Python Project 1

David Schmidt

February 25, 2018

1 General Comments

Two files will be referenced in this report. Both have very similar code but achieve different results. I will make sure to comment when the two codes are different and show both.

2 Housekeeping

```
11 import numpy as np
12 import matplotlib.pyplot as plt
13 import matplotlib as mpl
14 import matplotlib.animation as animation
15 
16 #Housekeeping to set plot parameters and close any open figure
17 plt.close()
18 font = {'weight' : 'bold',
19         'size'   : 22}
20 
21 mpl.rc('font', **font)
22 
```

This block of code imports the needed modules along with setting some global plot style. It also closes any open plots in line 17. (I start at line 11 since the lines of code before are comments on code description and metadata)

3 Input Data for System

```
#%% Lens Data (input as surfaces)
"""
Area to input lens system paramters. Two examples are given to show how to make the system.
Assumes even number of surfaces.
"""
surf_pow=[0.015071,0,-0.011662,-0.016327,0.005736,0.01424,0.02,0.01]
surf_dis=[40,8.74,11.05,2.78,7.63,9.54,20,4,100]
surf_ind=[1,1.617,1,1.649,1,1.617,1,3,1]
surf_tot=8

lenstot=int(surf_tot/2)
lens_pos=[]
for ii in range(lenstot):
    lens_pos.append([sum(surf_dis[0:(ii*2+1)]),30,surf_dis[ii*2+1],surf_ind[ii*2+1]])

"""
surf_pow=[0.00125,0.00125]
surf_dis=[10,4,450]
surf_ind=[1,1.516824,1]
surf_tot=2

```

```
44 lenstot=1
45 lens_pos=[[surf_dis[0],40,surf_dis[1]]]
46 """
```

Input of the optical system being used in the system being modeled. surf_pow is the optical power of each surface in diopters. surf_dis is the distance measured from surface to surface. This array is essentially the thickness of the lenses and the distance between them. surf_ind gives the refractive index of each section of the system. surf_tot gives the total surfaces the code should look through. This needs to be even since I am only doing lens systems. Lines 33-36 take this data and populate an array with lens position, height and index. This will be used when plotting the lenses. Two example systems are provided in the code. The second example doesn't use the for loop since it is a single lens.

4 Generate Ray Data

This code is split based upon which program is being used.

4.1 Marginal and Chief Ray

```
#%% Ray data [marginal ray, chief ray]
h=[14.93368,-14*np.pi/180*surf_dis[0]-6.9574]
ang=[0,14*np.pi/180]

#h=[20,-5*np.pi/180*surf_dis[0]]
#ang=[0,5*np.pi/180]


```

This code is for two specific rays being propagated through the system. This is very useful since a lens system can be fully defined by the marginal and chief ray. The user inputs the starting height and angle for these since they are defined by external parameters of the system in question.

4.2 N Rays

```
47 #%% Ray data
48 
49 #Ray data is populated below by giving amount of rays and an angle
50 
51 nray_m=80
52 nray_a=80
53 angle=5
54 h_m=np.linspace(-20,20,nray_m)
55 h_a=np.linspace(-angle*np.pi/180*surf_dis[0]-20,-angle*np.pi/180*surf_dis[0]+20,nray_a)
56 ang_m=np.zeros(nray_m)
57 ang_a=np.ones(nray_a)*angle*np.pi/180
58 
59 tot=len(h_m)+len(h_a)
60 h=np.reshape([h_m,h_a],(1,tot))[0]
61 ang=np.reshape([ang_m,ang_a],(1,tot))[0]
62 
63 
```

This code generates n rays of an object at inf and n rays for a specific angle. This gives different information then the first code since we can see and define focal points from having multiple rays. the rays are generated based upon the three parameters in 51-53. Array manipulation is done to condense the data into a single array for the generation of the data

5 Paraxial Equations

```
55 #%% paraxial equations
56
57 def refract(nin,nout,ang,y,lpow):
58     return (nin*ang-y*lpow)/nout
59
60 def prop(y,ang,dis):
61     return (ang*dis+y)
62
```

Definition of the two paraxial ray tracing equations used to propagate a ray in a medium and its refraction at a surface.

6 Generate Figure

```
93 xlimlow=0
94 xlimhigh=sum(surf_dis)+5
95 ylimlow=-(max(h)+25)
96 ylimhigh=-ylimlow
97
98 fig = plt.figure(figsize=(20, 10), dpi=80, facecolor='w', edgecolor='k')
99 ax = fig.add_subplot(111, aspect='equal', xlim=(xlimlow, xlimhigh), ylim=(ylimlow, ylimhigh),ylabel="Ray
    ↪  height (mm)",xlabel="Distance (mm)",title="Ray Tracing System")
100 ax.grid()
101
```

Here I generate the figure needed. This will allow me to later populate the figure with my lenses and rays.

7 Plot Lenses

```
102 #%% Place paraxial lenses
103 for ii in range(0,lenstot):
104     lens_use=lens_pos[ii]
105     xl=lens_use[0]
106     yl=lens_use[1]
107     wl=lens_use[2]
108     ax.add_patch(mpl.patches.Rectangle((xl,-yl), width=wl, height=2*yl, angle=0.0,
        ↪  color='b',alpha=lens_use[3]/max(surf_ind)))
109
110
```

I use the built-in patch function from matplotlib. This function takes in an xy position along with a height and width. These are found in the lens_pos array generated in lines 33-36. It also sets the trasparancy based upon a scaling around the max index of refraction lens.

8 Animation of Rays

8.1 Marginal and Chief Ray

```
111 #%% generate lines for animation from ray plot
112 plotlays, plotcols = [2], ["black","red"]
113 lines = []
114 for index in range(2):
115     lobj = ax.plot([],[],lw=2,color=plotcols[index])[0]
```

```python
    lines.append(lobj)

def init():
    for line in lines:
        line.set_data([],[])
    return lines

def animate(i):
    xu = [x[0:i],x[0:i]]
    yu = [y_marg[0:i],y_chief[0:i]]
    for lnum,line in enumerate(lines):
        line.set_data(xu[lnum], yu[lnum]) # set data for each line separately.
    return lines

ani = animation.FuncAnimation(fig, animate, interval=1, blit=True, init_func=init)

plt.show()
```

8.2 N Rays

```python
#%% generate lines for animation from ray plot
plotcols=["blue"]*nray_m+["red"]*nray_a

lines = []
for index in range(rays):
    lobj = ax.plot([],[],lw=2,color=plotcols[index])[0]
    lines.append(lobj)

def init():
    for line in lines:
        line.set_data([],[])
    return lines

def animate(i):
    xu=[]
    yu=[]
    for ll in range(rays):
        xu.append(x[0:i])
        yu.append(y_set[0:i,ll])
    for lnum,line in enumerate(lines):
        line.set_data(xu[lnum], yu[lnum]) # set data for each line separately.
    return lines

ani = animation.FuncAnimation(fig, animate, interval=1, blit=True, init_func=init)

plt.show()
```

Both codes follow the same outline. The main difference is that the first code was written first and as such didn't fully utilize loops since only 2 rays are used. The N rays is more general and populates lines for each ray in a loop