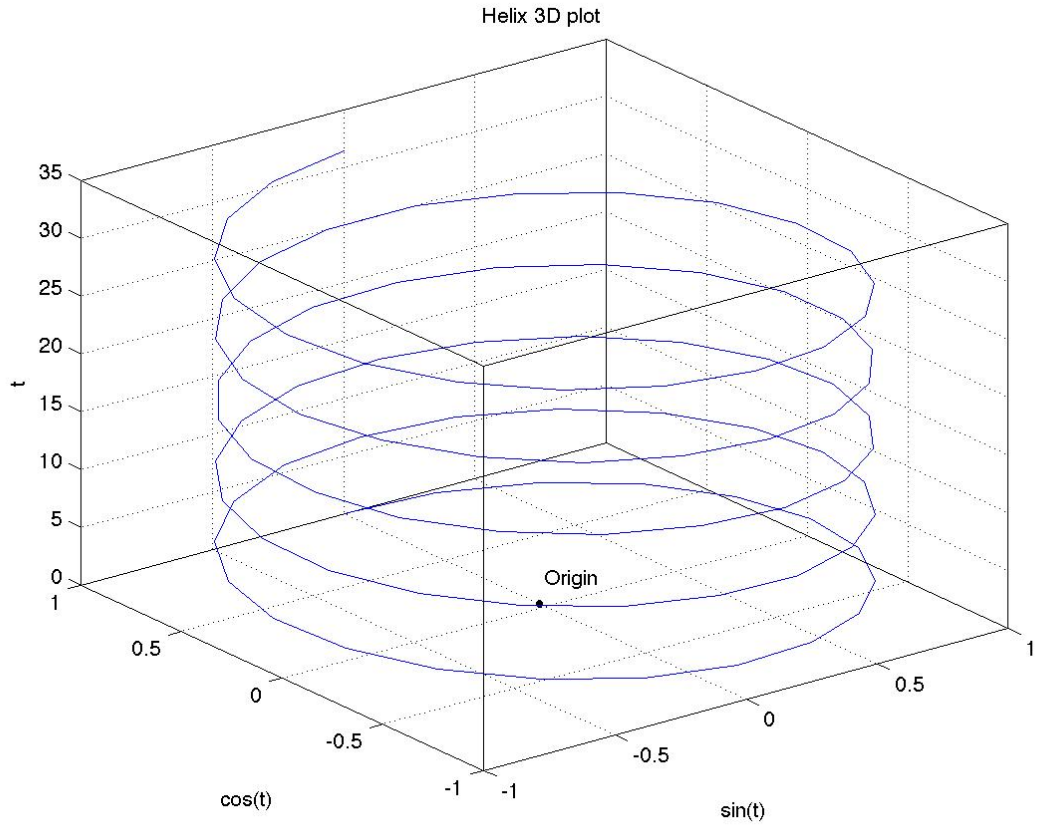
- There is now a `zlabel` for labelling the $z$ axis.

- The `grid` command toggles a 3D grid being on or off (`grid off` is the default for `plot3`).
- The `box` command creates a 3D box around the plot (`box off` is the default for `plot3`).
- A character string can be printed in 3D using `text(x,y,z,'string')`
- Consider MatLab code (**L08plot3A.m**) to plot a 3D function as a single variable:

```
close all
t=linspace(0,10*pi); % generate 100 values
plot3(sin(t),cos(t),t)
xlabel('sin(t)');
```

```
ylabel('cos(t)');
zlabel('t');
% Text at (0,0,0) uses those coordinates
% as the low left coordinates of its
% bounding box: adjust x and y to make
% the bullet centered exactly at (0,0,0)
text(-0.05,0.1,0.0,'\fontsize{36} \bullet');
text(0.0,0.0,2.5,'Origin');
grid on
box on
title('Helix 3D plot');
print L08plot3A.jpg -djpeg
```

It produces the 3D plot:




3D plot of a Helix with the origin labelled.

# Plotting 2D Scalar Arrays Using Meshgrid and Mesh

- Suppose you have a function $z = f(x, y)$ and you wish to plot $z$ as a function of $(x, y)$. An image is a perfect example of such a function.
- The plot of $z$ as a function of $x$ and $y$ is a surface in 3D.
- The values of $z$ must to stored in a 2D array.
- We must store all the $x$ and $y$ values so that they correspond to the correct $z$ values. We do this by creating matrices of the $x$ and $y$ values in the correct orientation using MatLab function `meshgrid`.

- Consider the MatLab code:

```
x=-3:3;
y=1:5;
% No semicolon, so X and Y print
[X,Y]=meshgrid(x,y)
```

produces the output:

```
X= -3  -2  -1  0  1  2  3
   -3  -2  -1  0  1  2  3
   -3  -2  -1  0  1  2  3
   -3  -2  -1  0  1  2  3
   -3  -2  -1  0  1  2  3
```

```
Y= 1   1   1   1   1   1   1
   2   2   2   2   2   2   2
   3   3   3   3   3   3   3
   4   4   4   4   4   4   4
   5   5   5   5   5   5   5
```

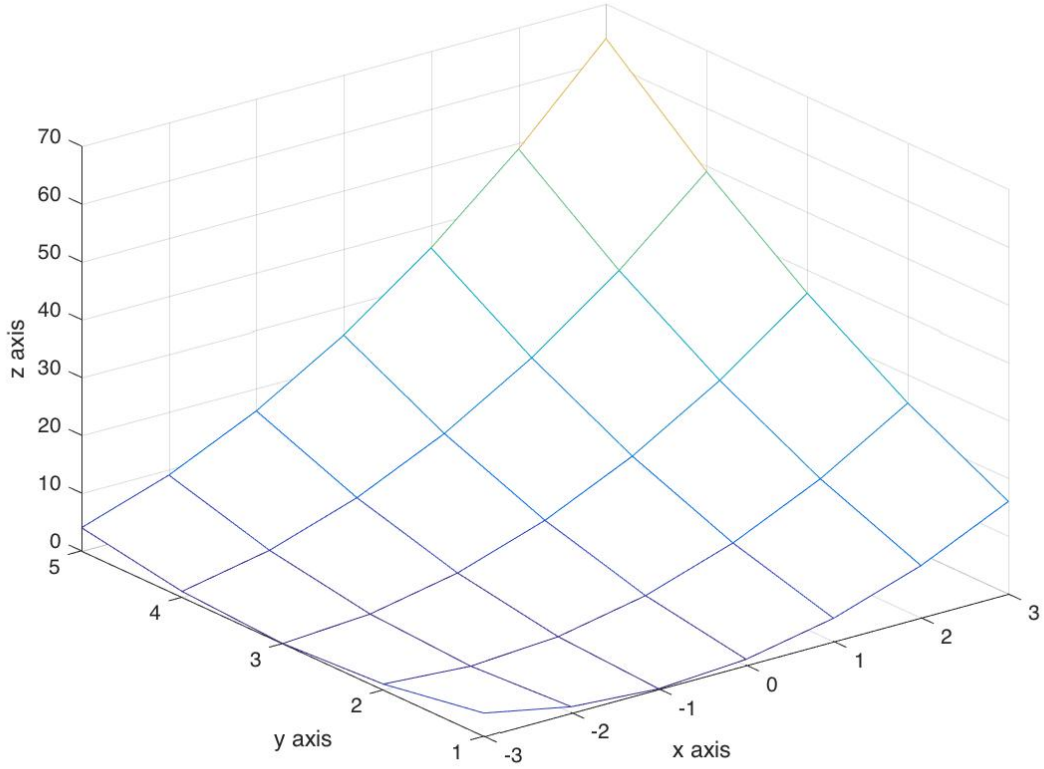
- `meshgrid` duplicates $x$ for each of the 5 rows of $y$ and duplicated $y$ for the 7 columns of $x$.
- Note that the $x$ axis varies from left to right and the $y$ axis varies from top to bottom.
- Given `X` and `Y`, if $z = f(x, y) = (x+y)^2$ we can plot it using the MatLab code (**L08plot3D.m**):

```
% No semicolon so Z prints
Z=(X+Y).^2
mesh(X,Y,Z)
xlabel('x axis');
ylabel('y axis');
zlabel('z axis');
print L08plot3D.jpg -djpeg
```

with `Z` computed as:

```
Z=  4     1     0     1     4     9    16
    1     0     1     4     9    16    25
    0     1     4     9    16    25    36
    1     4     9    16    25    36    49
    4     9    16    25    36    49    64
```

This produces the following plot:




3D surface plot of Z against X and Y.

# The Peaks Surface

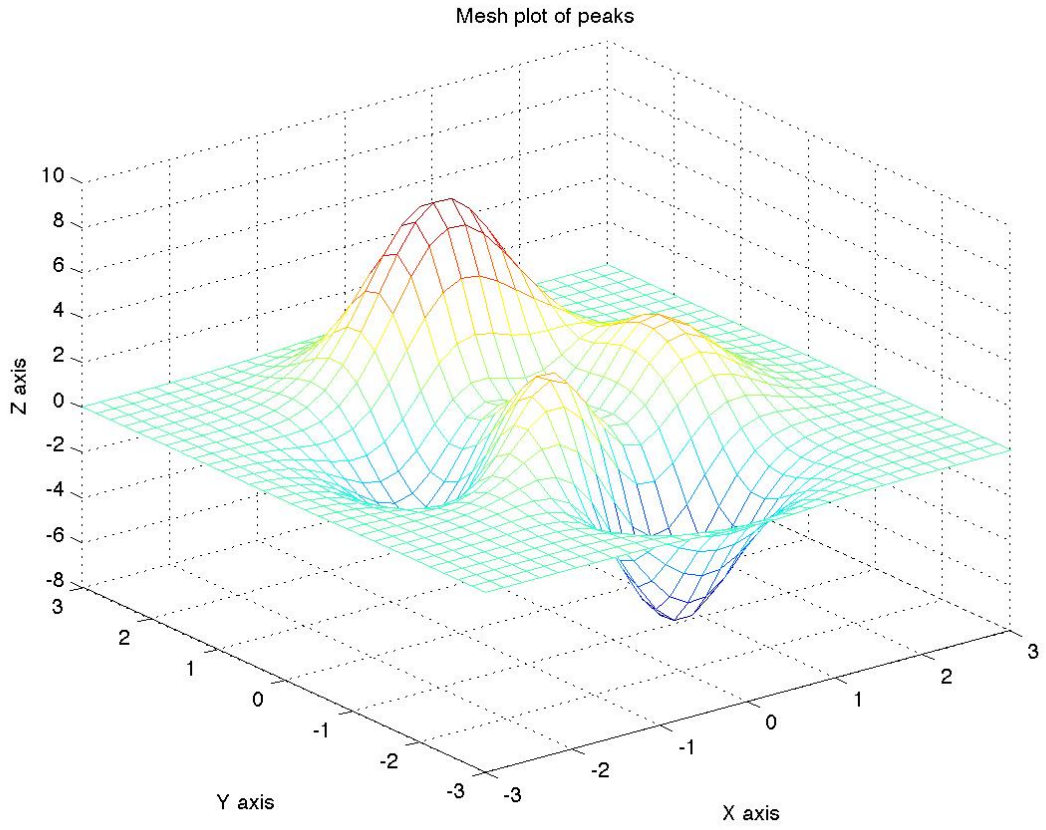
- The `peaks` MatLab demo function is an "interesting" function of two variables, obtained by translating and scaling Gaussian distributions and is used commonly for demonstrating a 3D surface rendering in MatLab. Its formula is given as:

$$\begin{aligned} peaks(x,y) &= 3(1-x)^2 e^{(-(x^2)+(y+1)^2)} \\ &- 10(x/5 - x^3 - y^5)e^{(-x^2-y^2)} \\ &- 1/3e^{(-(x+1)^2-y^2)}. \end{aligned}$$

- The following MatLab code (**L08plot3E.m**):

```
% 30 by 30 array of surface values
[X,Y,Z]=peaks(30);
mesh(X,Y,Z)
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Mesh plot of peaks');
print L08plot3E.jpg -djpeg
```

produces:




3D mesh plot of peaks.

