3D Plots in MATLAB

Leigh J. Little SUNY Brockport October 31, 2022

Contents

1 Three Dimensional Plotting

Three dimensional plotting encompasses a broad spectrum of graphs, but generally such graphs fall into 3 main categories

- Plotting a three dimensional parametric curve $(x(t), y(t), z(t))$ for $t \in [a, b]$.
- Plotting a function of the form $z = f(x, y)$. In this case, z is a function of two independent variables x and y. Surface plots and contour maps fall into this category.
- Plotting the value of some function $w = f(x, y, z)$. This is called volumetric visualization. In this case, w is a function of three independent variables x, y and z. This is the most challenging case because the domain is three dimensional region in space. Because we can only see in three dimensions, this requires the use of slice plots and isosurfaces to see the values of w that lie 'inside' the domain.

2 Parametric Curves in 3D

A three-dimensional parametric curve is not significantly different than a two-dimensional parametric curve, both in terms of how the data is represented and how it is plotted. The classic example of such a curve is a circular helix (or a spring).

$$\begin{aligned} x(t) &= R\cos(t); \quad t \in [a, b] \\ y(t) &= R\sin(t) \\ z(t) &= St. \end{aligned}$$

The x and y variables trace out a circle and the z variable changes the elevation of the circle as t increases.

```
clear
close all
R = 2;
S = 1;
t = linspace(0,6*pi,201);
x = R*cos(t);
y = R*sin(t);
```

```
z = S*t;
plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
```

This is shown in Figure 1. Note that the `plot` command has changed to `plot3`. This is the MATLAB command for plotting general three-dimensional line graphs.

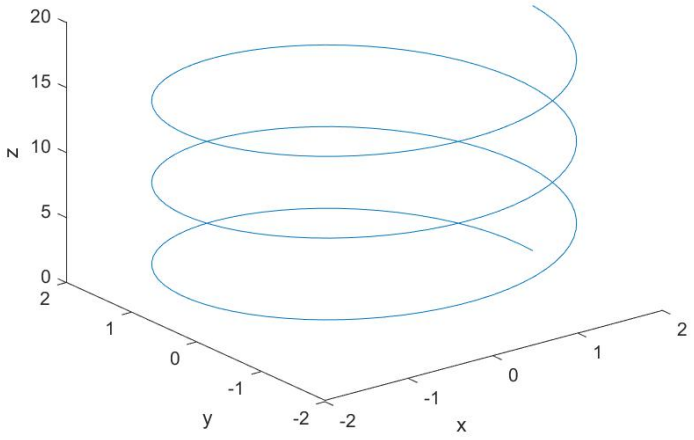

Figure 1: Circular helix.

3 Functions of the Form $z = f(x, y)$

One of the most common three-dimensional functions that needs to be viewed is a function of the type $z = f(x, y)$. In this case, the input to the function is a pair of (x, y) coordinates in the plane. The output from the function is the scalar z. z represents the elevation of $f(x, y)$ above or below the x-y plane. Viewing functions of this type depends on how the data is provided. This can be broken down into two broad categories

- Gridded data - in this case, the domain of the function is (usually) a rectangular region of the x-y plane and this region is divided into a grid of intersecting horizontal and vertical lines. (see Figure 2). This is the most common case.
- Non-gridded Data - in this case, the data points are spread throughout the domain, but don't have an organized structure to them. This case is more challenging.

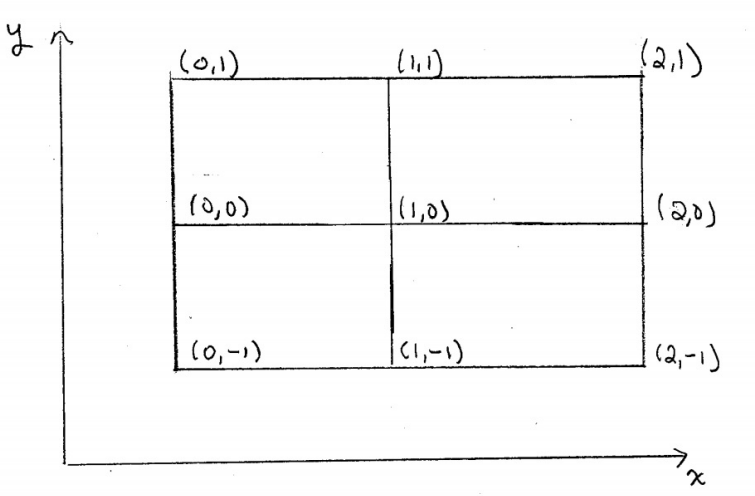

Figure 2: A small gridded domain in the x-y plane.

