

Modeling a Simple Optical System

Numerically Modeling a One - Dimensional Optical System in Python 3

Landon Buell

PHYS 708.01 - Fall 2019

1 Abstract

A classical optics, a system of thin lenses can be solved analytically by use of several pertinent equations and geometric principles. As these systems become increasingly complex, the volume of math and geometry require increases drastically, meaning that a system becomes more and more prone to calculation error. In order to combat this, I have developed a program that will algorithmically evaluate the components of an optical system. Component m will take object/image $m - 1$ and use the *thin lens law* to produce object/image m . The in a system with N components, the image $m = N$ will be the final image as produced by the entire system.

2 Introduction

This project will see the creation of a python 3 program that can efficiently and accurately model a simple, ideal, one - dimensional optical system. All components in the system will be assumed to adhere to all classical models and equations. Due to complexity, all possible systems may not be physically realistic. The legitimacy of each simulation is bound only by mathematical rules.

The program will revolve around a few basic optical principles as learned from class (PHYS 708 - Optics) as well as from various other references. For all computations, light will be assumed to be an perfectly idealized geometric ray. The exact wavelength of the ray will not change the index of refraction of the object as well.

3 General Methodology

The functionality of this project will rely on object-oriented programming. Additionally, all mathematics are based on classical optical laws. The needed ones are:

The **Lens Maker's Equation**

$$\frac{1}{f} = (n_l - 1) \left(\frac{1}{R_1} - \frac{1}{R_2} \right) \quad (1)$$

Gaussian Lens Formula

$$\frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f} \quad (2)$$

Transverse Magnification factor

$$M_T = -\frac{s_i}{s_o} \quad (3)$$

In all cases, the previous parameters are defined:

- f - lens focal length from center of lens (cm)
- n_l - index of refraction of lens
- R_1 - Radius of curvature of left side of lens (cm)
- R_2 - Radius of curvature of right side of lens (cm)
- s_o - object distance from center of lens (cm)
- s_i - image distance from center of lens (cm)
- M_T - Transverse Magnification factor

3.1 Optical Components Object

The computational part of this program will involve the construction of a *class* object that will contain all attributes that are relevant to modeling the system.

The main class object used will be the *Thin Lens* object. Each lens in the system will be an additional instance of this class. It will be defined:

```
class Thin_Lens ():
    """
    Creates Thin Lens Class Object
    -----
    name (str) : Name to indentify object
    x (float) : Position on x-axis where center of lens sits
    ref_idx (float) : index of refraction of thin lens
    R1 (float) : radius of curvature 1 of lens
    R2 (float) : radius of curvature 2 of lens
    -----
    """
```

Attributes: *component name, component position, index of refraction, Radius of Curvature 1, Radius of Curvature 2, Component Focal length, lense component type.*

Initialization

As per convention, the optical component object will begin with an `__init__()` method to set all object attributes. In python, this method takes the form:

```
def __init__(self,name,x,ref_idx,R1,R2):
    """ Initiaialize Thin Lens Class Object """
    self.name = name           # name of object
    self.x = x                 # horizontal position
    self.n = ref_idx           # refraction idx
    self.R1 = R1               # rad of curve 1
    self.R2 = R2               # rad of curve 2
    self.f = self.focus()      # focal length
    self.type = self.lens_type()# lens type
```

The method `self.focus()` computes the focal distance of the lens class instance based on the *Lens Maker's Equation*, Equation (1). The `self.lens_type()` method determines the type of thin lense based on the two radii of curvature, as defined below:

Other Methods

The *Thin_Lens()* class must have method that allows for the computation of the image distance as measured from the center of the thin lens. This will be computed as per the *Gaussian Lens Formula*, Equation (2). It is defined:

```
def image_distance(self,obj):
    """ Compute image distance from lens instance """
    d = obj.x - self.x         # distance between obj & lens
    si = (1/self.f - 1/d)**-1   # image distance
    return si                  # return image distance
```

Using this image distance s_i , and the horizontal position of the center of the lens from the origin of the coordinate system. Note that this method uses the parameter *obj* which is the class object that represents the physical image or object.