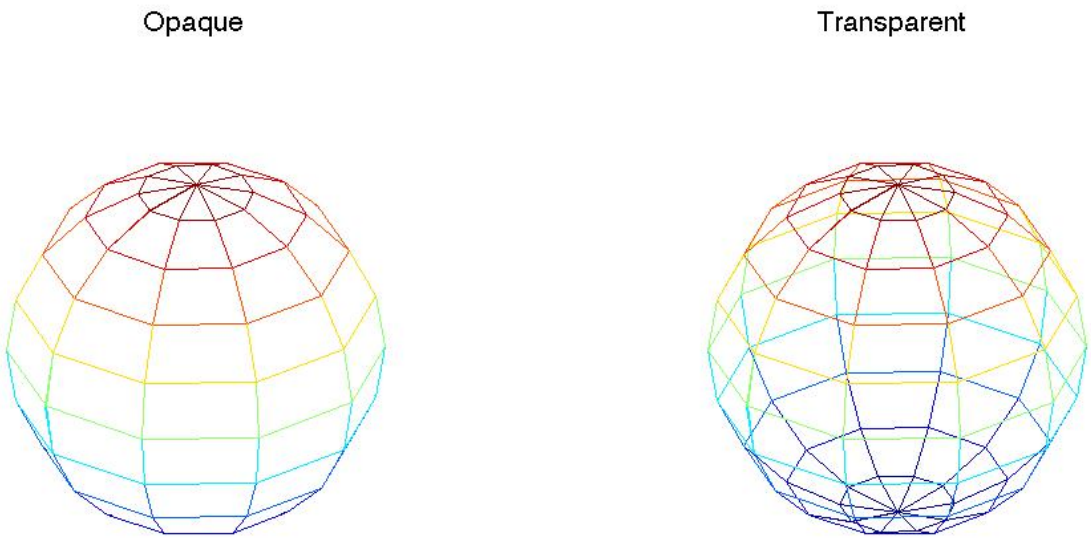
# Opaque and Transparent Spheres

- The areas between the mesh lines are opaque rather than transparent.
- MatLab provides a **tessellated** sphere function that draws a sphere with user given number of vertical and horizontal line (defining either triangles or quadlaterals). Consider the following code (**L08plot3F.m**):

```
[X,Y,Z]=sphere(10,8);
subplot(1,2,1)
mesh(X,Y,Z)
title('Opaque');
hidden on
axis square
```

```
axis off
subplot(1,2,2)
mesh(X,Y,Z)
title('Transparent');
hidden off
axis square
axis off
print L08plot3F.jpg -djpeg
```

The following plot is printed:




3D tessellated opaque and translated spheres

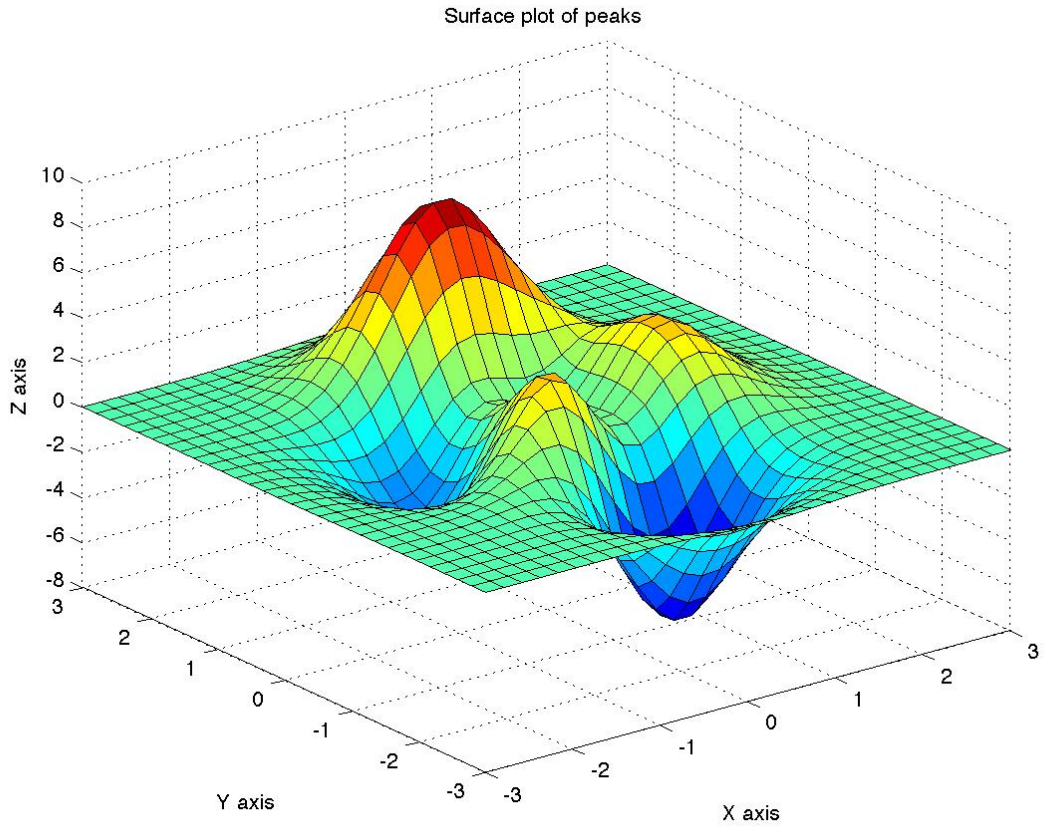
# Surface Plots with Shading

- A **surface** plot looks like a mesh plot, except the space between the lines, called **patches**, are filled in.
- A surface plot is the **dual** of a mesh plot. In a surface plot lines are black and the patches have colour, whereas for meshes, the patches are the colour of the axes and the lines have colour.
- The default surface **shading** is **faceted** shading, where the patches are like stained glass outlined by black lines.
- The following MatLab code (**L08plot3G.m**):

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

```
surf(X,Y,Z)
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Surface plot of peaks');
print L08plot3G.jpg -djpeg
```

produces:




3D surface plot of peaks with facet shading

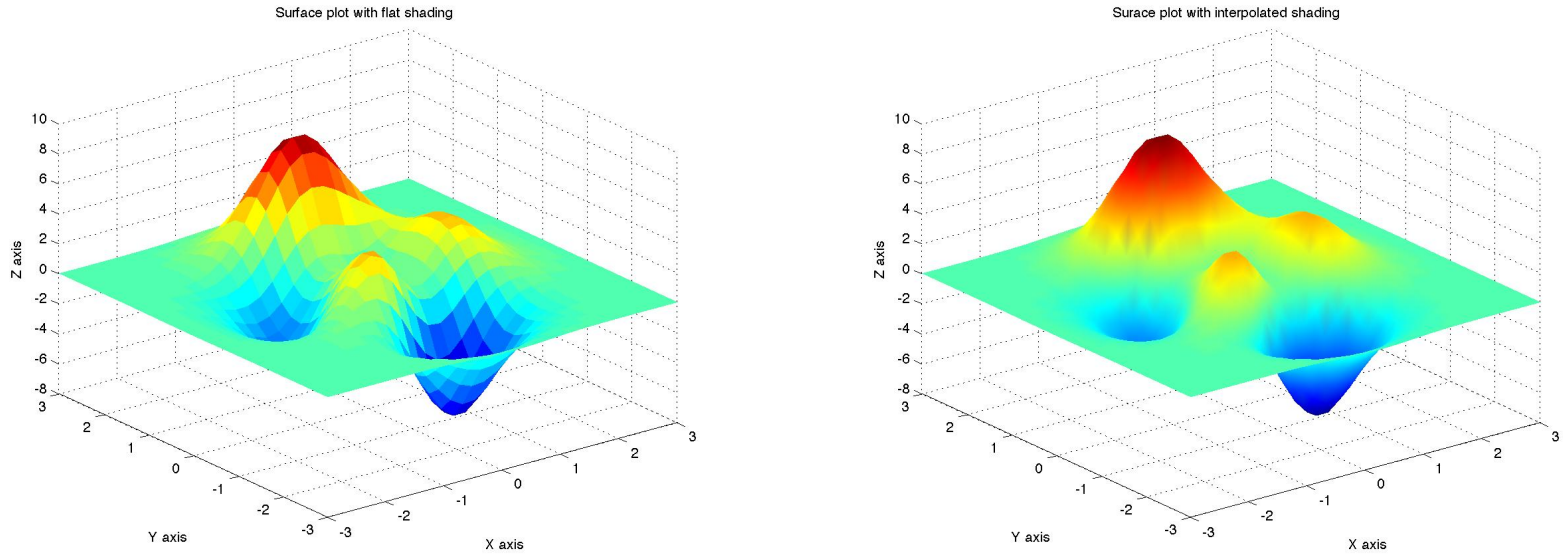
- MatLab also provides for **flat** shading and **interpolated** shading.
- With flat shading, the black lines are removed and each patch has constant colour.
- With interpolated shading, the black lines are also removed but each the colour of each patch is interpolated over its area based on the colour of its vertices. The colours of the surface blend smoothly over all the patches. This may be visually pleasing but is computationally more expensive.
- Note that shading works with meshes as well, but since only lines are coloured the visual effect is minimal.
- Consider the following MatLab code (**L08plot3H.m**):

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

```
surf(X,Y,Z)
shading flat
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Surface plot with flat shading');
print L08plot3H1.jpg -djpeg
% wait 5 seconds
pause(5)
shading interp
title('Surace plot with interpolated shading');
print L08plot3H2.jpg -djpeg
```

- Note that the surfaces if plotted with flat shading first, saved as plot3H1.jpg, then the shading is changed to interpolated and re-saved as plot3H2.jpg.

