BYU-MCL Boot Camp
Matplotlib Labs

1 Introduction

This lab is to act as an introduction to the matplotlib plotting environment and provide you the resources and ability to create plots and graphs that are suitable to be put into peer-reviewed journals. Matplotlib was originally established by John Hunter in order to, as its homesite says, "...produce publication quality figures in a variety of hardcopy formats and interactive environments across pltatforms." It has perhaps become the single most used Python pakcage for 2D-graphics. I would suggest referencing these three sites for good information on matplotlib (although there are many others).

- http://matplotlib.org/
- http://scipy-lectures.github.io/intro/matplotlib/matplotlib.html
- http://jkitchin.github.io/pycse/pycse.html#sec-11

*Note: I imported matplotlib.pyplot as plt so plt is my abbreviation for matplotlib similar to how I use np for numpy.

2 Basic Plots

Many of you are already familiar with some of the basic functionality of matplotlib. This will be a brief introduction for those of you who are not. Some of the basic commands are listed below:

- plt.plot(x,y): Plots your x and y data
- plt.hist(x): Creates a histogram for the data you input
- plt.title('title'): Gives a title to your graphic
- plt.x(y)label('label'): Labels your x(or y) axis.
- plt.show(): Shows the image that you have built previous to this command

Lets begin by plotting $\cos(x)$ from 0 to 2π. In the code below we will define the data and then plot the output.

```
x = np.linspace(0, 2*np.pi, 100)
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

You can see from the above that these commands create the plot that we wanted. Now let's do that again, but we will give our plot a title and label the axes.

```
x = np.linspace(0, 2*np.pi, 100)
y = np.cos(x)
plt.plot(x, y)
plt.title('This is the graph of cos(x)')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```

This is a very basic graph and you can do similar things with histograms.

Exercise 1. Now that you know how to use the basic commands of matplotlib. Create a graph of $\sin(x)$ on $[-2\pi, 2\pi]$. Title this plot as "My Sin(x) graph" and label the x and y axes.

Exercise 2. Now draw 1000 random elements from the normal distribution, $N(0, 1)$. We will refer to your random sample as D. Create a histogram of D, D^2, and $\sqrt{(|D|)}$. Title each of these graphs and label your axes appropriately.

3 More Options for Basic Plot

The options that are in matplotlib are almost innumerable and thus it is not pratical for me to go over all of them, but we will cover several that are important.

Colors: Look in the matplotlib and experiment with multiple colors.

```
plt.plot(x, y, color="color")
```

Line Width: Try linewidth=2.5 versus the default.

```
plt.plot(x, y, linewidth=2.5)
```

Line Style: Try linestyle='–'

```
plt.plot(x, y, linestyle='--')
```

Label/Legend: Labels your line and creates legend. Try the following:

```
plt.plot(x, y, label='func1')
plt.legend('best')
```

Exercise 3. Now that you are familiar with some of the possible options of matplotlib, we are going to create a slightly more complicated graph. Graph $\sin(x)$ and $\cos(x)$ on $[-2\pi, \pi]$. Label the $\sin(x)$ graph as 'sin(x)' and $\cos(x)$ graph as 'cos(x)' and make them two different colors and line styles. Increase their line width to 3 and place a legend in the upper left corner.

4 Limits, Spines and Ticks

You can change the limits on your graph. There are times when you would like to see a little beyond where the points are (Not to mention I think it looks better when used correctly). Changing the limits on the graph is quite intuitive. Use plt.xlim and plt.ylim (Who would've guessed...) Try the following with some x and y:

```
plt.plot(x, y)
 # If x.min() and y.min() are negative then * 1.1 will make them
 # more negative and expand your bounds.
plt.xlim(x.min() * 1.1, x.max() * 1.1)
plt.ylim(y.min() * 1.1, y.max() * 1.1)
```

Matplotlib calls the lines that box the graph 'spines.' To move the spines to the middle to form an x and y axis use the following commands:

```
plt.plot(x, y)
ax = plt.gca() # Stands for get current axis
ax.spines['right'].set_color('none') # Only need two so make this one
    invisible
ax.spines['top'].set_color('none') # ^^
ax.spines['left'].set_position(('data', 0))
ax.spines['bottom'].set_position(('data', 0))
```

Additionally you can add ticks where you want them and label them how you'd like. plt.xticks and plt.yticks move and label your x and y ticks. The format is the following:

```
plt.x(y)ticks([list where ticks should be], [labels that correspond to
    previous list])
```

Exercise 4. Practice specifying appropriate limits and ticks by adding appropriate limits and ticks (and labeling them) at $-2\pi, -\pi, 0, \pi, 2\pi$ to exercise 3. Create two graphs. One should have normal spines and the other should use the spines as an xy axis.

