Plotting

Carver J. Bierson

June 30, 2017

The standard tool for plotting in python is matplotlib. matplotlib is a powerful tool and we will only cover some what it has to offer in this class. There are two main ways of using matplotlib. One is more of a function based method that I will show while the other is more object based. Both are acceptable on assignments.

1 Making basic plots

To start lets plot a basic sin curve

```
import numpy as np
import matplotlib.pyplot as plt

#make some data
x=np.linspace(0,2*np.pi)
y=np.sin(x)

# Create figure
plt.figure()

#plot data in figure
plt.plot(x,y)

#show all figures
plt.show()
```

If you want to set the x or y axis limits you can use *plt.xlim([xmin,xmax])* where *xmin* and *xmax* are the values for the minimum and maximum x values. If you want the axes to automatically shrink to your data limits you can use *plt.axis('tight')*.

Often we will have multiple data sets we want to plot against one another. You should also add axis labels and a legend.

```
import numpy as np
import matplotlib.pyplot as plt

#make some data
x=np.linspace(0,2*np.pi)
ysin=np.sin(x)
ycos=np.cos(x)

# Create figure
plt.figure()

#plot data in figure
plt.plot(x,ycos,'--k', label='cos(x)')
```

```
plt.plot(x,ysin, '.r', label='sin(x)')

plt.xlabel('x (radians)')
plt.ylabel('y')

plt.legend()

#show all figures
plt.show()
```

Instead of plotting our data on top of one another we could plot it on two axes in the same figure.

```
import numpy as np
import matplotlib.pyplot as plt

#make some data
x=np.linspace(0,2*np.pi)
ysin=np.sin(x)
ycos=np.cos(x)

# Create figure
plt.figure()


# The subplot syntax is: # of columns, # of rows, active subaxis
plt.subplot(2, 1, 1)
plt.title('cos(x)')

plt.plot(x,ycos,'--k', label='cos(x)')
plt.ylabel('y')

plt.subplot(2, 1, 2) #make 2nd pannel active

plt.title('sin(x)')

plt.plot(x,ysin, '.r', label='sin(x)')

plt.xlabel('x (radians)')
plt.ylabel('y')


#show all figures
plt.show()
```

You can set the axis range to custom values using the *plt.xlim* and *plt.ylim* functions. If you want to add text to a plot use *plt.text(x,y,'string')*. It usually takes some trial and error to get the text where you want on a figure. There are other plotting commands for a plot with errorbars, a pure scatter plot, histogram, etc. All of these though have basically the same syntax as the basic plot command. See the documentation for specifics on any particular function.

2 2D and 3D data

Lets say we have a 2D field of data (like temperature across a surface). There are several ways of plotting this data. The simplest is to make a contour plot.

```
import numpy as np
import matplotlib.pyplot as plt

```

```
# Make 1D grid vectors
x=np.linspace(-2*np.pi,2*np.pi)
y=np.linspace(-2*np.pi,2*np.pi)

# Make these 2D grid arrays
xx,yy=np.meshgrid(x,y)

#Create some data to plot
r = np.sqrt(xx**2 + yy**2)
phi = np.arccos(x/r)

z1 = np.sin(2*r) / r

# Create figure
plt.figure()

#plot data in figure
plt.contour(xx,yy,z1)

# Make x and y spacing equal (gets rid of stretching)
plt.gca().set_aspect('equal')

# Label axes
plt.xlabel('x')
plt.ylabel('y')

# Make another figure with custom contours
plt.figure()

plt.contour(xx, yy, z1, np.arange(-0.4, 2.0, 0.1))


# Make a third figure with labeled contours
plt.figure()

cs = plt.contour(xx, yy, z1, np.arange(-0.4, 2.0, 0.4))

plt.clabel(cs)


#show all figures
plt.show()
```

But what if you don't just want contours, you want some color in your life. There are a few functions to choose from. The first is filled contours, *contourf*. It has the same syntax a *contour* so I won't show an example of that one (try it on your own). My perfered 2D plotting function is *pcolor* because I think it is the most flexable. That said for some applications it may be better to use *imshow*.

```
import numpy as np
import matplotlib.pyplot as plt

# Make 1D grid vectors
x=np.linspace(-2*np.pi,2*np.pi)
y=np.linspace(-2*np.pi,2*np.pi)

# Make these 2D grid arrays
xx,yy=np.meshgrid(x,y)

#Create some data to plot
r = np.sqrt(xx**2 + yy**2)
```

```
phi = np.arccos(x/r)

z1 = np.sin(2*r) / r

# Create figure
plt.figure()

#plot data in figure
plt.contour(xx,yy,z1)

# Make x and y spacing equal (gets rid of stretching)
plt.gca().set_aspect('equal')

# Label axes
plt.xlabel('x')
plt.ylabel('y')

#Plot heatmap with pcolor
plt.pcolor(xx,yy z1)


plt.colorbar() #show the colorbar


#show all figures
plt.show()
```