3.1 Gridded Data

For gridded data sets, you typically are given a set of x-coordinates (which can be equally spaced or not) and a set of y-coordinates. The spacing of the y-coordinates can be different than that of the x-coordinates. When vertical and horizontal lines are plotted through these points, this generates a grid. The z values are often given as a matrix of values, one for each point of intersection. The z values can be the result of a computer simulation, physical measurements (such as in a topographical map) or defined by a mathematical function. We will examine this latter case here.

In order to get the data in the proper form for plotting, it is necessary to define the domain, divide the domain into intervals, then create what is called a *meshgrid*. The value of z is computed on the meshgrid.

What is a meshgrid? This is a device that easily allows for the computation of functions of two or more variables in gridded data situations. Suppose we subdivide $x \in [0, 2]$ as

$$x = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$$

and $y \in [-1, 1]$ as

$$y = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}.$$

This results in the grid shown in Figure 2. We need a way to easily compute the value of $z = f(x, y)$ at each of these grid points.

One way would be to do a double nested loop

```
 for i = 1:length(x)
     for j in length(y)
         z(i,j) = .....
     end
end
```

however, this is going to be computationally inefficient for grids that have a large number of points. We would prefer to evaluate the entire grid in a single command. A meshgrid is what permits this.

A meshgrid is a set of two matrices; a matrix of x-coordinates and a matrix of y-coordinates. When these two matrices are superimposed, they create the grid shown in Figure 2. The commands below will generate the meshgrid.

```
x = linspace(0,2,3);
y = linspace(-1,1,3);
[X,Y] = meshgrid(x,y);
```

Note the that output from the `meshgrid` command is two matrices. The X matrix looks like

$$X = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}$$

while the Y matrix looks like

$$Y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

When these are mentally superimposed, you get the 'matrix' below

$$M = \begin{pmatrix} (0,-1) & (1,-1) & (2,-1) \\ (0,0 & (1,0) & (2,0) \\ (0,1) & (1,1) & (2,1) \end{pmatrix}$$

This is a matrix representation of the same grid in Figure 2 except that the y-coordinates are flipped top to bottom.

Once the meshgrid has been created, it is easy to compute the value of z on the grid. It can then be plotted in a variety of ways. For example, suppose we wanted to plot

$$z = e^{-(x^2+y^2)}$$

on the domain $x \in [-1,1], y \in [-1,1]$. The following commands will perform the various tasks above:

```
x = linspace(-1,1,21);      % Generate 21 points from -1 to 1
y = linspace(-1,1,41);      % Generate 41 points from -1 to 1
[X,Y] = meshgrid(x,y)         % Generate mesh grid (note capital X,Y)
Z = exp(-(X.^2 + Y.^2))     % Compute Z on the mesh grid (X,Y)
```

Different numbers of points have been used in each direction to make the distinction between the x and y axes more clear.

3.1.1 Surface Plot

In a surface plot, the gridded domain is treated like a set of rectangles. Each corner of the rectangle has an elevation specified by the corresponding value of z. This gives a series of interlocked, small, three dimensional planes. To create the generic surface plot, you can just do

```
surf(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
```

Note that the inputs to the `surf` command are the meshgrid matrices X and Y and the matrix Z. This is shown in Figure 3.

