Simple Plots in Python

Leigh J. Little SUNY Brockport November 13, 2020

Contents

1 Introduction

Python has no native plotting capabilities. This gives rise to a large number of third party libraries that provide these features. These libraries are varying in quality. Historically the `pyplot` library has been the standard, but this has been superseded. The most frequently used library for plotting is `matplotlib`. This library incorporates features of the original `pyplot`, but is more robust and has better capabilities. The library more or less provides the same plotting ability as MATLAB. The main change is in the syntax of the commands.

2 Examples

2.1 Example 1 - Simple x-y plot

The code below will generate a simple plot of x versus y.

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5]  # Note that x and y are standard lists
y = [5,4,3,2,1]

plt.figure(1)
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simple plot')
plt.show()
plt.savefig('ex1.png')
```

Some comments on the above example:

1) The entire `matplotlib` module is not being pulled into the workspace (just the `pyplot` submodule is being imported).

2) Every plot command gets preceded by `plt.`

3) The `plt.show()` command is used to make the figure visible.

4) The `plt.savefig('ex1.png')` command will save the plot as a .png file in the working directory.

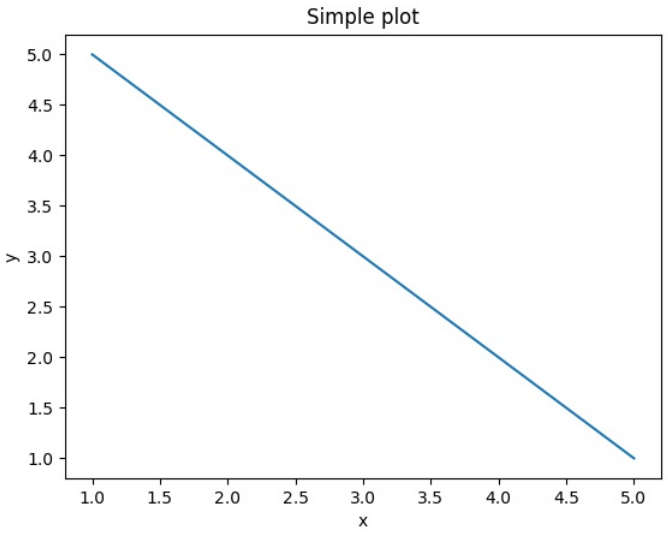

Figure 1: Simple x-y plot example.

2.2 Example 2 - Simple plot with a legend

The code below will generate several plots in the same frame with a legend

```
import matplotlib.pyplot as plt

x = [1,2,3,4,5]  # Note that x, y and z are standard lists
y = [5,4,3,2,1]
z = [3,3,3,3,3]

