

# 1 Objectif

Un traceur de rayon est un programme qui permet de calculer la propagation d'un faisceau lumineux à travers un système optique. Des programmes commerciaux (tels que [Zemax](#) ou [OSLO](#)) permettent de faire de tel calcul. Notre objectif est de réaliser en Python un tel programme.

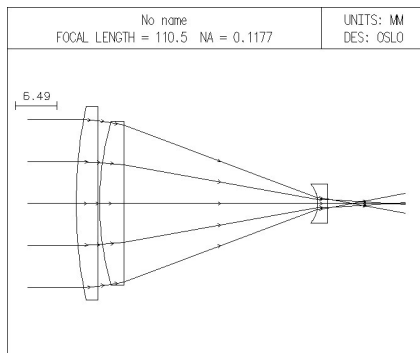


FIGURE 1 – Tracé de rayon obtenu avec le logiciel OSLO. Sur ce triplet de lentille, nous pouvons voir les aberrations sphériques.

## 2 Dioptre optique

Un dioptre optique est l'interface entre deux milieux d'indice différent. On supposera que nos dioptres sont sphériques.

Un dioptre sphérique est déterminé par la position  $z_0$  de son intersection avec l'axe  $z$  et par son rayon  $R$ . Le rayon  $R$  peut être positif ou négatif, la convention étant que le centre du dioptre se trouve en  $z_0 + R$ . Il sépare deux milieux d'indice  $n_1$  et  $n_2$ . Son diamètre est aussi donné.

On va créer une classe `Dioptre` de la façon suivante :

```
class Dioptre(object):
    diametre = 25.4
    def __init__(self, z0, R, n_1, n_2, diametre=None):
        self.z0 = z0
        self.R = float(R)
        if diametre is not None:
            self.diametre = diametre
        self.n_1 = n_1
        self.n_2 = n_2
```

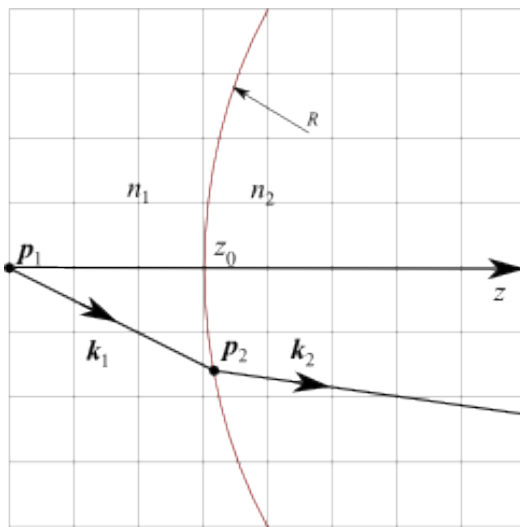


FIGURE 2 – Traversé d'un dioptré sphérique

- Expliquer ce que l'on a fait avec le diamètre. Pourquoi ne pas avoir plus simplement mis `diametre=25.4` dans la définition de `__init__` ?
- Dans la suite, on aura besoin de connaître la position `z_center` du centre du dioptré. Calculer et rajouter cet attribut dans le `__init__`.
- Ecrire une méthode `__repr__` qui affiche les principaux paramètres du dioptré
- Ecrire une méthode `plot` qui trace le dioptré dans un graphique.

### 3 Loi de Snell - Descartes

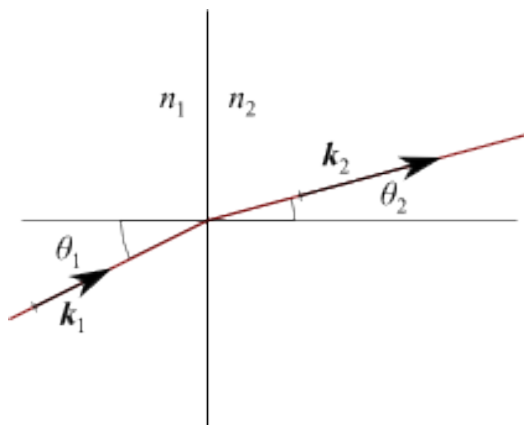


FIGURE 3 – Rayon traversant un dioptré

La loi de Snell - Descartes permet de relier l'angle de réfraction à l'angle d'incidence. Elle est connue sous la forme  $n_1 \sin(\theta_1) = n_2 \sin(\theta_2)$ . Cependant, une façon beaucoup plus commode d'utiliser cette loi consiste à utiliser les vecteur d'onde  $\mathbf{k}_1$  et  $\mathbf{k}_2$ . Ces vecteurs sont parallèles à la propagation du faisceau et ont une longueur proportionnelle à l'indice  $n$  du milieu. La loi de Snell-Descartes dit alors que la composante du vecteur  $k$  parallèle au dioptre est conservée.