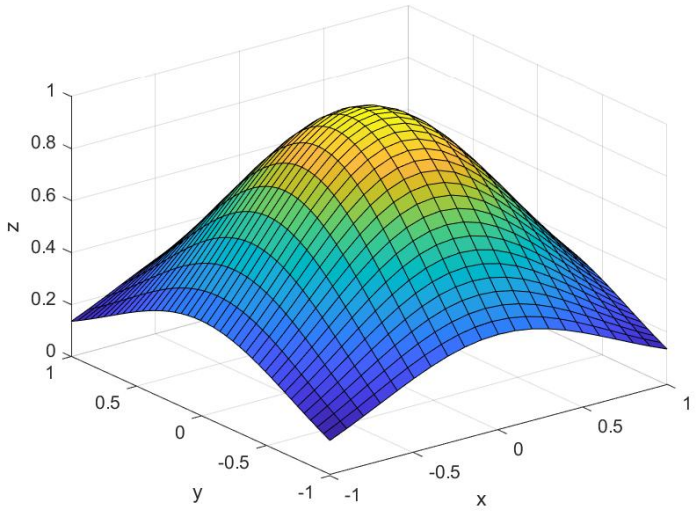

Figure 3: Simple surface plot.

Notice that each of the small rectangles has a uniform color. This color is determined by the average of the z values at the corner of each rectangle. If you want to create a surface plot without showing grid lines, you can issue the command

```
>> shading flat
```

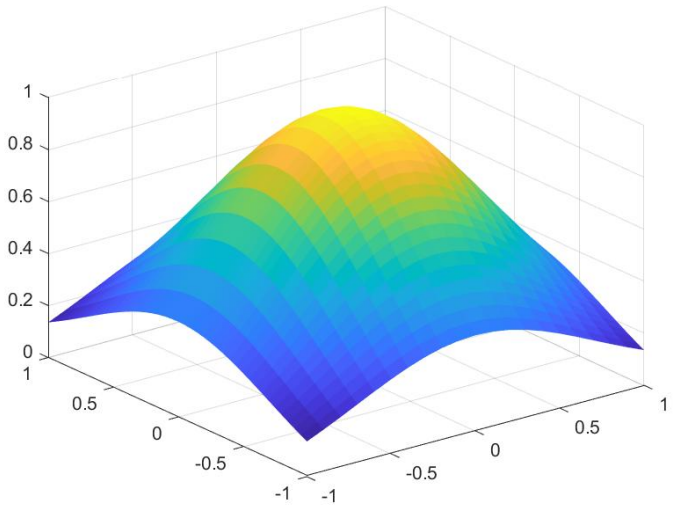

Figure 4: Surface plot with flat shading.

The resulting plot is shown in Figure **??**. Notice that you can still see the solid colors of the individual rectangles.

If you want to create a surface plot with a smoother level of color changes, you can issue the command

```
>> shading interp
```

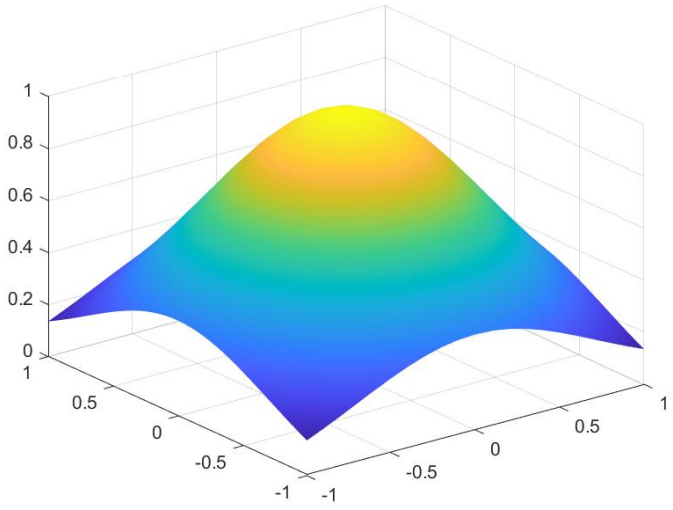

Figure 5: Surface plot with interpolated shading.

The resulting plot is shown in Figure 5.

3.1.2 Color Maps

As the name implies, color maps define the sequence of colors used to color the surface plot. Each z elevation corresponds to a different color. MATLAB has many built-in color maps (search the `colormap` command in the help window to see these), or you can create your own. For example,

```
surf(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
colormap(summer)
colorbar
```

This is shown in Figure 6.