- These 2 plots are shown below:




3D surface plots of peaks with flat and interpolated shading

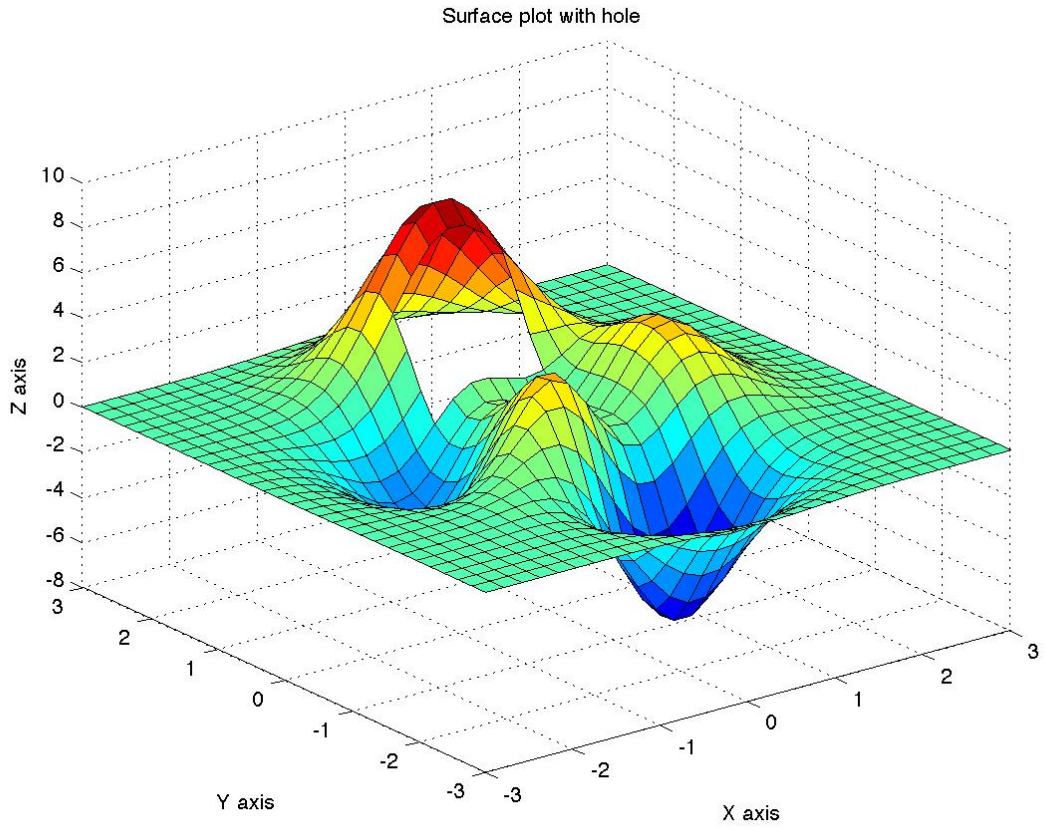
# Surfaces with Cutouts

- Sometimes it may be necessary to remove a part of a surface so that other underlying parts can be seen. We do this by setting the desired surface holes with `NaN` (Not a Number). MatLab plotting functions ignore `NaN` values, leaving a hole where they appear. Consider the following MatLab code (**L08plot3I.m**):

```
[X,Y,Z]=peaks(30);
x=X(1,:); % take one column vector X as x axis
y=Y(:,1); % take one row vector of Y as y axis
% The hole is for y between 0.8 and 1.2
% and x between -0.6 and 0.5
```

```
% We find the coordinates satisfying these conditions
i=find(y>0.8 & y<1.2);
j=find(x>-0.6 & x<0.5);
Z(i,j)=nan;
surf(X,Y,Z)
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Surface plot with hole');
print plot3I.jpg -djpeg
```

produces the plot:



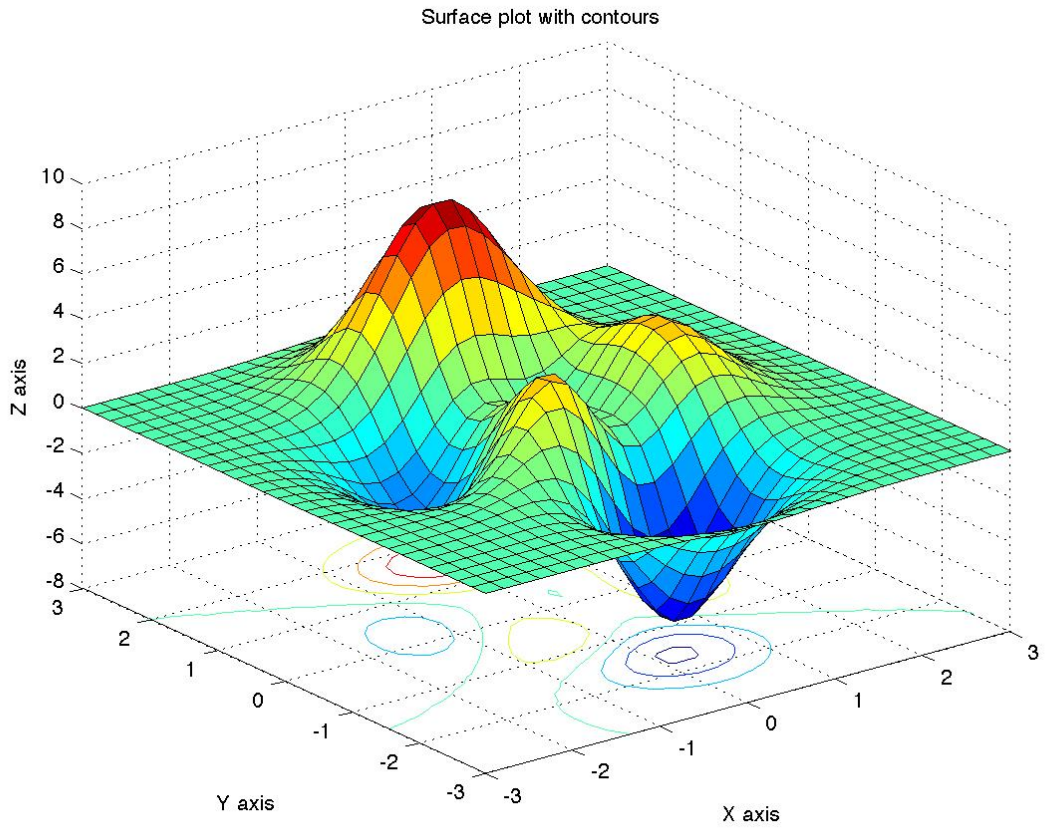

3D surface plot with hole

# Surface Plots with Contour

- `surfc` is a surface plot with underlying contour plot.
- The MatLab code (**L08plot3J.m**):

```
[X,Y,Z]=peaks(30);
% surf plot with contour
surfc(X,Y,Z)
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Surface plot with contours');
print L08plot3J.jpg -djpeg
```

produces the following plot:




3D surface plot with hole

# Surface Plots with Lighting

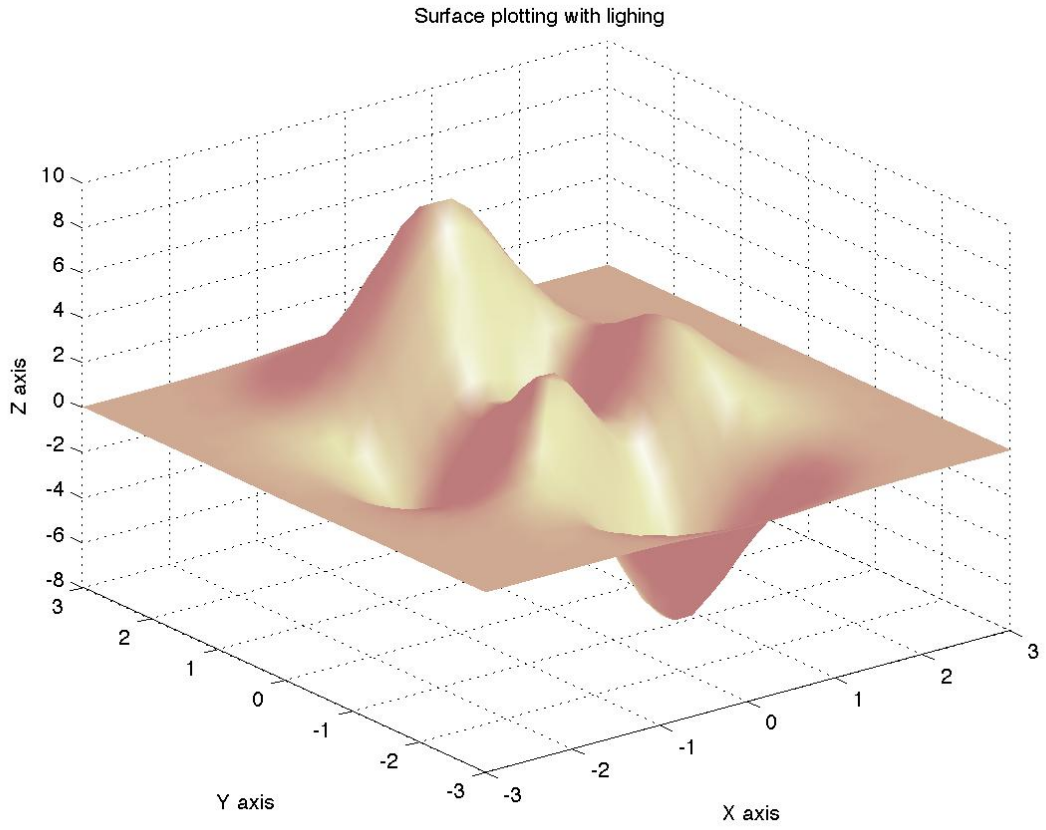
- `surfl` produces a surface plot with lighting. `surfl` modifies the colour of the surface to give the appearance. We can use a MatLab `colormap` to change the different set of colours to a figure. A more detail of `colormap` is delayed to later.

- The following MatLab code (**L08plot3K.m**):

```
[X,Y,Z]=peaks(30)
surfl(X,Y,Z);
% surfl plots look best with interp shading
shading interp
% they look better with shades of a single color
```

```
colormap pink
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Surface plotting with lighing');
print L08plot3K.jpg -djpeg
```

produces the 3D plot:




3D surface plot with lighting

# Surface Plots with Surface Normals

- The `surfnorm(X,Y,Z)` function compute surface normals for the surface at the data points defined at the `X`, `Y` and `Z` values. Both surface and surface normals are plotted.