You may also want to change the colormap. For a list of built in colormaps in python see `https://matplotlib.org/examples/color/colormaps_reference.html`. To change the color map you are using you can either pass it as an optional argument for pcolor or use

```
plt.set_cmap('bwr')
```

2.1 3D plots

3D plots can make great visuals but can also be more confusing than their 2D counterparts so use them with caution. You can do line and scatter plots in 3D but I will show surface plots. If you want to see all of the options look at `https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html`. I will also note that for 3D you must (to my knowlage at least) use the object oriented syntax of pyplot (shown below).

```
import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D # This gives us access to
    the 3D plots

# Make 1D grid vectors
x=np.linspace(-2*np.pi,2*np.pi,100)
y=np.linspace(-2*np.pi,2*np.pi,100)

# Make these 2D grid arrays
xx,yy=np.meshgrid(x,y)

#Create some data to plot
r = np.sqrt(xx**2 + yy**2)
phi = np.arccos(x/r)

```

```
z1 = np.sin(2*r) / r

# Create figure
plt.figure()
ax=plt.gcf().add_subplot(111, projection='3d')

#plot data in figure
ax.plot_surface(xx,yy,z1)

# Make x and y spacing equal (gets rid of stretching)

# Label axes
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')



plt.colorbar() #show the colorbar


#show all figures
plt.show()
```

3 Making Simple Movies

In this class we will be looking at lots of systems that evolve with time and it is nice to be able to plot that evolution. As and example lets plot a sin wave with a constant phase speed.

```
import numpy as np
import matplotlib.pyplot as plt


k=1.0 # Define wavenumber
w=1.0 # Define angular frequency
tmax=10 # end time

dt=0.1 # timestep

x=np.linspace(-2*np.pi, 2*np.pi,500) # x-grid

plt.figure()

t=0 # start time
while t<tmax:
    plt.cla() # clear the last figure
    plt.plot(x, np.sin(k*x+w*t)) #plot wave equation

    plt.pause(0.05) # update plot and wait 0.05 sec before moving
    on
    t+=dt # increment time
```

Movies like this can also be save by making a list of figure frames. There animation objects can also be saved. They also have of the advantage that they will repeat continuously.

```

import numpy as np
import matplotlib.pyplot as plt
```

```
4
5  import matplotlib.animation as animation # load animation module
6
7
8  k=1.0
9  w=1.0
10 tmax=10
11
12 dt=0.1
13
14 x=np.linspace(-2*np.pi, 2*np.pi,500)
15
16 fig1=plt.figure()
17 plt.ylim([-1,1])
18 plt.xlim([-2*np.pi, 2*np.pi])
19 #plt.show()
20
21 ims=[] #make empty list for frames
22 t=0
23 while t<tmax:
24
25     #plt.plot(x, np.sin(k*x+w*t))
26     ims.append((plt.plot(x, np.sin(k*x+w*t)))) #add frame to list
27     plt.cla() # clear figure
28
29     plt.pause(0.05) # not really needed here
30     t+=dt
31
32
33
34 im_ani = animation.ArtistAnimation(fig1, ims, interval=50,
       repeat_delay=1000,
35                                    blit=True) #make animation
36
37 plt.show() # show animation
38
39 #im_ani.save('wave.mp4') #save animation (does require an encoder
       which your computer probably already has)
```

4 In Class Problem:

Potential fields (electrostatic, gravitational, or otherwise) are great because they are scalar fields that can be linearly summed. Lets plot the potential filed of the Earth - Moon system assuming the Earth and Moon are both point masses.

The gravitational potential of a point mass is

$$V = -\frac{GM}{r} \tag{1}$$

Some values you will need are $M_{Earth} = 6 \times 10^{24}$ kg, $M_{Moon} = 7 \times 10^{22}$ kg, and the distance between the Earth and Moon is 3.8×10^8 m. *Hint:* First plot the potential for the Earth, then the moon, then their sum.

Can you identify the 5 Lagrange points from your figure?

Bonus: Plot the potential as the moon orbits around the Earth (For now assume a circular orbit).