3D Plotting

Leigh J. Little SUNY Brockport December 3, 2021

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 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 a three dimensional parametric curve $(x(t), y(t), z(t))$ for $t \in [a, b]$.
- 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.

MATLAB can handle all three of these cases, but Python is only capable of easily handling the first two.

2 Functions of the Type $z = f(x, y)$

When plotting a function of this type (say a surface plot), there are two main ways that the data to be plotted can appear

- 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 1). 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.

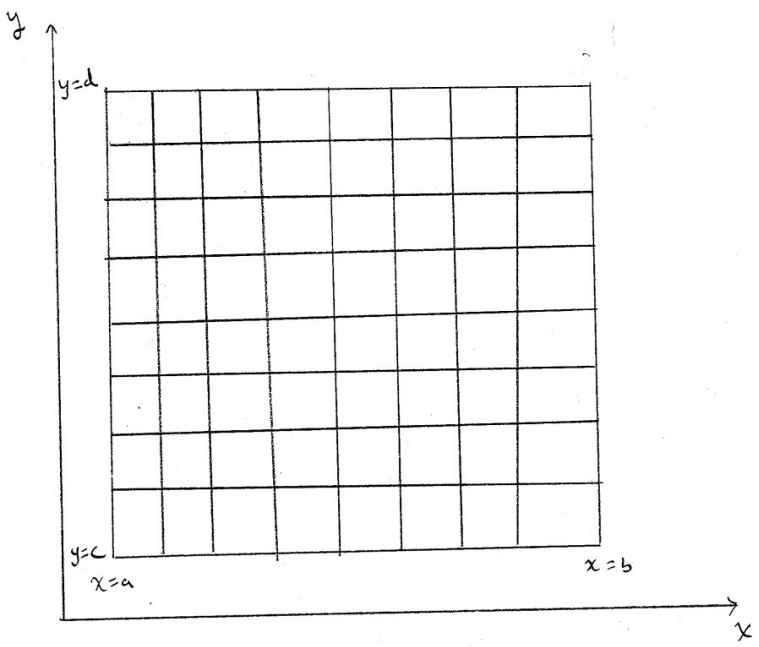

Figure 1: A rectangular region for gridded data.

2.1 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 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 in range(len(x)):
    for j in range(len(y)):
        'z[i,j] = f(x[i],y[j])'
#end of loop
```

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.

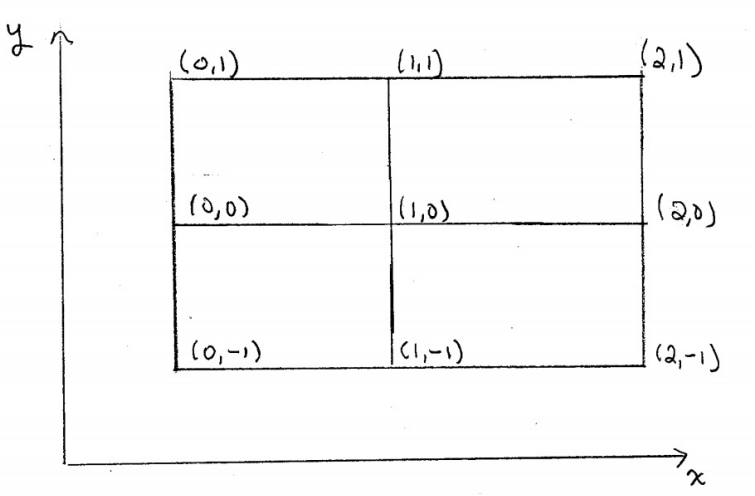

Figure 2: A small gridded section with coordinates.

```
x = np.linspace(0,2,3)
y = np.linspace(-1,1,3)
X,Y = np.meshgrid(x,y)
```

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 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 = np.linspace(-1,1,21)    # Generate 21 points from -1 to 1
y = np.linspace(-1,1,41)    # Generate 41 points from -1 to 1
X,Y = np.meshgrid(x,y)      # Generate mesh grid (note capital X,Y)
Z = np.exp(-(X**2 + Y**2))  # Compute Z on the mesh grid (X,Y)
                            # Note that there are no . operators
```

Different numbers of points have been used in each direction to make the distinction between the x and y axes more clear. The sequence of commands would be essentially the same in MATLAB.

2.1.1 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, use the sequence of commands below:

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d

plt.figure(1)
x = np.linspace(-1,1,21)
y = np.linspace(-1,1,41)
X,Y = np.meshgrid(x,y)
Z = np.exp(-(X**2 + Y**2))

ax = plt.axes(projection='3d')
ax.plot_surface(X,Y,Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```

This is shown in Figure 3.