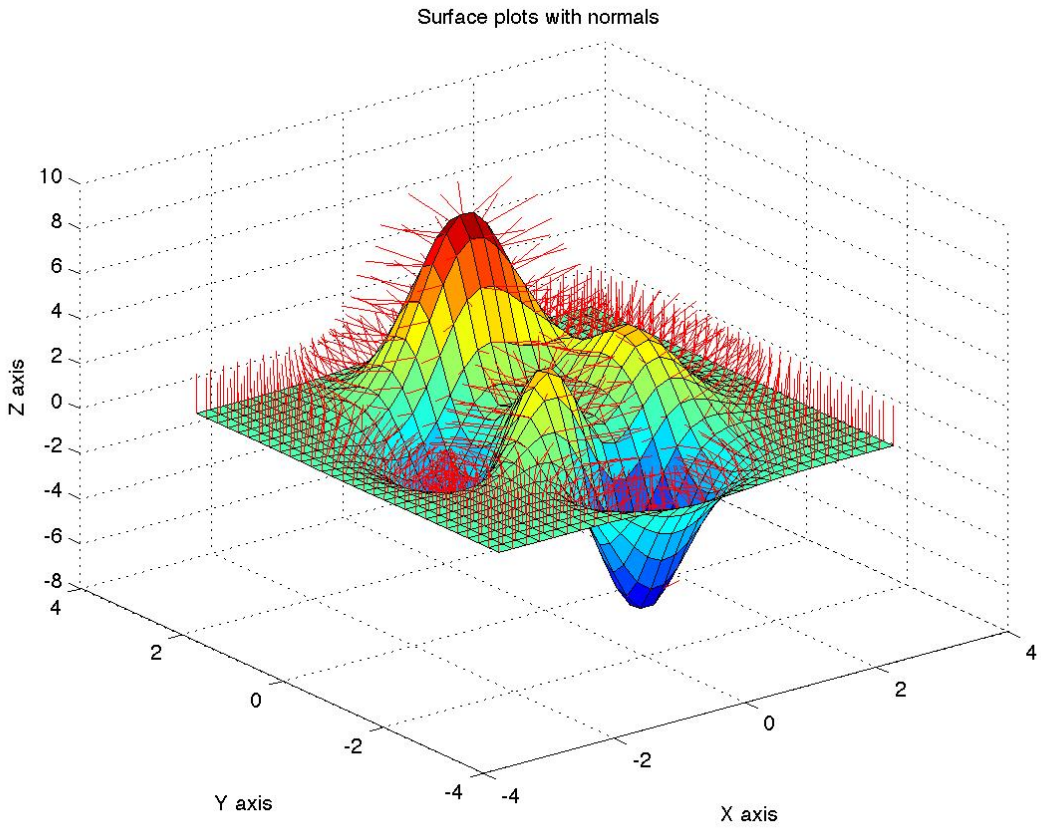
- Note that the surface normals are unnormalized, i.e. $||\vec{n}||_2 \neq 1$ for some normal $\vec{n}$.

- The following MatLab code (**L08plot3L.m**):

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

```
ylabel('Y axis');
zlabel('Z axis');
title('Surface plots with normals')}
print L08plot3L.jpg -djpeg
```

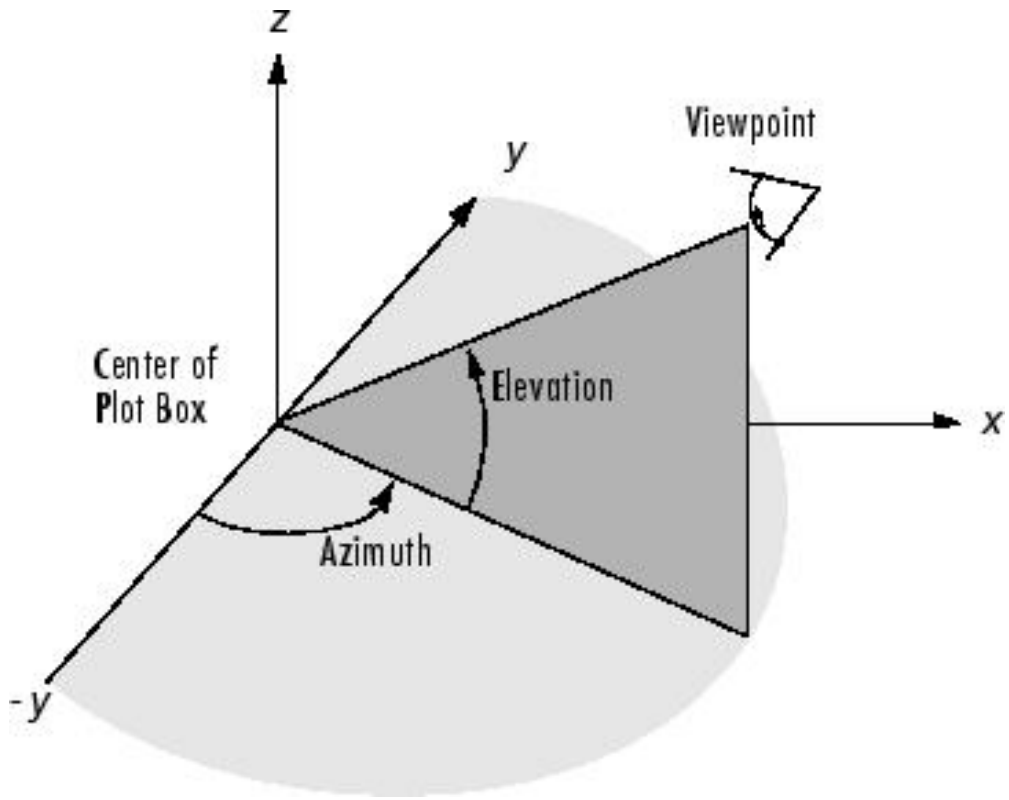
produces the 3D plot:




3D surface plot with surface normals

# Changing Viewpoints

- The default viewpoint of 3D plots is looking down at the $z = 0$ plane at an angle of $30^\circ$ and looking up at the $y = 0$ plane at angle of $-37.5^\circ$
- The angle of orientation with respect to the $z = 0$ plane is called the **elevation** angle and the angle with respect to the $y = 0$ plane is called the **'azimuth** angle. The concepts of the azimuth and elevation angles are described in the following figure.
- For 2-D plots, the default is azimuth = $0^\circ$ and elevation = $90^\circ$. For 3-D plots, the default is azimuth = $-37.5^\circ$ and elevation = $30^\circ$.


z
y
Viewpoint
Center of
Plot Box
Elevation
x
Azimuth
-y

# The Elevation and Azimuth Angles

- The follow MatLab code (**L08plot3M.m**) demonstrates the effect of changing these 2 angles:

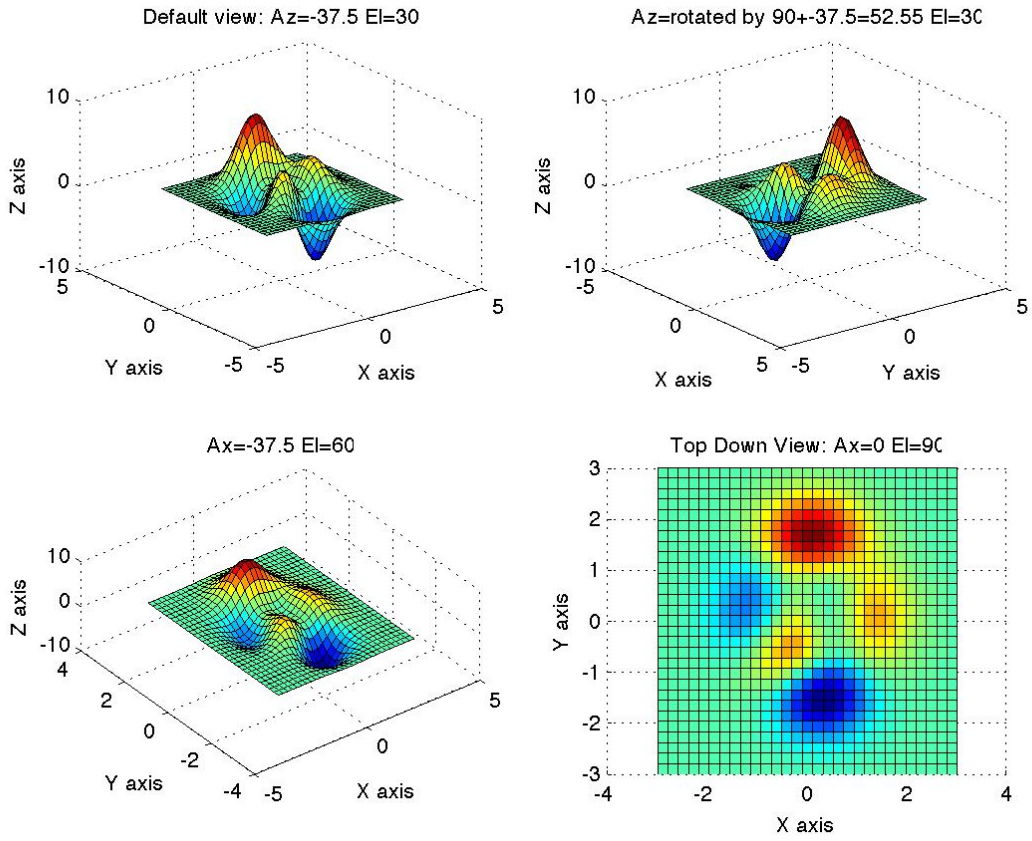
```
% Generate general peaks plot
[X,Y,Z]=peaks(30);
subplot(2,2,1);
surf(X,Y,Z);
% Use default view angles
view(-37.5,30);
xlabel('X axis');
ylabel('Y axis');
```

```
zlabel('Z axis');
title('Default view: Az=-37.5 El=30');


subplot(2,2,2);
surf(X,Y,Z);
% Add 90 to the azimuth angle
view(-37.5+90,30);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Az=rotated by 90+-37.5=52.55 El=30');
```

```
subplot(2,2,3);
surf(X,Y,Z);
% Add 30 to the elevation angle
view(-37.5,60);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Ax=-37.5 El=60');

subplot(2,2,4);
surf(X,Y,Z);
% Set the azimuth angle to 0 and
```

```
% the elevation angle to 90: this
% is a top down view
view(0,90);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Top Down View: Ax=0 El=90');

% plot the 4 graphs as a jpg file
print L08plot3M.jpg -djpeg
```

produces the following plot:




The effect of various elevation and azimuth angles

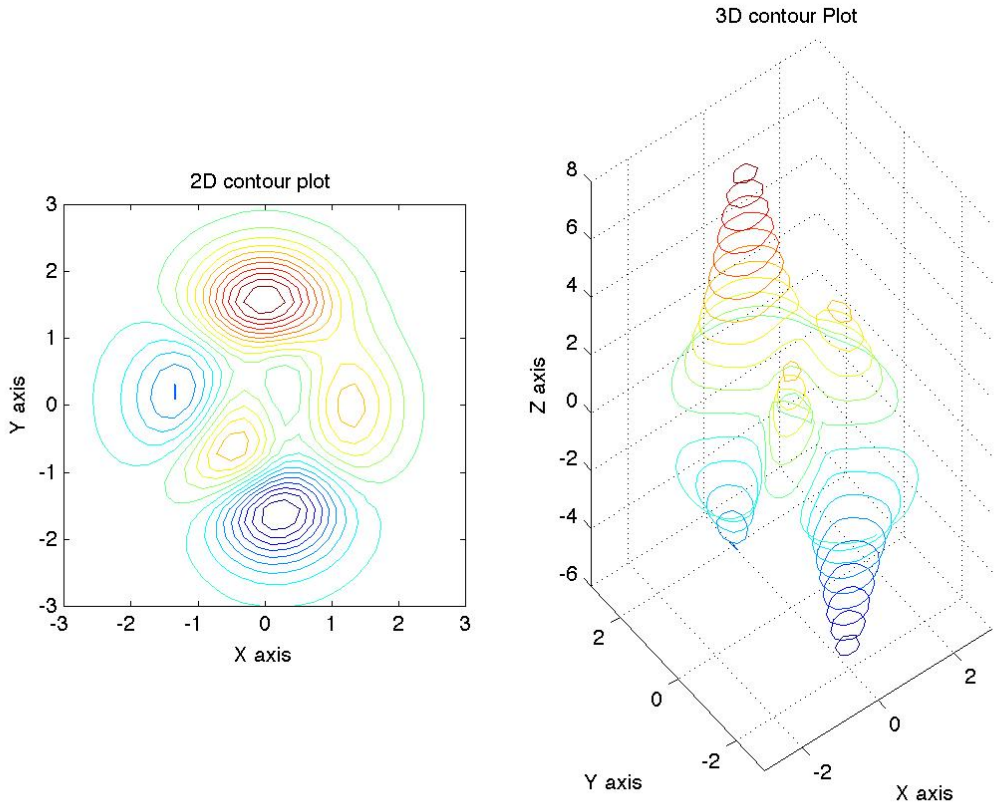
# Contour Plots

- Contour plots show lines of constant elevation of height. A topographical map is a contour map.
- Consider the following MatLab code (**L08plot3N.m**):

```
% Use standard peaks surface
[X,Y,Z]=peaks(30);
subplot(1,2,1);
% generate 20 2D contour lines
contour(X,Y,Z,20);
axis square;
xlabel('X axis');
```

```
ylabel('Y axis');
title('2D contour plot');
subplot(1,2,2);
% The same contour in 3D
contour3(X,Y,Z,20);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('3D contour Plot');
print L08plot3N.jpg -djpeg
```

The produces the plot:




2D and 3D contour plots of the peaks data

- The function `pcolour` maps height to a set of colours and presents the same information as the contour plot at the same scale.

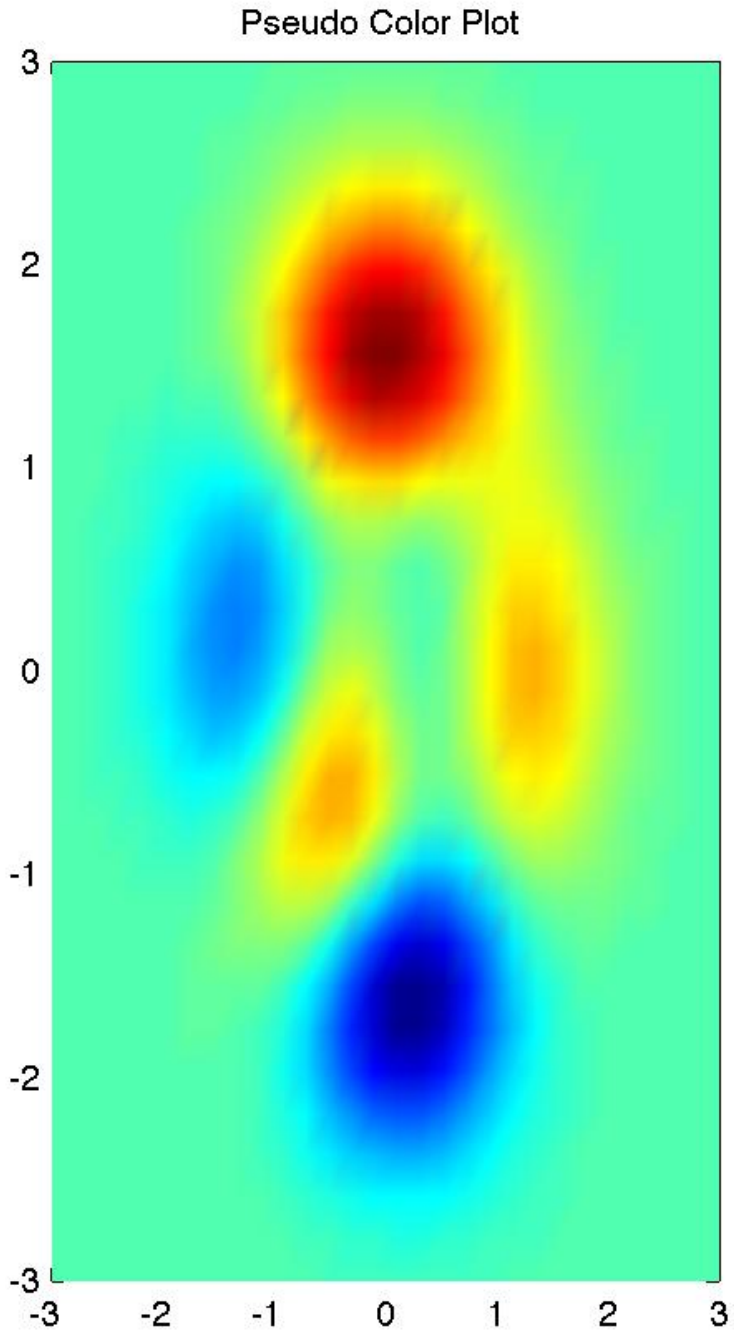
- Function `conrourf` combines the pseudo color plot with a 2D contour plot to produce a filled contour.

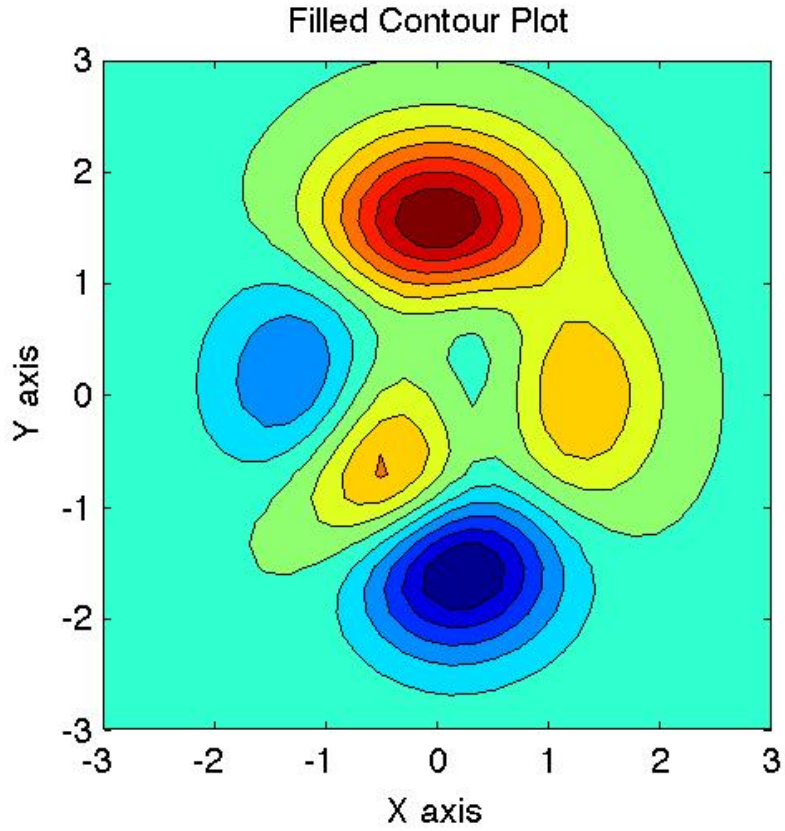
- The following MatLab code (**L08plot3O.m**):

```
subplot(1,2,1)
[X,Y,Z]=peaks(30);
pcolor(X,Y,Z);
shading interp
axis square
title('Pseudo Color Plot');
```

```
subplot(1,2,2);
% filled contour plot with 12 contours
contourf(X,Y,Z,12);
axis square
xlabel('X axis');
ylabel('Y axis');
title('Filled Contour Plot');
print L08plot3O.jpg -djpeg
```

produces the plot:


Pseudo Color Plot
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


Filled Contour Plot
Y axis
X axis
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

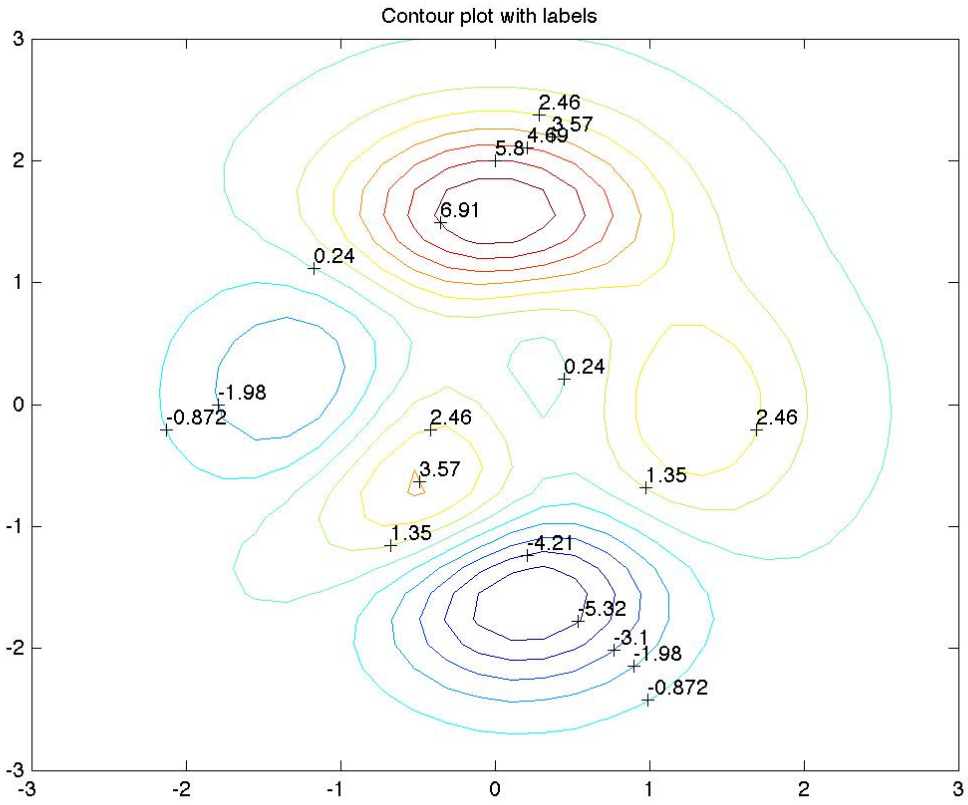
# Labelled Contour Lines

- Contour lines can be labelled using the `@clabel` function, which requires a matrix of lines and text strings that are return by `contour`, `contourf` and `contour3`.