plt.figure(2)
plt.plot(x,y,label='y')    # Legend labels specified when plotting
plt.plot(x,z,label='z')
plt.xlabel('x')
legend()                   # Displays all legend items
plt.title('Simple plot with legend')
plt.show()
plt.savefig('ex2.png')
```

The `plot` command has options similar to the MATLAB `plot` command. See the documentation at

`https://matplotlib.org/tutorials/introductory/pyplot.html`

for more examples on how to use the command.

3 3D Graphics in MATLAB

This section will discuss some basic 3-dimensional graphs in MATLAB. This is a very broad topic and is complicated by the great diversity of plots that can be generated in 3-dimeionsional space. We will focus on the most common type of plots that are needed.

3.1 Surface Plots

Consider the function

$$z(x, y) = \sin\left(\frac{x}{2}\right)\cos\left(\frac{y}{2}\right).$$

In this case, z has two independent (or input) variables, x and y.

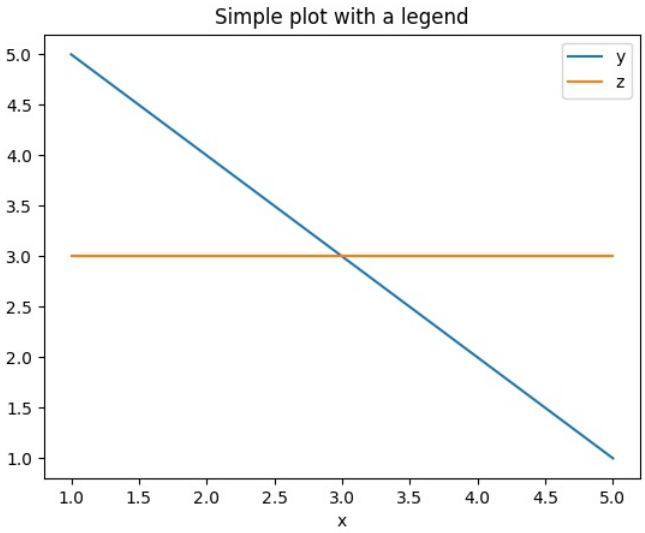

Figure 2: Simple plot with a legend.

In principle, to plot z, we generalize the idea when plotting a standard $y = f(x)$ type function. In this case, we input values for x and y and compute z. We generate a table of values, then plot these points in 3-dimensional space. We then join these points in some manner.

There is a subtle difference in what we want to do here and what we would do for a standard $y = f(x)$ function. In the $y = f(x)$ case, the only choice for the domain of the function is a line segment. It may be the interval [0,2] or [-3,3] or some other interval, but all of these are variations of the same basic shape.

In the case of a $z = f(x, y)$ function, we have an infinite number of choices for the domain of the function. It can be a simple square or rectangle, but it could also be the circular region defined by $x^2 + y^2 \leq 4$. It could be the triangular region bounded by the lines $x = 0, y = 0, y = x - 1$. It could also be the region bounded by an arbitrary blob shaped curve.

All of these cases can be handled, but plotting a function like z over anything but a square or rectangle requires some additional details that we won't look at for the time being. For right now, assume that we want to plot z over the rectangular domain $x \in [-\pi, \pi], y \in [-2\pi, 2\pi]$. We could do something like the following to generate the values of z to be plotted:

```
x = linspace(-pi,pi,21)';
y = linspace(-2*pi,2*pi,41)';
for i = 1:length(x)
    for j = 1:length(y)
        z(i,j) = sin(x(i)/2)*cos(y(j)/2);
     end
end
```

However, this is not how this would normally be done in MATLAB. Instead, the idea of a *meshgrid* is the more common approach.

To see what a meshgrid is, it helps to look at a much smaller case. Suppose we have the set of x-coordinates $x = \{0, 1, 2\}$ and the set of y-coordinates $y = \{-1, 0, 1\}$. This gives a total of 9 coordinates in the x-y plane for which the value of z needs to be computed. These coordinates would consist of all combinations of the individual x and y coordinates: $(0, -1), (1, -1), (2, -1), \ldots$.

The MATLAB `meshgrid` command will quickly generate computational grids that will permit the value of z to be computed in a single command.

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

```
     0     1     2
     0     1     2
     0     1     2
Y =
    -1    -1    -1
     0     0     0
     1     1     1
```

The outputs `X` and `Y` consist of matrices. If you mentally superimpose these matrices on top of one another, you can see that this will form a grid of coordinates. We can then compute the value of z by doing

```
>> z = sin(X/2).*cos(Y/2)
z =
    0    0.4207    0.7385
    0    0.4794    0.8415
    0    0.4207    0.7385
```

Note that we have used the meshgrid variables `X` and `Y` instead of the arrays `x` and `y`.

Now we can easily generate the data to be plotted:

```
>> x = linspace(-pi,pi,21)';
>> y = linspace(-2*pi,2*pi,41)';
>> [X,Y] = meshgrid(x,y)
>> z = sin(X/2).*cos(Y/2);
```

To create a surface plot of z, do the following:

```
>> figure(1)
>> surf(x,y,z)    ; note x and y, not X and Y.
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
```

The resulting plot is shown in Figure 3. In the event that you have a very dense grid of coordinates, you

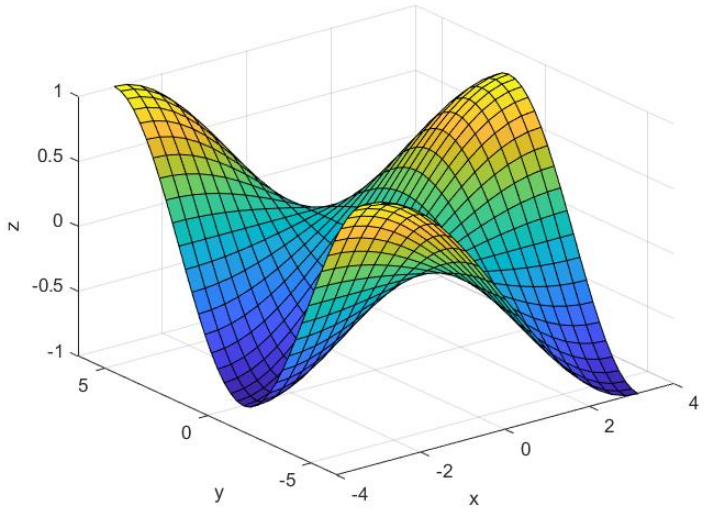

Figure 3: Surface plot of z.

will observe that the grid lines on the surface dominate the image. To fix this, you can issue one of two commands:

```
>> shading flat    ; removes grid lines
>> shading interp  ; removes grid lines and smooths the colors in the patches
```

3.2 Contour Plots

A contour plot is another way to view a function of the form $z = f(x, y)$. A topographical map is an example of a contour plot. In this type of plot, a series of lines are drawn. The value of z is constant along these lines. To generate a contour plot of z, you can do the following:

```
>> (generate z as before)
>> contour(x,y,z)
```

This will draw 10 equally spaced contour lines. The resulting figure is shown in Figure 4. If you want more

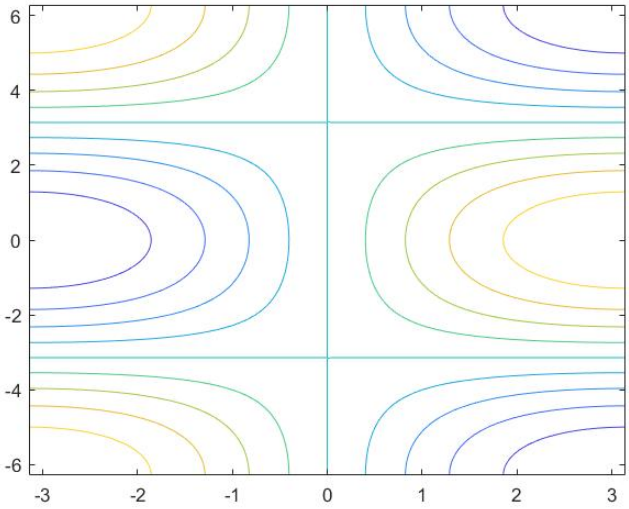

Figure 4: Contour plot of z with 10 contour lines

contours, you can send in a fourth argument with the number of contours desired:

```
>> contour(x,y,z,20)
```

This is shown in Figure 5. You can also send in a vector with the desired contour levels if you want to see a specific set of contours.

```
>> contour(x,y,z,[0 0.2 0.4 0.6 0.8 1])
```

This is shown in Figure 6.

3.3 3-Dimensional Contour Plots

A 3-dimensional contour plot is a combination of a surface and contour plot. In this case, the contours are drawn at the physical level of the contour. A relief map is an example of a 3-dimensional contour map. To generate a plot of this type in MATLAB, you can do