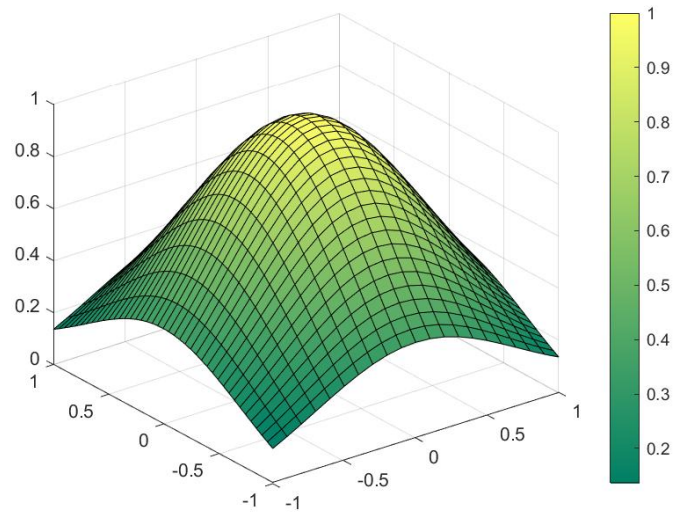

Figure 6: Surface plot with the `summer` color map and a color bar.

3.1.3 Wireframe Plots

A wireframe plot is similar to a surface plot, but there is no coloring of the planar sections. Use the `mesh` command to get a wireframe plot.

```
mesh(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('z')
```

The result is shown in Figure 7.

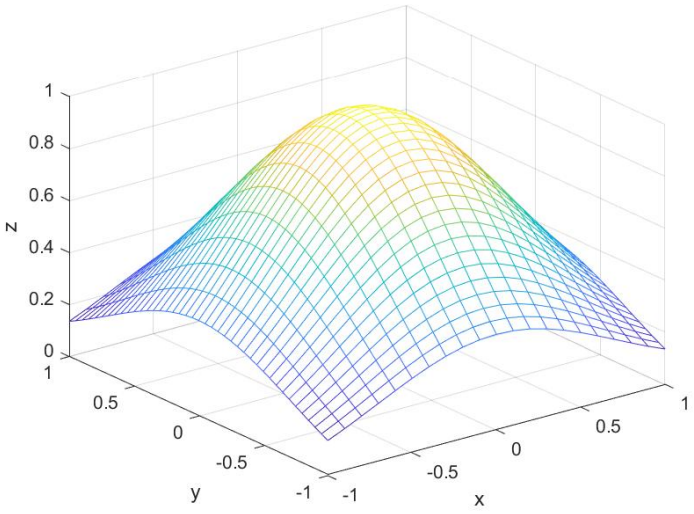

Figure 7: Wireframe plot.

3.1.4 Contour Plots

You are probably familiar with contour maps. These are a convenient way to view a three-dimensional surface on a two-dimensional plane. These are often used on maps to illustrate terrain heights. To generate a basic contour map, you can do

```
contour(X,Y,Z)
colormap(copper)
xlabel('x')
ylabel('y')
```

The result is shown in Figure 8.

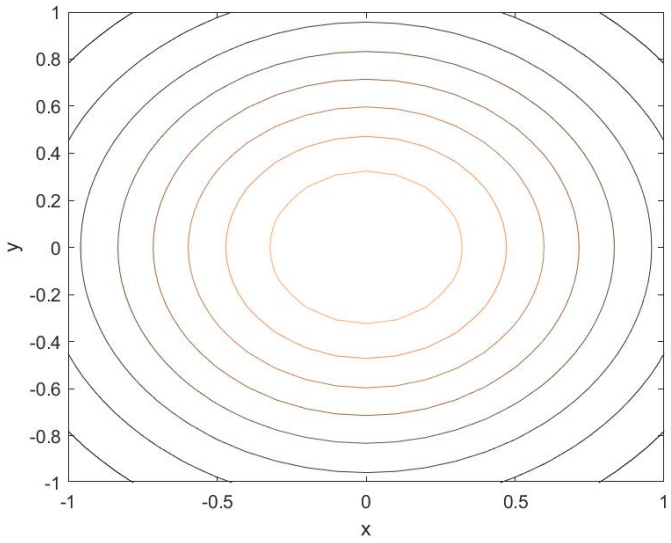

Figure 8: Simple contour plot.