- The following MatLab code (**L08plot3P.m**):

```
[X,Y,Z]=peaks(30);
C=contour(X,Y,Z,12);
clabel(C);
title('Contour plot with labels');
print L08plot3P.jpg -djpeg
```

produces the plot:




2D Contour Plot With Labels

# Ribbon Plots

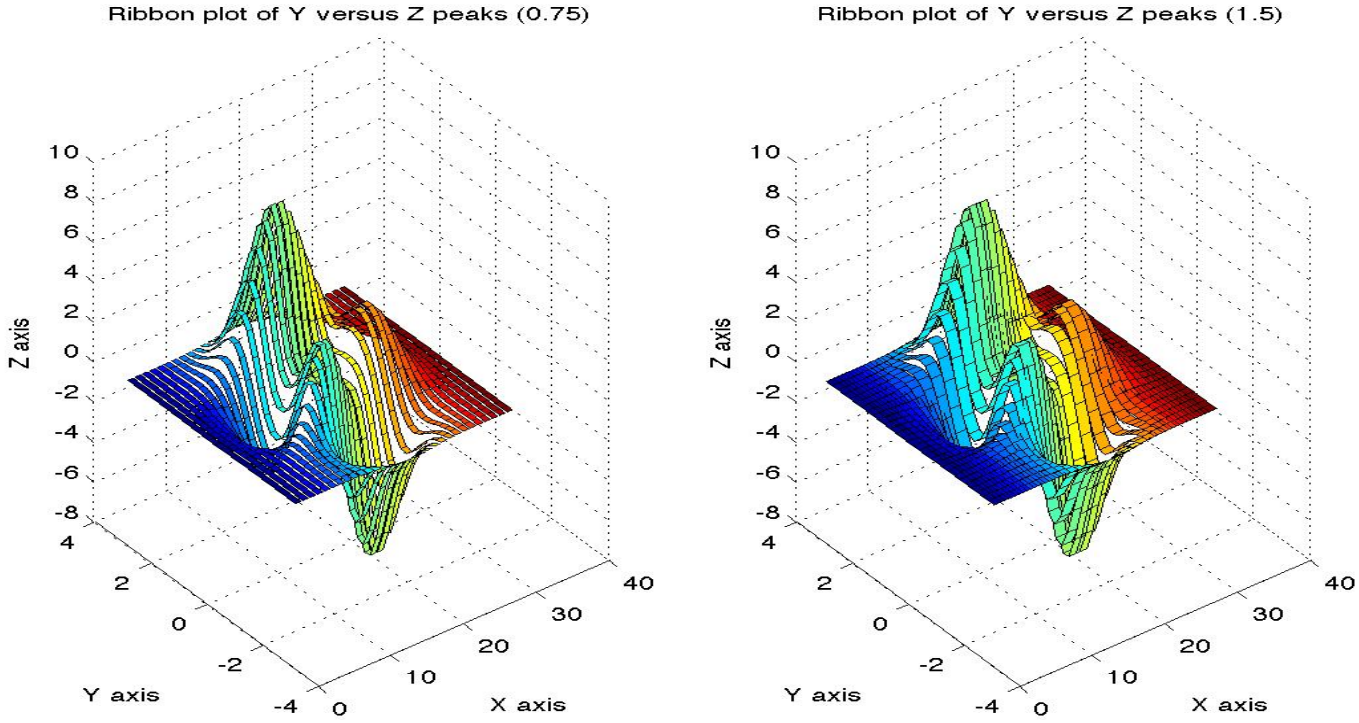
- MatLab provides a way to plot "ribbon" flows of functions. Each ribbon plot the contours of some dimension, say `T`, as separate ribbons.
- The function `ribbon(Y,Z)` plots the columns of `Y` against `Z`.
- The width of a ribbon can be specified as a $3^{rd}$ argument of `ribbon`, where the default width being 0.75.
- Consider the following MatLab code (**L08plot3Q.m**): which produces the plot:

```
% X and Y default to the size
% of the array produced for Z.
```

```
[X,Y,Z]=peaks(30);
subplot(1,2,1)
ribbon(Y,Z,0.75);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Ribbon plot of Y versus Z peaks (0.75)');
subplot(1,2,2)
ribbon(Y,Z,1.5);
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
```

```
title('Ribbon plot of Y versus Z peaks (1.5)');
print L08plot3Q.jpg -djpeg
```

which produces the 2 plots:



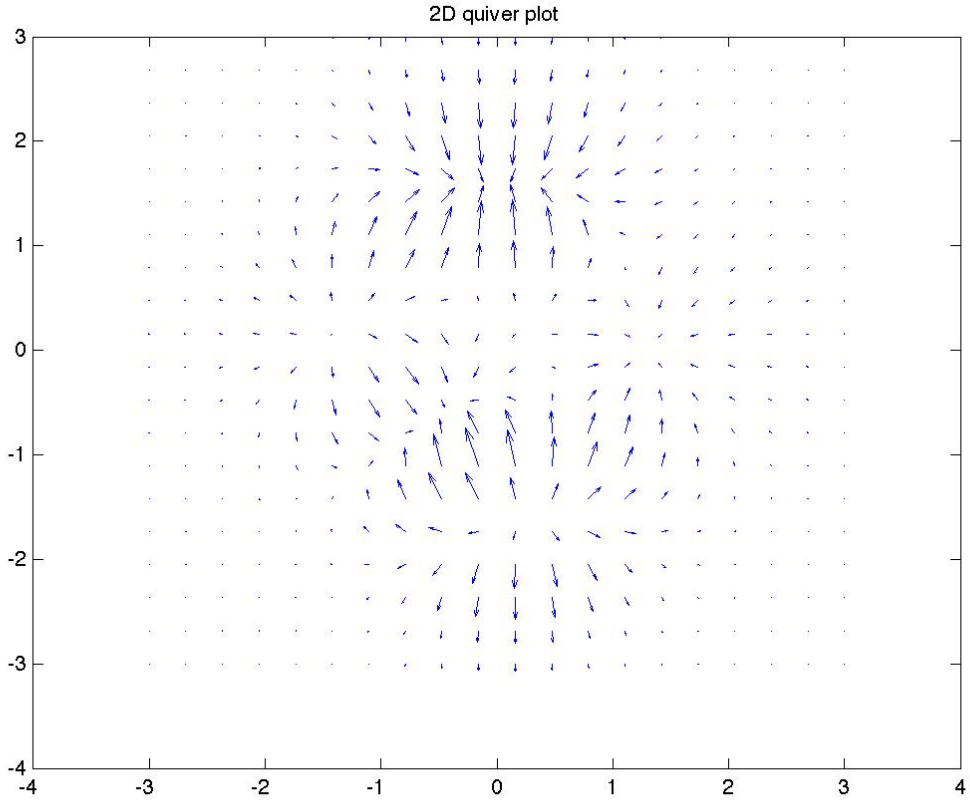

3D ribbon plots with width 0.75 and 1.5 respectively

# 2D Quiver Fields

- The function `quiver(x,y,dz,dy)` draws a vector field `(dx,dy)` at points `(x,y)`.
- Consider the following MatLab code (**L08plot3R.m**):

```
[X,Y,Z]=peaks(20);
% Compute the gradient of Z with respect
% to X and Y for values 0.5,0.5
[dx,dy]=gradient(Z,0.5,0.5);
quiver(X,Y,dx,dy);
title('2D quiver plot');
print L08plot3R.jpg -djpeg
```

The code produces the plot:




2D quiver field

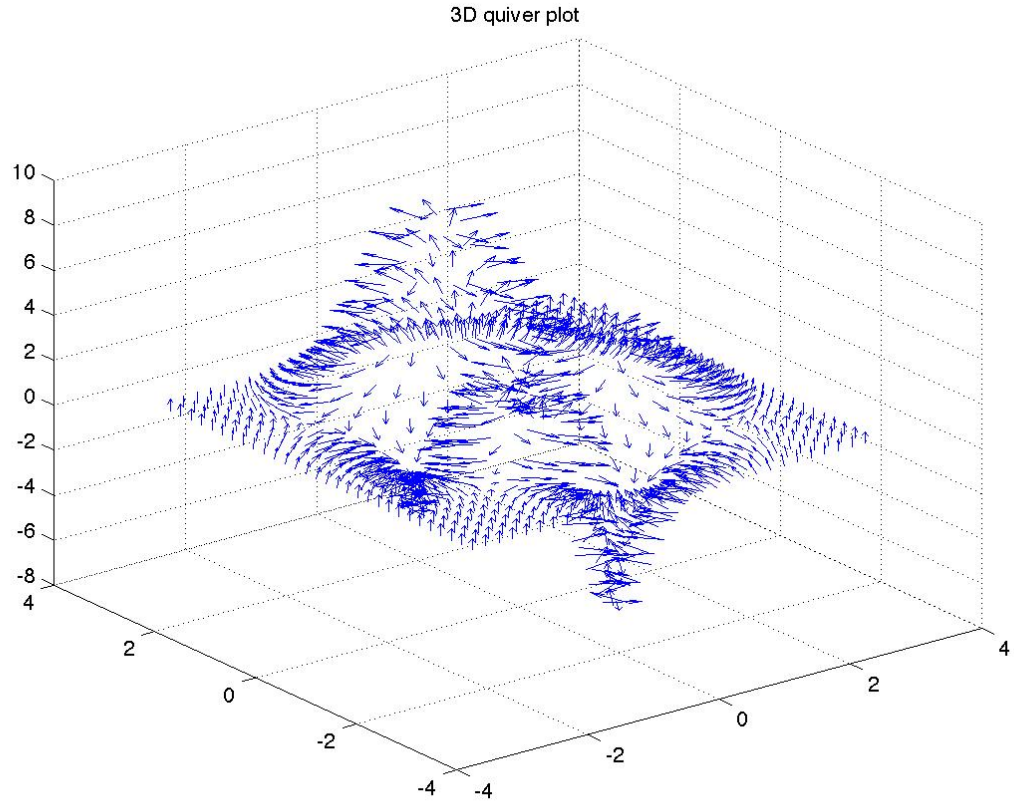
# 3D Quiver Fields

- 3D quiver plots `quiver3(x,y,z,dx,dy,dz)` display the vectors `(dx,dy,dz)` at the 3D points `(x,y,z)`.