```
>> (generate z as before)
>> contour3(x,y,z,20)
```

This is shown in Figure 7.

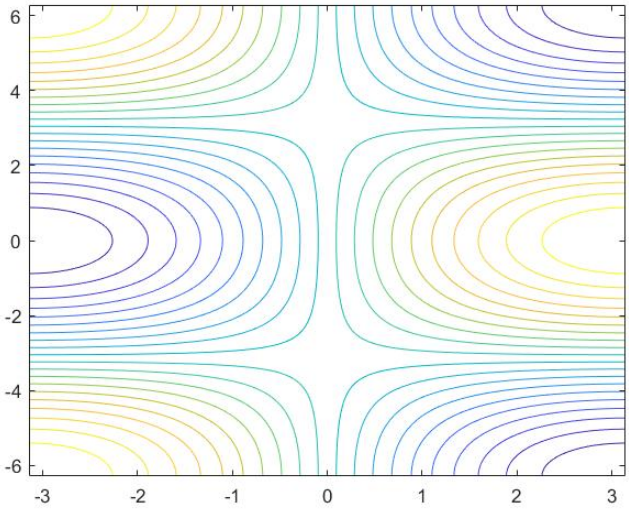

Figure 5: Contour plot of z with 20 contour lines

4 The numpy Library

The `numpy` library is used to make solving mathematical problems in Python easier. You should use this library instead of the `math` library from now on. `numpy` contains all the features of the `math` library, but also introduces new features. One of the most important is a new data structure called an `array` that behaves more like an array/matrix in MATLAB. In fact, augmenting Python with the `numpy` and `matplotlib` libraries results in an environment very similar to MATLAB. This will be illustrated using some 3D graphics capabilities.

4.1 Surface Plot in Python

Suppose we wanted to create a surface plot of the function

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

over some rectangular region of the x-y plane.

As in MATLAB, we will use a meshgrid to evaluate z, Note that the Python commands to do this are nearly the same as in MATLAB.

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

plt.figure(3)
x = np.linspace(-1,1,51)    # Generate 51 points from -1 to 1
y = np.linspace(-2,2,101)   # Generate 101 points from -2 to 2
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

ax = plt.axes(projection='3d')  # Generate 3D axes
ax.plot_surface(X,Y,Z)      # Generate surface plot of Z on ax
plt.show()
```

In the above example, `x`, `y`, `X` and `Y` are `numpy` arrays, not lists.

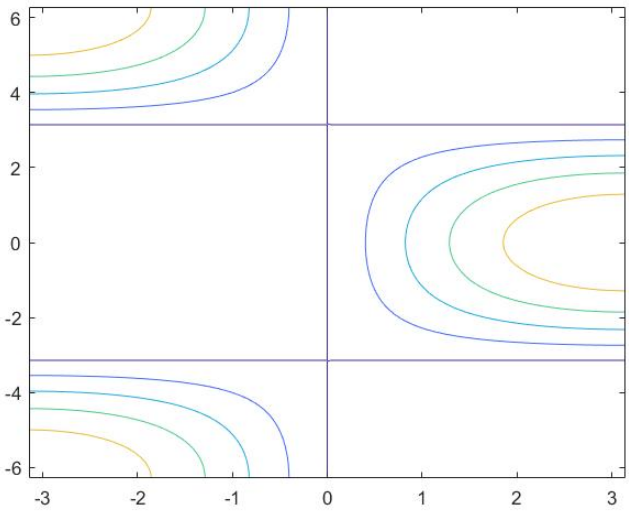

Figure 6: Contour plot of z with specific contour lines

4.2 Example 4 - Contour Map

Contour maps can also be generated. You can do a 2D contour map

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

plt.figure(4)
x = np.linspace(-1,1,51)
y = np.linspace(-2,2,101)
X,Y = np.meshgrid(x,y)
Z = np.exp(-(X**2 + Y**2))
plt.contour(X,Y,Z)          # No ax here because the plot is 2d
plt.show()
```

You can also do 3D contour maps.

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

plt.figure(5)
x = np.linspace(-1,1,51)
y = np.linspace(-2,2,101)
X,Y = np.meshgrid(x,y)
Z = np.exp(-(X**2 + Y**2))
ax = plt.axes(projection='3d')
ax.contour(X,Y,Z)                # Plot 3D version on ax
plt.show()
```