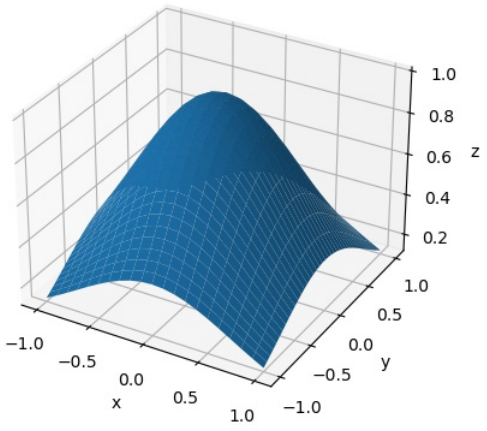

Figure 3: Simple surface plot.

Note that there is a new library (`mplot3d`) that needs to be imported. This library is being imported without a prefix. Whenever you import a library without a prefix, you do not need to precede the command name in the library with any prefix. This is done to avoid having long strings of prefixes in the commands.

Also, the commands look somewhat different than for two dimensional plotting. The command

```
ax = plt.axes(projection='3d')
```

is necessary to create a set of three dimensional axes and all subsequent commands need the prefix of `ax` to let Python know which set of axes the command should apply to (in the event you are creating several plots at once).

2.1.2 Colormaps

Colormaps are a way to enhance a surface plot by coloring the planes according to some value. Most often, this is the z value. In the example above, change the `plot_surface` command to

```
ax.plot_surface(X,Y,Z, cmap=cm.cool)
```

and add the import command

```
from matplotlib import cm
```

at the top. Running the script gives the figure shown in Figure 4. The color of the planes change as the

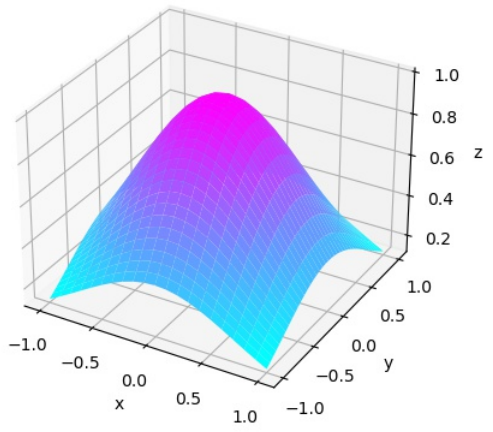

Figure 4: Surface plot with a colormap.

elevation z changes. This makes it easier to distinguish changes in z. You can find a list of all the predefined colormaps in Python at

```
https://matplotlib.org/stable/tutorials/colors/colormaps.html
```

2.1.3 Wireframe Plots

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

```
ax.plot_wireframe(X,Y,Z)
```

ans run the script again. The result is shown in Figure 5. Note that the wireframe image allows you to see through the plot due to the lack of coloring.

2.1.4 Contour Plots

Contour plots are frequently used in topology. They allow for an easy to interpret two dimensional visualization of a three dimensional surface. A contour plot consists of a series of lines. The value of z is constant along each of these lines. If you were to walk along a contour line, your elevation would not change.

Because a contour plot is a two dimensional plot, the commands look different than in the previous examples. The sequence below will create a contour map of the function z with 10 evenly spaced contours.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d
```

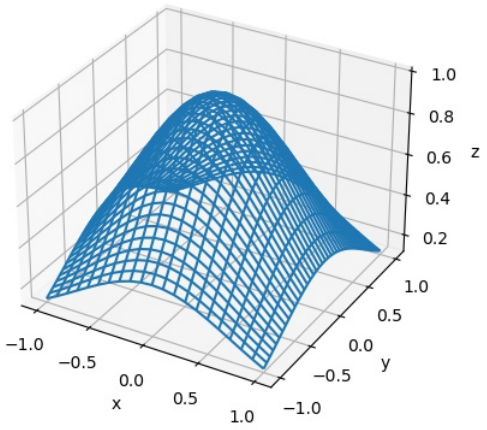

Figure 5: Wireframe plot.