- The MatLab code (**L08plot3S.m**):

```
[X,Y,Z]=peaks(30);
[Ny,Ny,Nz]=surfnorm(X,Y,Z);
% We won't display the 3D surface
quiver3(X,Y,Z,Nx,Ny,Nz);
title('3D quiver plot');
print L08plot3S.jpg -djpeg
```

produces the 3D vector field plot:




3D quiver field

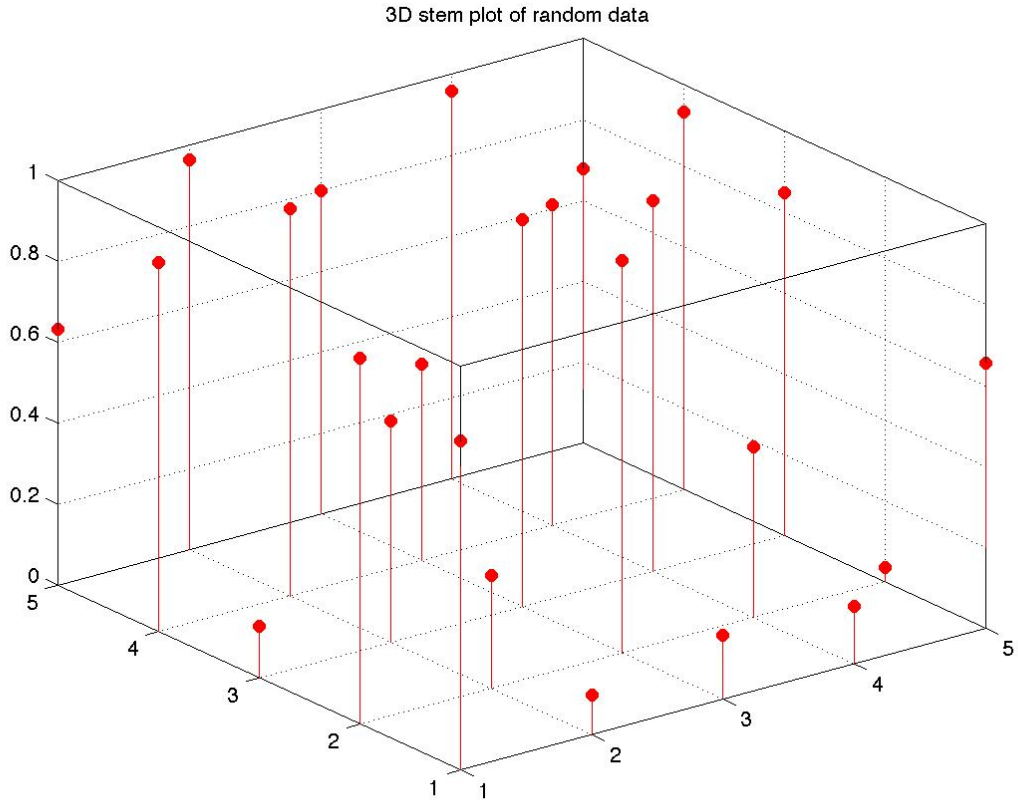
# 3D Stem Plots

- The 3D version of stem plots extends naturally from 2D. The command `stem3(X,Y,Z,C,'filled')` plots data points in `(X,Y,Z)` with lines extending to the $x-y$ plane. The optional `C` argument specifies the marker type of the colour and the optional 'filled' argument causes the marker to be filled in, `stem3(Z)` points the values in `Z` and automatically generates the `X` and `Y` values.

- Consider the MatLab code (**L08plot3T.m**):

```
Z=rand(5,5) % rand(5) gives the same values
stem3(Z,'ro','filled');
grid on
title('3D stem plot of random data');
print L08plot3T.jpg -djpeg
```

This code produces the following plot:




3D stem field

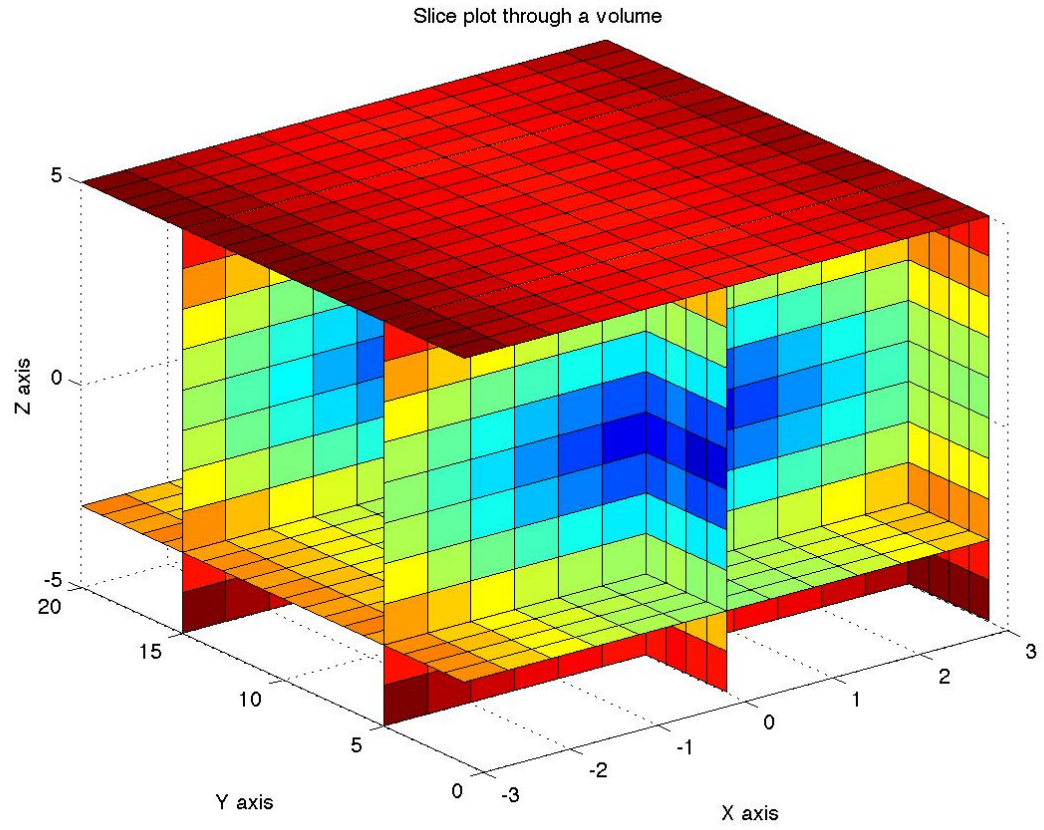
# Volume Visualization - Slice Plots

- In addition to mesh, surface and contours plots, MatLab offers volume visualization capabilities.
- As an example of slice images, consider the following MatLab code (**L08plot3V.m**):

```
x=linspace(-3,3,13);
y=1:20;
z=-5:5;
[X,Y,Z]=meshgrid(x,y,z);
V=sqrt(X.^2+cos(Y).^2+Z.^2);
slice(X,Y,Z,V,[0 3],[5 15],[-3 5]);
```

```
xlabel('X axis');
ylabel('Y axis');
zlabel('Z axis');
title('Slice plot through a volume');
print L08plot3V.jpg -djpeg
```

The following figure is plotted:



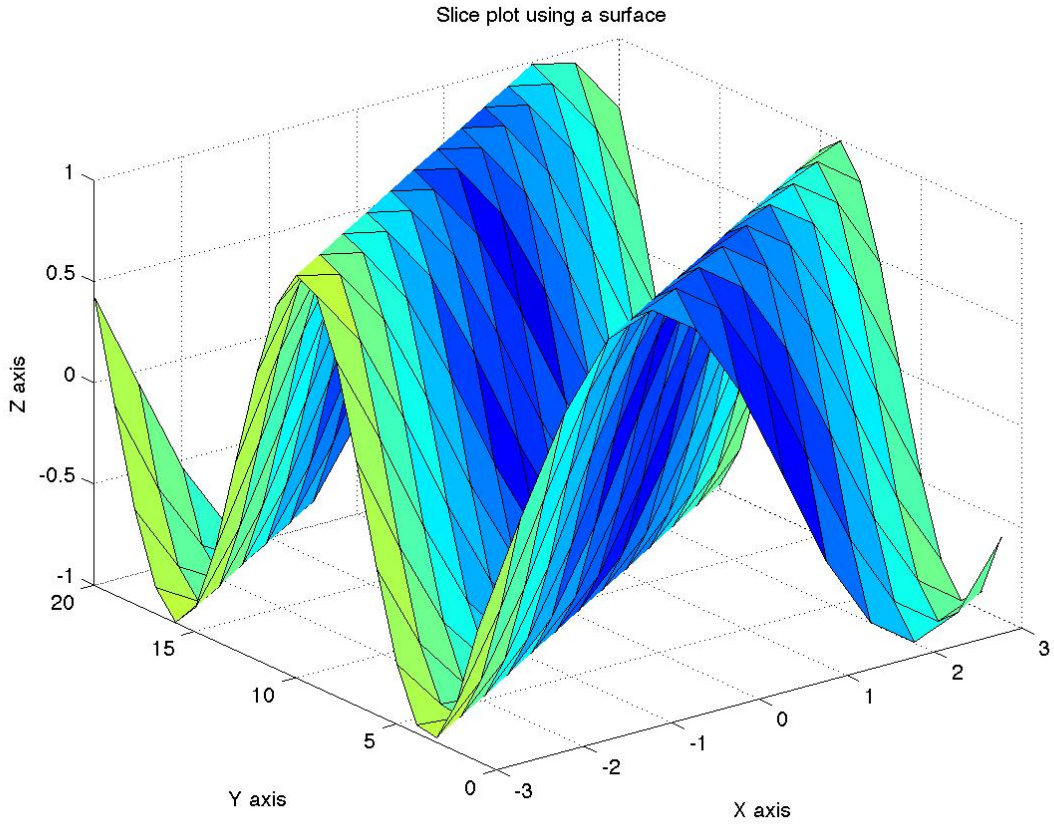

3D slice plot through a volume. Note the slices at X=0 and X=3, the slices at Y=5 and Y=15 and the slices at Z=-3 and Z=5.

- Slices do not have to be planes. They can be surfaces. Consider the following MatLab code (**L08plot3W.m**):

```
x=linspace(-3,3,13);
y=1:20;
z=-5:5;
[X,Y,Z]=meshgrid(x,y,z);
[xs,ys]=meshgrid(x,y);
% Generate a surface
zs=sin(-xs+ys/2);
V=sqrt(X.^2+cos(Y).^2+Z.^2);
slice(X,Y,Z,V,xs,ys,zs);
xlabel('X axis');
```

```
ylabel('Y axis');
zlabel('Z axis');
title('Slice plot using a surface');
print L08plot3W.jpg -djpeg
```

This code defines a surface using `xs`, `ys` and `zs`. The following figure is plotted:




3D slice plot using a surface

# Slice Plot for MRI Cardiac Data

- Consider a 4D volume of MRI cardiac gated data (20 3D volumes of 1 heart beat). Suppose we wish to visualize volume 9. We can use `slice` to view orthogonal slices of the data. Programs to move these slices interactively while viewing the data can be written in MatLab.