By defaut, this will draw 8 contours with the z value of each contour equally spaced between the minimum and maximum values of z. If you want more contours, you can do

```
contour(X,Y,Z,20)
colormap(copper)
xlabel('x')
ylabel('y')
```

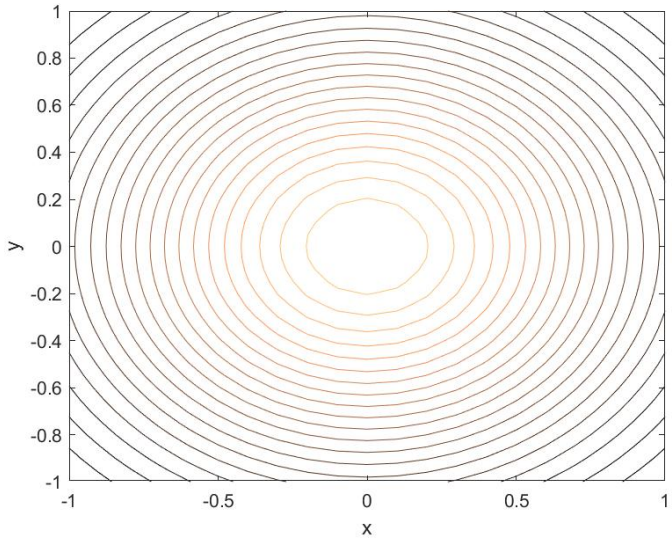

Figure 9: Simple contour plot with 20 contours.

You can also specify exactly which contours you want to draw by sending a vector with the desired levels into the `contour` command (see Figure 10)

```
contour(X,Y,Z,[0.2 0.6 0.7 0.72 0.73 0.74 0.9])
colormap(copper)
xlabel('x')
ylabel('y')
```

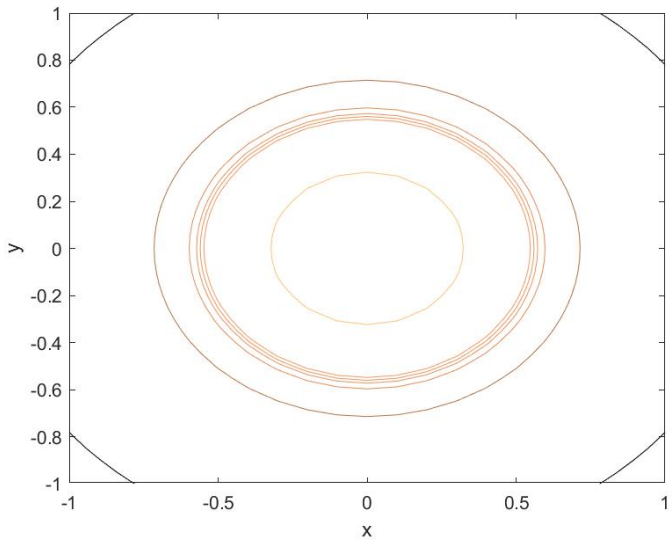

Figure 10: Simple contour plot with specified contours.

You can also label your contours with the elevation value (see Figure 11). Note the slightly different command when calling the `contour` function.

```
c = contour(X,Y,Z)
clabel(c)
```

```
colormap(copper)
xlabel('x')
ylabel('y')
```

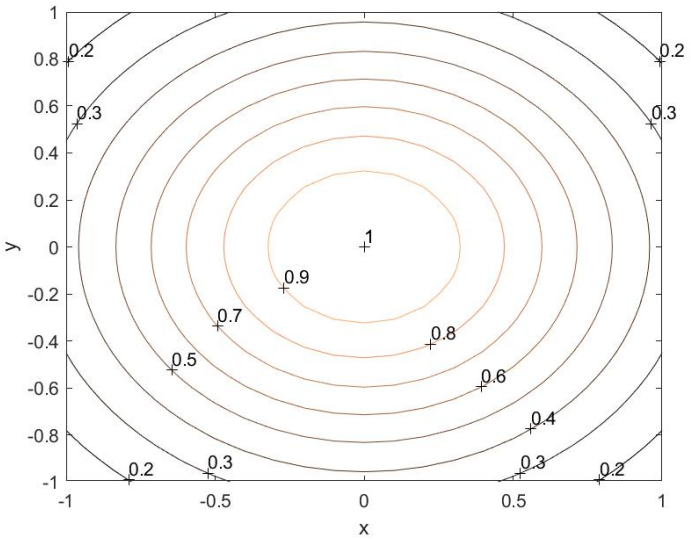

Figure 11: Contour plot with contour labels.