```
plt.figure(4)
x = np.linspace(-1,1,21)
y = np.linspace(-1,1,41)
X,Y = np.meshgrid(x,y)
Z = np.exp(-(X**2 + Y**2))
plt.contour(X,Y,Z,10)          # No ax here because the plot is 2d
plt.axis('equal')
plt.show()
```

This is shown in Figure 6. You can also send in a list object containing the specific contour levels desired

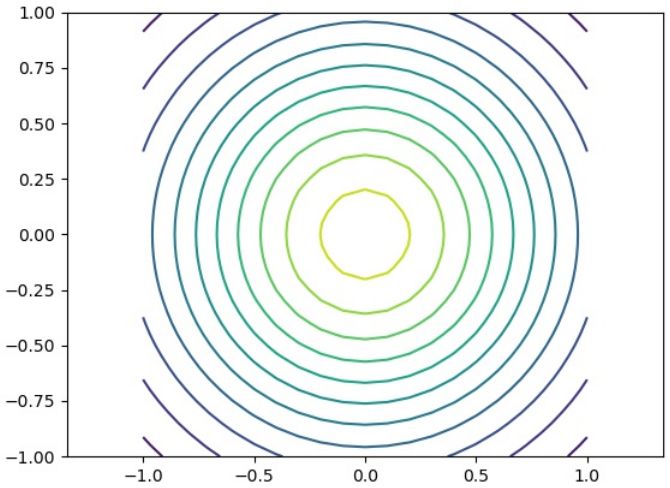

Figure 6: Contour plot with 10 evenly spaced contours.

rather than a fixed number of contours.

2.1.5 Three Dimensional Contour Plots

You can also create three dimensional contour plots.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d

plt.figure(5)
x = np.linspace(-1,1,21)
y = np.linspace(-1,1,41)
X,Y = np.meshgrid(x,y)
Z = np.exp(-(X**2 + Y**2))
ax = plt.axes(projection='3d')
ax.contour(X,Y,Z,10)                # Plot 3D version on ax
plt.show()
```

This is shown in Figure 7.

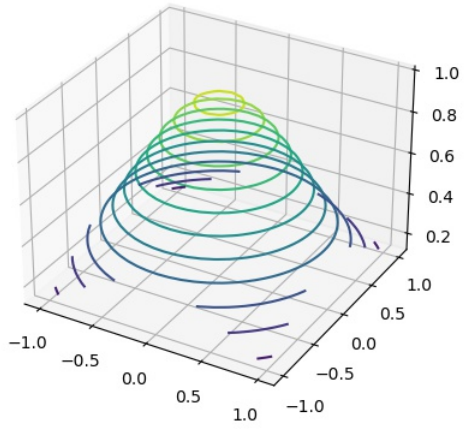

Figure 7: Three dimensional contour map with 10 contours.

2.2 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. This capability is in Python, but it is not ideal.

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

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

at this set of points. Note that the random number library we used in a previous assignment is also part of the `numpy` library.

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)*np.random.random()
```

The sequence of commands below will generate such a triangulated surface plot in addition to the raw data point themselves.

```
a = -1
b = 1
n = 200
x = a + (b-a)*np.random.random(n)
y = a + (b-a)*np.random.random(n)
z = np.exp(-(x**2 + y**2))

plt.figure(7)
ax = plt.axes(projection='3d')
ax.scatter3D(x,y,z)

plt.figure(8)
ax = plt.axes(projection='3d')
ax.plot_trisurf(x, y, z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```

These are shown in Figures 8 and 9.

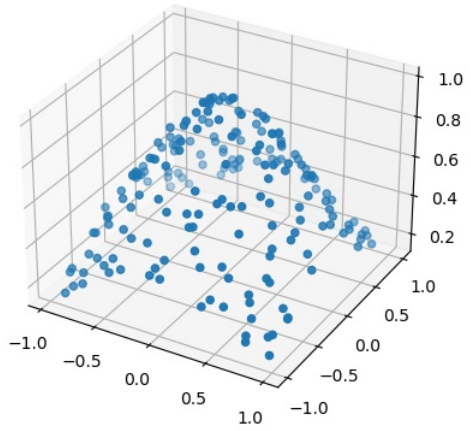

Figure 8: Scatter plot of non-gridded data.

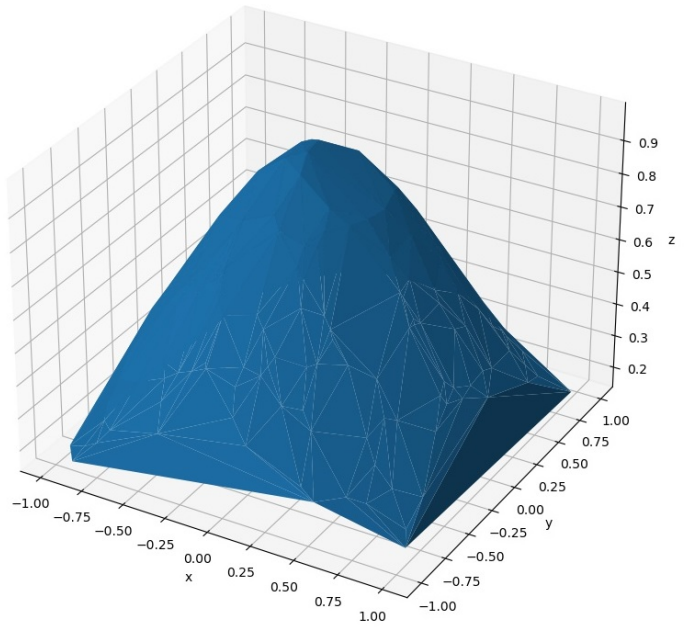

Figure 9: Triangulated surface plot of non-gridded data.