- Consider the following MatLab code (**L08plot3U.m**), a long example):

```
% The variable name used by save was vol9
% A volume is 75 slices of 256 by 256
% unsigned short data: grayvalues in
% the range 0-4095, i.e 12 bit data,
% stored as 16 bit (2 byte) data
load MRI_DATA_9.mat
fprintf('9th volume of MRI data loaded\n')
```

```
sample_xy=8;
sample_z=4;
% Size of volume: 75*256*256 shorts
[end_z,end_x,end_y]=size(vol9);

start_z=1;
start_x=1;
start_y=1;
fprintf('start_z=%d start_x=%d start_y=%d\n',...
        start_z,start_x,start_y);
fprintf('end_z=%d end_x=%d end_y=%d\n',...
        end_z,end_x,end_y);

vol9=cast(squeeze(vol9),'double');
% subsample the data to allow MatLab to display
% Data now 19 by 32 by 32
vol9=vol9(start_z:sample_z:end_z,...
          start_x:sample_xy:end_x,...
          start_y:sample_xy:end_y);

% must switch the first 2 arguments
```

```
% of meshgrid in 3D
[Z,X,Y]=meshgrid(1:size(vol9,2),...
                 1:size(vol9,1),...
                 1:size(vol9,3));

% The data has been subsampled so we must
% subsample the z, x and y slice
% numbers to be displayed as well
% Swap slice numbers as in meshgrid
% Original slice numbers 36,128,128
% become 9,16,16
slice_number_z=36/sample_z;
slice_number_x=128/sample_xy;
slice_number_y=128/sample_xy;

% Generate  xy, xz and yz images
% for these slice numbers
% 1. Color the images with the summer colormap
% using grs2rgb available at
% http://www.mathworks.com/matlabcentral/
% fileexchange/13312-grayscale-to-rgb-converter/
```

```
% content/grs2rgb.m
% 2. Enlargen the images by a factor
% of 5 using imresize
imz=squeeze(vol9(slice_number_z,:,:));
imx=squeeze(vol9(:,slice_number_x,:));
imy=squeeze(vol9(:,:,slice_number_y));
slice_z=imresize(grs2rgb(imz,...
        colormap(summer(4096))),5);
slice_x=imresize(grs2rgb(imx,...
        colormap(summer(4096))),5);
slice_y=imresize(grs2rgb(imy,...
        colormap(summer(4096))),5);
imwrite(slice_z,'slice_z.jpg');
imwrite(slice_x,'slice_x.jpg');
imwrite(slice_y,'slice_y.jpg');

imshow(slice_z,[]);
title(['vol9(' num2str(slice_number_z) ',:,:)']);

figure
imshow(slice_x,[]);
```

```
title(['vol9(:,' num2str(slice_number_x) ',:)']);

figure
imshow(slice_y,[]);
title(['vol9(:.:,' num2str(slice_number_x) ')']);

% Generate 3D slice image
% Exchange Z and X or slice_number_x
% and slice_number_z to be consistent
figure
slice(Z,X,Y,vol9,slice_number_x,...
      slice_number_z,slice_number_y);
% could use colormap(gray(4096));
colormap(summer(4096));
axis tight
shading interp
view(-37.5,30.0);
% Swap axis labels as well
zlabel('Y axis');
xlabel('X axis');
ylabel('Z axis');
```

```
title({['Slice plots of 3 orthogonal planes'...
       ' in the 9th volume of a subsampled' ...
       ' MRI cardiac dataset'];
       ['Original data size:' num2str(end_z)...
        ' by ' num2str(end_x)...
        ' by ' num2str(end_y) '   ' ...
        'Subsampled data size:' num2str(size(vol9,1))...
        ' by ' num2str(size(vol9,2))...
        ' by ' num2str(size(vol9,3))];
       ['Subsampling in z by ' num2str(sample_z) ', '...
        'subsampling in x and y by ' num2str(sample_xy)];
       ['Displaying:' ...
        num2str(slice_number_z) 'th z slice  ' ...
        num2str(slice_number_x) 'th x slice  ' ...
        num2str(slice_number_x) 'th y slice  ']});

print L08plot3U.jpg -djpeg
```

- Note that MatLab often cannot nicely display volume data because there

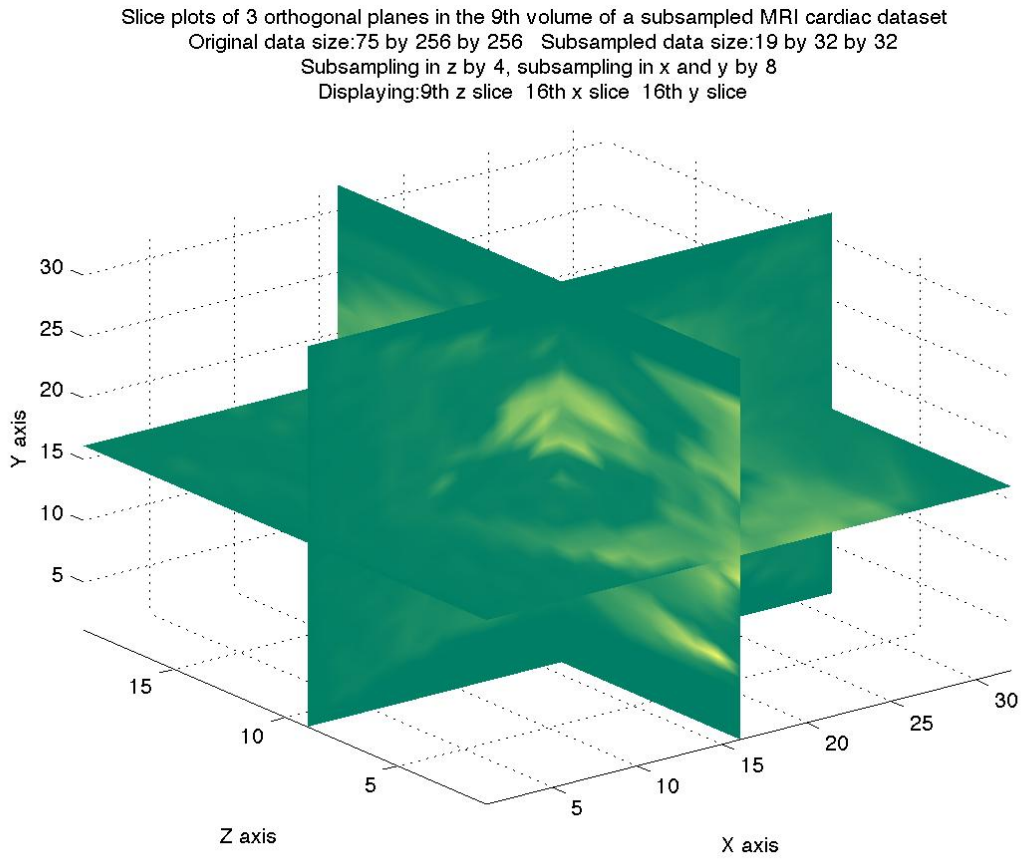
is too much data. In the above code, we handled this problem by sub-sampling the data (by 4 in $z$ and by 8 in $x$ and $y$). Matlab does have a function `reduce volume` which reduces the data (in some unknown way).

- The $xy$, $xz$ and $yz$ subsampled images are shown below:

**xy** **xz** **yz**

The $xy$ ($z = 36$), $xz$ ($y = 128$) and $yz$ ($x = 128$) subsampled MRI images

- Finally, the following 3D slice plot is produced:




3D slice plot of 3D MRI data

# Easy Plotting

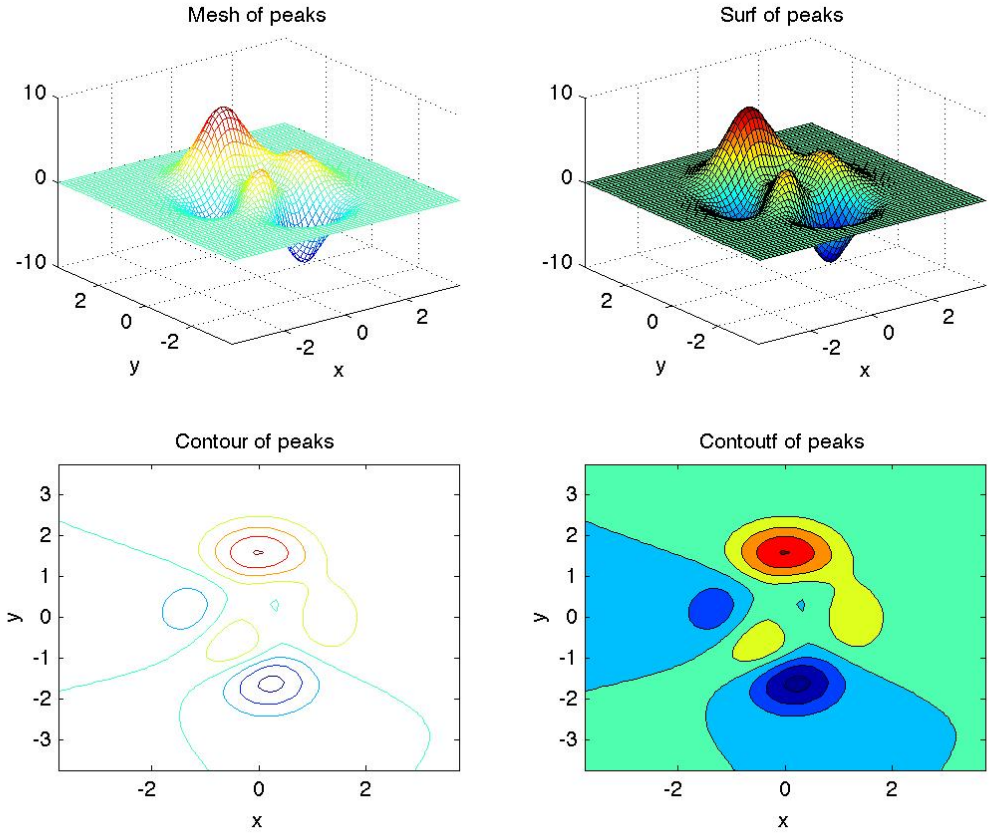
- There may be times when you want to let MatLab specify the data points for a 3D plot.
- MatLab has functions `ezcontour3`, `ezcontourf`, `ezmesh`, `ezmeshc`, `ezplot3`, `ezsurf` and `ezsurfc` that plot the data as their non `ez` counterparts but usually take string expressions as arguments.
- The following MatLab code (**L08plot3X.m**):

```
fstr=['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)'];
```

```
subplot(2,2,1)
ezmesh(fstr);
title('Mesh of peaks');

subplot(2,2,2)
ezsurf(fstr);
title('Surf of peaks');

subplot(2,2,3)
ezcontour(fstr);
title('Contour of peaks');
```

```
subplot(2,2,4)
ezcontourf(fstr);
title('Contoutf of peaks');

print L08plot3X.jpg -djpeg
```

produces the plot:




Four easy plots