5 Subplots

Subplotting is a way to place multiple plots on a single figure. There are several possibilities for setting up subplots (and once again the command is obvious... plt.subplot or plt.subplots). The specifications for subplots can be found more in depth on the matplotlib website and the scipy tutorial website. Please read through some of this to understand the possibilities of subplots. Below is an example.

```
x = np.linspace(0, 2*np.pi, 25)
y1 = np.cos(x)
y2 = x * np.cos(x)
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y1)
ax1.set_title(r'$\cos(x)$')
ax2.plot(x, y2)
ax2.set_title(r'$x \cos(x)$')
plt.show()
```

Exercise 5. Using the Chebyshev Polynomial creating function that you wrote for the curve fitting homework create a figure with the first 6 Chebyshev polynomials on 1 plot. This figure should be well labeled and have a legend. Then create a figure with 6 subplots where each subplot has one of the Chebyshev Polynomials. The subplots should use the same x and y axis and make sure to distinguish which polynomial is which.

6 3D Plotting

So far we have only worked with plotting in 2 dimensions. It is very possible that you will need to create 3 dimensional plots, so we will discuss how to create them.

Typically to plot a function, f, of one variable you use an array of values and plot them against $f(x)$. For a 2D function (3D plot) you need a function of two variables. Then you need an array of values for each of your variables. Then you have to plot the function for every pair of values (the Cartesian Product of the two arrays). This is done in the following fashion.

```
from mpl_toolkits.mplot3d import Axes3D
func = lambda x,y: np.sqrt(x**2 + y**2) # Define a func
x = np.linspace(0, 5, 25) # Create an array of xs
y = np.linspace(0, 5, 25) # Create an array of ys
Y, X = np.meshgrid(x, y) # Create the proper matrices (See meshgrid
    doc)
fig = plt.figure()
ax = Axes3D(fig) # Define the 3D axes
ax.plot_surface(X, Y, func(X, Y)) # Plot the 3d graph
```

```
ax.set_title('Example of 3D plot')
plt.show()
```

Exercise 6. Plot the 3D function $z = \cos(x)^2 + \sin(y)^2$ on the rectangle $x \in [0, 2\pi], y \in [0, 2\pi]$. Label and title the graph appropriately.

7 Saving Figures

You can manually save Python plots when they are displayed following the command plt.show(), but there is an easier way. Matplotlib has a command plt.savefig('filename') that you can type before or instead of plt.show(). One of the more important specifications that you can give to plt.savefig is the number of dots per inch.

Exercise 7. In the matplotlib documentation figure out how to change the dots per inch (dpi) on a graph. Then save one of the figures from a previous exercise with the plt.savefig command at 10 dpi and at 500 dpi. Note that for posters and other presentations you need enough dpi for the picture not to be blurry.

8 More Exercise

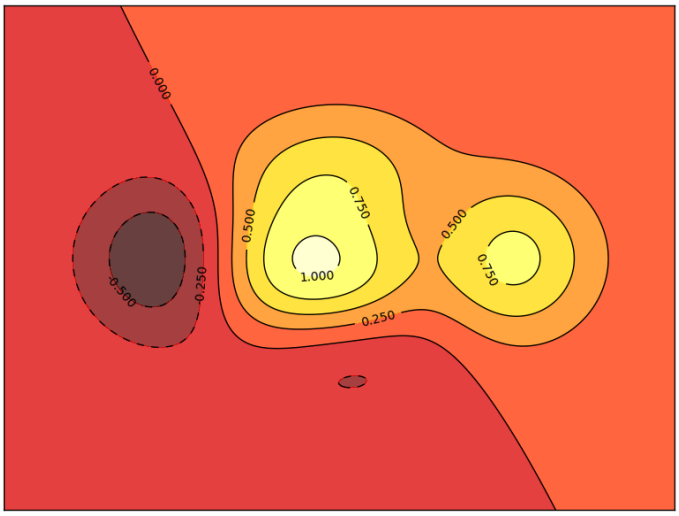
Exercise 8. Mimic the two following graphs. Use the matplotlib documentation to find and understand some of the new options required to successfully plot these graphs using the following code as a start.

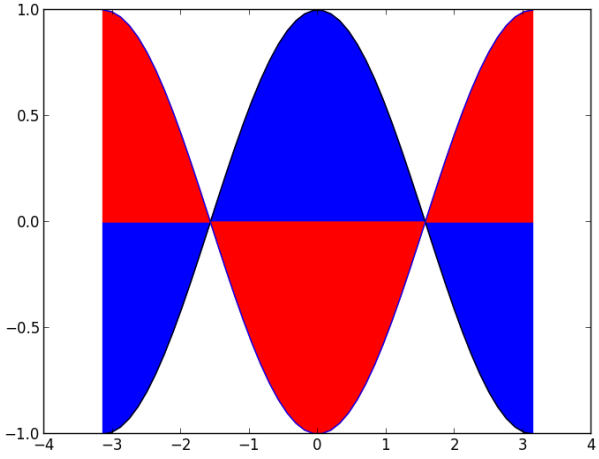
```
# problem 1 (Hint use plt.contour())
def f(x,y):
    return (1 - x / 2 + x**5 + y**3) * np.exp(-x**2 -y**2)

n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X,Y = np.meshgrid(x, y)

plt.axes([0.025, 0.025, 0.95, 0.95])

# problem 2
x = np.linspace(-np.pi, np.pi, 50)
y1 = np.cos(x)
y2 = -np.cos(x)
```


0.000
0.500
0.750
1.000
0.500
0.750
-0.500
-0.250
0.250


1.0
0.5
0.0
−0.5
−1.0
−4
−3
−2
−1
0
1
2
3
4