The `textitThin.Lens()` class also has a method for determining it's lens type. As defined by standard optical lens conventions, The method uses the radii of curvature and a series of if statements. it is defined:

```
def lens_type(self):
    """ Determine lens type based on radii of curvature """
```

```

if (self.R1 > 0) and (self.R2 < 0):
    return 'Biconvex'          # biconvex lens
if (self.R1 > 1e6) and (self.R2 < 0):
    return 'Planar - Convex'    # planar convex lens
if (self.R1 > 0) and (self.R2 > 0):
    return 'Meniscus - Convex'  # meniscus convex len
if (self.R1 < 0) and (self.R2 > 0):
    return 'Biconcave'          # biconcave lens
if (self.R1 > 1e6) and (self.R2 > 0):
    return 'Planar - Concave'   # planar concave lens
if (self.R1 > 0) and (self.R2 > 0):
    return 'Meniscus Concave'   # meniscus concave lens

```

The simplification has been made where if the radius of curvature of a lens is greater than 10^6 units, it is treated as if to have an 'infinite' radius of curvature - i.e. it is a flat surface. This was made to handle large values, and potentially infinite ones.

3.2 Object & Image Object

The image of each lense will

3.3 Visualization

Once the system has been tracked computationally, a visualization is produced. This visualization is done in Python's *matplotlib.pyplot* module and contains the position of all optical components, position of respective focii and each successive image as produced by the system. All components are labeled and visualized to scale. The plot will be displayed when the system is finished with all computations. Note that radii of curvature of each lens are not considered in visualization.

In addition to the plot, the details for each component and object or image will also be printed out concisely for the ease of human reading. Again, this will happen at the end of the full computation cycle.

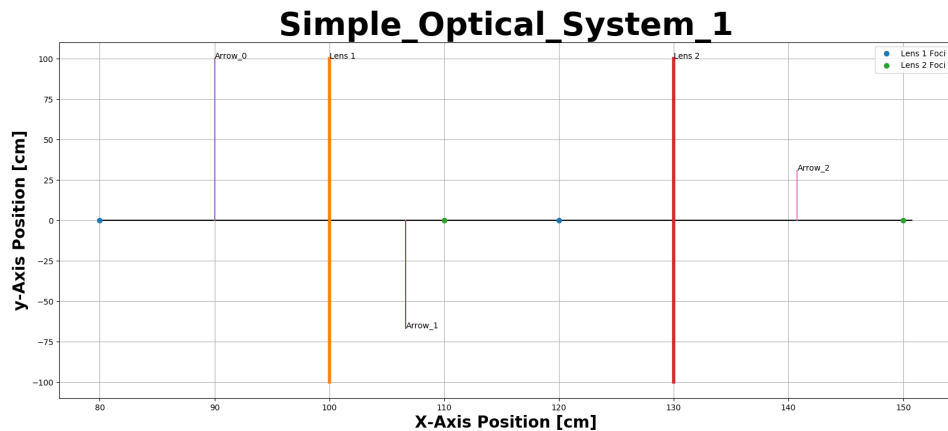
4 Example System #1

The first test of the program will be a simple system, containing two lenses. The program prints out the details of each lens:

```
=====
Component Number: 1
    Name: Lens 1
    Component type: Biconvex lens
    X-Axis Postion: 100
    1st Curvature radius: 20
    2nd Curvature radius: -20
    Focal length: 20.0
```

```
=====
Component Number: 2
    Name: Lens 2
    Component type: Biconvex lens
    X-Axis Postion: 130
    1st Curvature radius: 20
    2nd Curvature radius: -20
    Focal length: 20.0
```

And finally, the visualiztion of the system:



The legitimacy of the computation can be confirmed by hand or with another program if desired.

5 Example System #2

6 Additional Notes

- **Realistic Systems**

Due to the lack of detailed error handling, is it possible to attempt to model physically unrealistic systems. These may included (but not limited to): over lapping lenses, infinite or undefined values, etc.

- **Python Modules**

Required Python Modules: Numpy, Scipy, Matplotlib

7 Conclusion

8 References

- “Geometrical Optics.” Optics, by Eugene Hecht, 5th ed., Pearson, 2017, pp. 151–246.