3 Parametric Curves

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.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits import mplot3d

plt.figure(6)
R = 2
S = 1
t = np.linspace(0,6*np.pi,201)
x = R*np.cos(t)
y = R*np.sin(t)
z = S*t
ax = plt.axes(projection='3d')
ax.plot3D(x,y,z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```

This is shown in Figure 10.

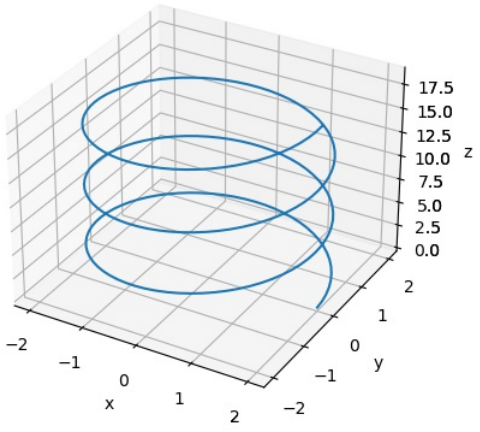

Figure 10: Circular helix.

4 Volumetric Visualization

Volumetric visualization in Python is being developed, but is not ready for prime time (at least for free libraries, there are probably commercial libraries you can pay for that do this). MATLAB can do this quite well.

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 11 In this case we are creating 3 slice planes perpendicular to the x-axis.

We can also create single slice planes along each axis

```
clear
close all
x = linspace(-1,1,21)';
```

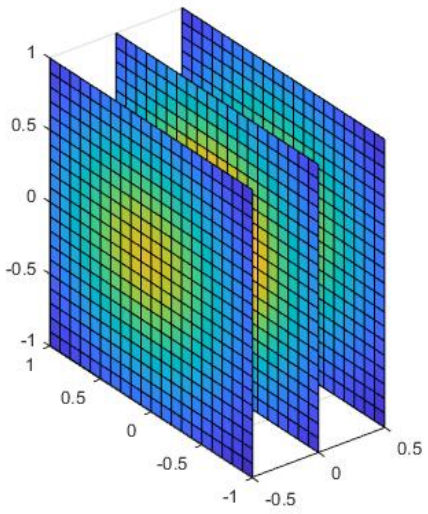

Figure 11: A volume sliced in the x-direction.

```
y = x;
z = x;
[X,Y,Z] = meshgrid(x,y,z);
w = exp(-(X.^2 + Y.^2 + Z.^2));
figure(2)
    xs = [-.5, 0, .5];
    ys = [];
    zs = ys;
    slice(x,y,z,w,0,0,0)
    axis('equal')
```

This is shown in Figure 12.

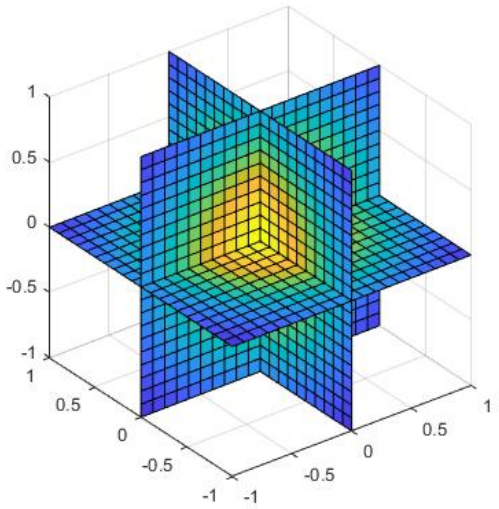

Figure 12: A volume sliced along each axis.

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 13. In this case the function is

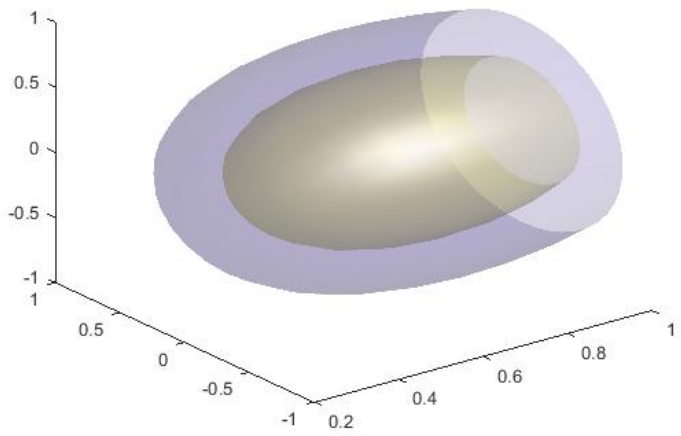

Figure 13: Some simple isosurfaces.

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

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.