3.1.5 Three Dimensional Contour Plots

You can create three-dimensional contour plots using (see Figure 12)

```
contour3(X,Y,Z)
colormap(copper)
xlabel('x')
ylabel('y')
```

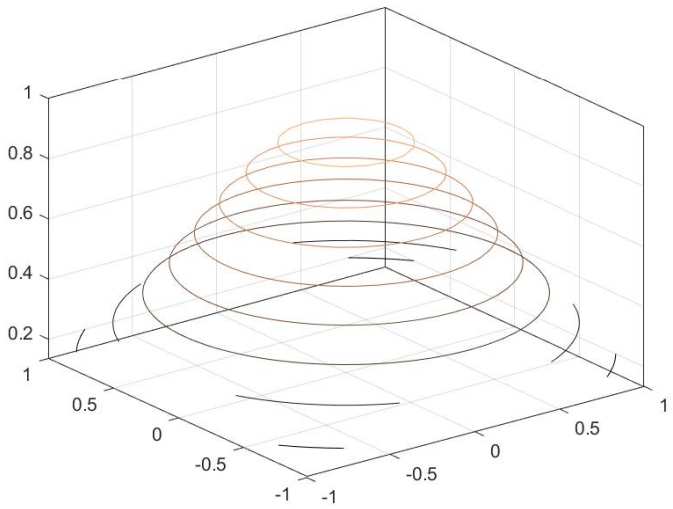

Figure 12: Three-dimensional contour plot.

3.1.6 Filled Contour Plots

You can create filled contour plots using (see Figure 13)

```
contourf(X,Y,Z)
colormap(copper)
xlabel('x')
ylabel('y')
```

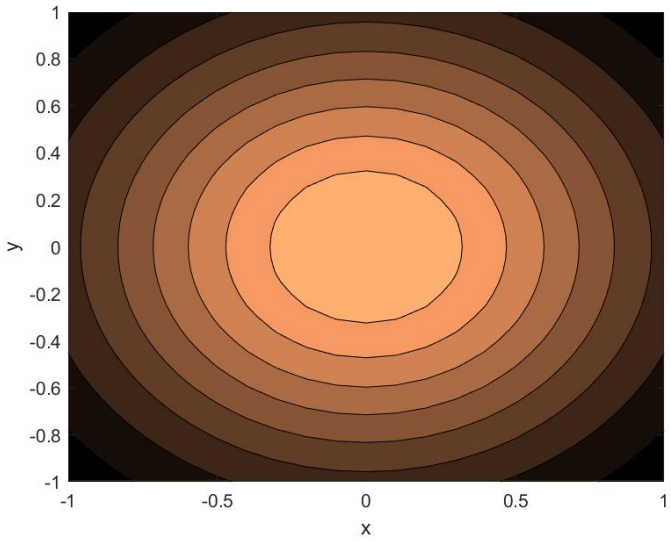

Figure 13: Filled contour plot.

3.2 Non-gridded Data

When the data is not available in a gridded format, the situation becomes much more complicated. One option is to interpolate the data to a gridded data set, but this might not always be feasible. This is because the most likely situation to encounter non-gridded data is when the domain is not rectangular or contains holes.

The most common approach used in this situation is to create a *triangulation* of the points, then generate a surface plot on this triangulation. Creating optimal triangulations of non-gridded data is a very active area of research.

To demonstrate, we will generate some random data points, then compute

$$z = e^{-(x^2+y^2)}$$

at this set of points.

