```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  4 13:01:07 2019

@author: irg16

Description: This module is used to propagate a bundle of rays through a lens and
analyse the output.
"""

import RayTracer as RT
import matplotlib.pyplot as plt
import numpy as np

"""
Define the components:
    Choose between:
        -single spherical refracting surface
        -plano-convex singlet lens:
            -with the plane surface facing the input
            -with the convex surface facing the input
"""

"""
Components for: single spherical refracting surface
"""
sphere1=RT.SphericalRefraction(100, -0.03, 1., 1.5, 10) #define the lens
plane1=RT.SphericalRefraction(101, 0, 1.5, 1.5, 10) #nonexistent plane
bun=RT.RayBundle([0,0,0],[0,0,1],5,10) #define the bundle of rays

"""
Components for: plano-convex singlet lens with the plane surface facing the input

sphere1=RT.SphericalRefraction(45, 0.02, 1.5168, 1., 10) #define the lens
plane1=RT.SphericalRefraction(40, 0, 1., 1.5168, 10) #define the plane
bun=RT.RayBundle([0,0,0],[0,0,1],5,10) #define the bundle of rays
"""

"""
Components for: plano-convex singlet lens with the convex surface facing the input

sphere1=RT.SphericalRefraction(40, -0.02, 1., 1.5168, 10) #define the lens
plane1=RT.SphericalRefraction(45, 0, 1.5168, 1, 10) #define the plane
bun=RT.RayBundle([0,0,0],[0,0,1],5,10) #define the bundle of rays
"""


"""
```

```
Defining the output plane to be at the point of the paraxial focus;
     Find an estimate of the position of the paraxial focus using:
         a ray very close to the optical axis (e.g., 0.1mm)
"""

ray0=RT.Ray([0,0.1,0],[0,0,1]) #make sure initial direction of ray is same as
bundle
output0=RT.OutputPlane(500)
sphere1.propagate_ray(ray0)
plane1.propagate_ray(ray0)
output0.propagate_ray(ray0)

"""
Arrange the dictionary of points for each ray into components:
"""
def raydata(ray):
    data_x = []
    data_y = []
    data_z = []
    for i in ray.vertices():
        data_x.append(i[0])
        data_y.append(i[1])
        data_z.append(i[2])
    return data_x, data_y, data_z

"""
Find the point that the ray intercepts the optical axis
"""

x0,y0,z0 = raydata(ray0)

x1,y1,x2,y2=z0[-2],y0[-2],z0[-1],y0[-1]
x3,y3,x4,y4=0,0,500,0
uppert = (x1-x3)*(y3-y4)-(y1-y3)*(x3-x4)
lowert = (x1-x2)*(y3-y4)-(y1-y2)*(x3-x4)
t= uppert/lowert
Px = x1+t*(x2-x1)
Py = y1+t*(y2-y1)
focal = Px
#print ('focal point:', Px)
if focal < 45:
    focal = 250 #arbitrary point for instances when the rays do not converge
output1=RT.OutputPlane(focal) #define the output plane

"""
Propagate and plot the bundle
"""
bun.FireBundle(sphere1) #propagate the bundle through the lens
```

```
bun.FireBundle(plane1) #propagate the bundle through the plane
bun.FireBundle(output1) #propagate the bundle to the output plane
bun.plotbundle() #plot the graphs for the bundle of rays
RMS = bun.RMS() #calculate the RMS for the bundle

"""
Calculate the diffraction scale and the effect on the RMS
"""
wavelen = 0.000588 #set the wavelength of the light rays - this is 588nm
beamD = bun._br*2
focallen = focal - sphere1._z0
diffscale = wavelen*focallen/beamD
print ('RMS/mm:', RMS)
print ('Diffraction scale/mm:', diffscale)
```