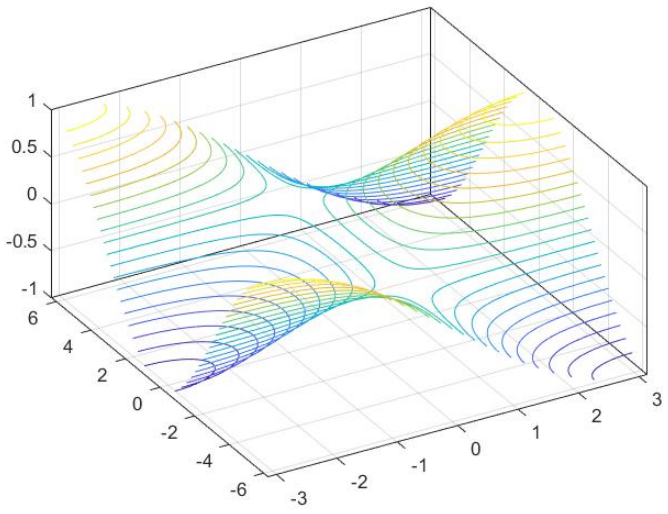

Figure 7: 3-dimensional contour plot of z with 20 contour lines.

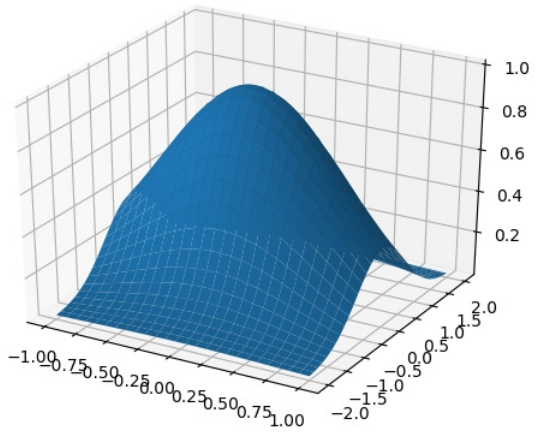

Figure 8: Surface plot of z.

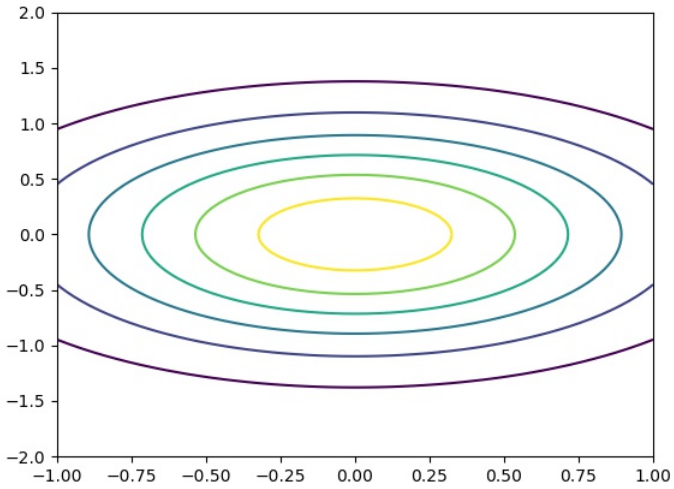

Figure 9: 2D contour map of z.

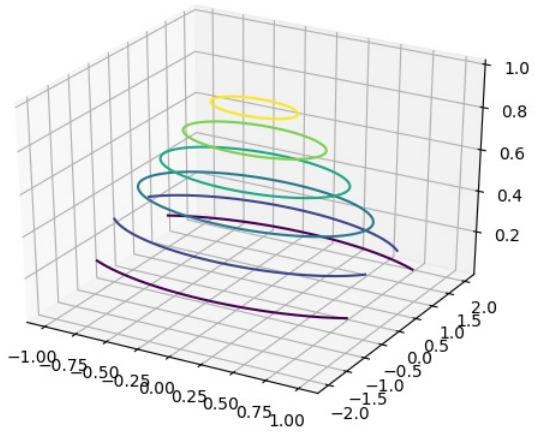

Figure 10: 3D contour map of z.