The basic random number generator will generate a number in the interval $[0, 1]$. To get a random number in the interval $[a, b]$, use

```
x = a + (b-a)*rand
```

For this example, generate 100 random points in the domain and plot the individual points.

```
a = -1;
b = 1;
x = a + (b-a)*rand(100,1);
y = a + (b-a)*rand(100,1);
z = exp(-(x.^2+y.^2));       % Note: x,y,z are vectors, not matrices
```

```
plot3(x,y,z,'k*')
xlabel('x')
ylabel('y')
zlabel('z')
```

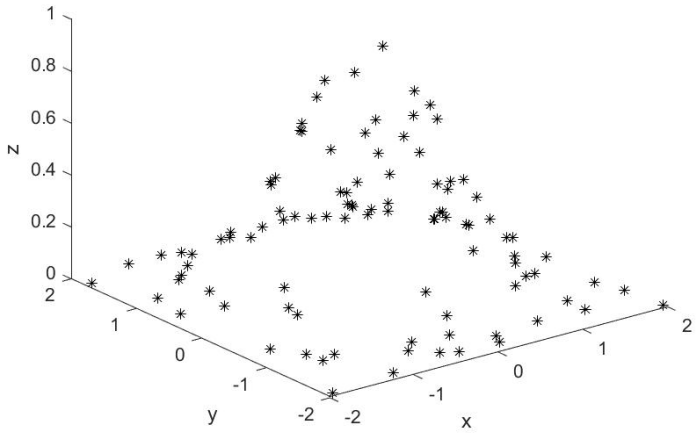

Figure 14: A set of non-gridded points in three-dimensional space.

As you can see, the points map out a shape similar to the surface plots we saw before. We can't use any of the previous plots because the data is not on a regular grid.

What we can do is to create a triangulation of the data, then do a surface plot of this triangulation. A triangulation is an attempt to create a series of non-overlapping triangles that cover the area of the domain occupied by the irregular points. Once you have these triangles, a process similar to a surface plot can be performed.

```
a = -1;
b = 1;
x = a + (b-a)*rand(100,1);
y = a + (b-a)*rand(100,1);
z = exp(-(x.^2+y.^2));       % Note: x,y,z are vectors, not matrices
tri = delaunay(x,y);    % Create the triangulation using x and y.
trisurf(tri,x,y,z)      % Create a surface plot of the triangulation
xlabel('x')
ylabel('y')
zlabel('z')
```

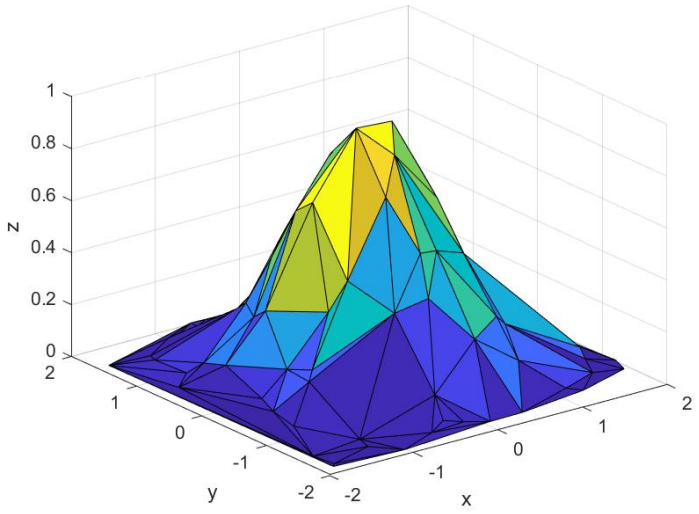

Figure 15: A sufrace plot of the triangulation of the non-gridded points.

4 Volumetric Visualization

Volumetric visualization occurs when you need to view the value of a function of the form $w = f(x, y, z)$. In theory, you could just generalize the graphs we just examined; create a three-dimensional meshgrid, then evaluate w on the grid. There is a problem with this. We can't see past three dimensions. There is no fourth axis to act like the height w as we did with surface plots. In this case, the main tools used are slicing plots and isosurface plots.

4.1 Slicing

The most common technique for viewing 'inside' a volume is slicing. This is a fast and easy way to reveal areas inside the volume where interesting things might be happening. The following commands will generate a volume of

$$z = e^{-(x^2+y^2+z^2)}$$

on the cube

$$x \in [-1, 1], y \in [-1, 1], z \in [-1, 1].$$

```
clear
close all
x = linspace(-1,1,21)';
y = x;
z = x;
[X,Y,Z] = meshgrid(x,y,z);
W = exp(-(X.^2 + Y.^2 + Z.^2));
figure(1)
    xs = [-.5, 0, .5];
    ys = [];
    zs = ys;
    slice(x,y,z,W,xs,ys,zs)
    axis('equal')
```

This is shown in Figure 16

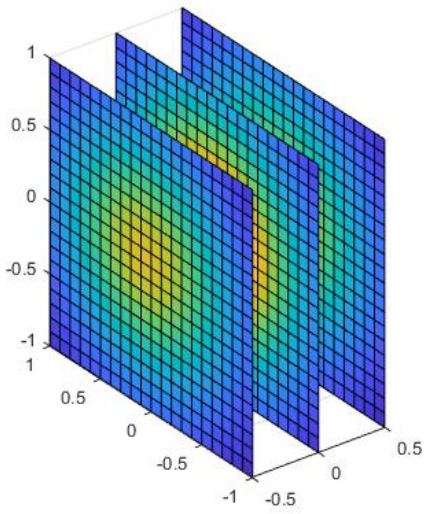

Figure 16: A volume sliced in the x-direction.

In this example we are creating 3 slice planes perpendicular to the x-axis. The locations of the planes are given in the vector `xs`. The `ys` and `zs` vectors are empty because we don't want any slice planes in those directions.

We can also create single slice planes along each axis

```
clear
close all
x = linspace(-1,1,21)';
y = x;
z = x;
[X,Y,Z] = meshgrid(x,y,z);
W = exp(-(X.^2 + Y.^2 + Z.^2));
figure(2)
    slice(x,y,z,W,0,0,0)
    axis('equal')
```

This is shown in Figure 17.

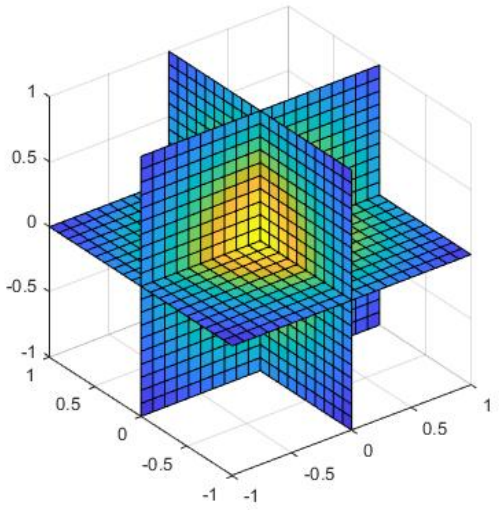

Figure 17: A volume sliced along each axis.

4.2 Isosurfaces

An isosurface is the two-dimensional analog of a contour line. The value of w would not change if you were to walk along an isosurface.

```
clear
close all
x = linspace(-1,1,21)';
y = x;
z = x;
[X,Y,Z] = meshgrid(x,y,z);
W = X.*exp(-(X.^2 + Y.^2 + Z.^2));
figure(3)
     isosurface(x,y,z,W,0.3)
     alpha(0.5)
     isosurface(x,y,z,W,0.2)
     alpha(0.2)
```

This is shown in Figure 18.

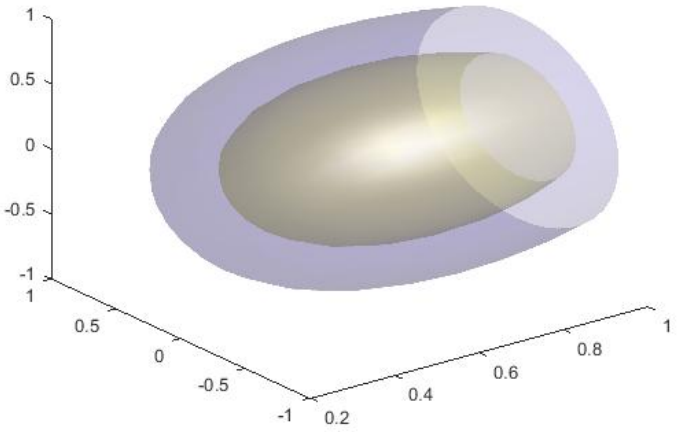

Figure 18: Some simple isosurfaces.

In this case the function is

$$z = xe^{-(x^2+y^2+z^2)}$$

and the isosurfaces corresponding to $w = 0.3$ and $w = 0.2$ are plotted. The function `alpha` is used to change the transparency of each surface.