## 4 Rayon lumineux

Un rayon lumineux est déterminé par un point  $\mathbf{p}$  et le vecteur  $\mathbf{k}$ . On veillera à ce que la norme de  $\mathbf{k}$  soit toujours égal à l'indice du milieu. On représentera un rayon par la classe suivante :

```
class Rayon():
    def __init__(self, p0, k, n=None):
        self.p0 = p0
        self.k = k
        if n is not None:
            self.normalize(n)
    def normalize(self, n):
        """Normalise le rayon pour que la norme de k soit n"""
        pass
    def __repr__(self):
        return "Rayon(p0={p0}, k={k} ".format(p0=self.p0, k=self.k)
```

— Ecrire la méthode `normalize`.

## 5 Traversé d'un dioptre sphérique

Un rayon  $(\mathbf{p}_1, \mathbf{k}_1)$  intersecte le dioptre à la position  $\mathbf{p}_2$ . Le nouveau rayon sort avec une direction  $\mathbf{k}_2$ .

On souhaite créer une méthode `traversee(self, rayon)` de `Dioptre` dont le but est de calculer le rayon de paramètre  $\mathbf{p}_2$  et  $\mathbf{k}_2$ .

On utilisera des tableaux numpy pour stocker des vecteurs.

Aide :

- L'intersection avec la sphère s'obtient en paramétrant la position d'un point du rayon par  $\mathbf{p}(t) = \mathbf{p}_1 + t\mathbf{k}$  puis en résolvant l'équation du second degré en  $t$  disant  $\|\mathbf{p}(t) - \mathbf{c}\|^2 = R^2$  où  $\mathbf{c}$  désigne le centre de la sphère. Créer une fonction qui effectue cette tâche.
- Calculer le vecteur normal orthogonal à la surface au point  $\mathbf{p}_2$ . En déduire la composante parallèle à la surface  $\mathbf{k}_{\parallel} = \mathbf{k} - (\mathbf{k} \cdot \mathbf{n})\mathbf{n}$ . Puis le vecteur  $\mathbf{k}_2$  avec  $\mathbf{k}_2 = \mathbf{k}_{\parallel} + \alpha\mathbf{n}$  où  $\alpha$  est choisi tel que  $\|\mathbf{k}_{\parallel}\|^2 + \alpha^2 = n_2^2$ . Attention au signe de  $\alpha$ . Il est possible qu'il n'y ait pas de solution à cette équation. Qu'elle est la signification physique ?

On fera les calculs avec les paramètres suivants

```

p1 = np.Array([0,0,-3])
z0 = 0
R = 6
n1 = 1
n2 = 1.5
kl_x = np.array([0,.5,math.sqrt(.75)])

```

## 6 Faisceau

Un faisceau est une liste de rayon. Nous créons donc une nouvelle classe `Faisceau` qui hérite de `list` :

```

class Faisceau(list):
    #Il n'y a pas de méthode __init__ car on utilise celle de list
    def plot(self):
        pass

```

- Ecrire la méthode `plot`. On tracera uniquement les faisceaux contenant au moins deux rayons en joignant les points de départ de chacun des rayons.

## 7 Système optique

Un système optique est une liste de dioptré. Nous créons donc une nouvelle classe `SystemeOptique` qui hérite de `list` :

```

class SystemeOptique(list):
    #Il n'y a pas de méthode __init__ car on utilise celle de list
    def calcul_faisceau(self,r0):
        faisceau = Faisceau()
        faisceau.append(r0)
        for dioptré in self:
            faisceau.append(dioptré.traversee(faisceau[-1]))
        return faisceau
    def plot(self):
        pass

```

- Ecrire la méthode `plot` (deux lignes).

## 8 Exemple

Voici l'exemple d'utilisation de notre programme (lentille du catalogue Thorlabs) :

```

wave_length = 780E-6 # mm

n_LAH64 = 1.77694
n_SF11 = 1.76583
n_air = 1.0002992

S1 = Dioptre(0,-4.7, n_air, n_SF11, diametre=3)

S2 = Dioptre(1.5,1E10, n_SF11, n_air, diametre=3)

LC2969 = SystemeOptique()
LC2969.append(S1)
LC2969.append(S2)

plan_image = Dioptre(100 ,1e10,30,n_air,n_air)

system = SystemeOptique()
system.extend(LC2969)
system.append(plan_image)

r0 = Rayon(p0=np.array([0,1,-5]), k=np.array([0,0,1]), n=n_air)
faisceau = system.calcul_faisceau(r0)

LC2969.plot()
faisceau.plot()

```

## 9 Pour aller plus loin

- La méthode `append` et `extend` de `SystemeOptique` hérite de celle de `list`. Il serait judicieux de les réécrire afin de vérifier que l'argument de `append` est uniquement un `Dioptre` et de `extend` un `SystemeOptique`. On effectuera donc un test, on soulèvera une erreur si la condition n'est pas valide, et ensuite, on appellera la méthode du parent (par exemple `list.append(self, arg)`).
- Ecrire les méthodes `__add__` de `Dioptre` et `SystemeOptique`. On souhaite pouvoir simplement écrire :

```

S1 = Dioptre(0,-4.7, n_air, n_SF11, diametre=3)
S2 = Dioptre(1.5,1E10, n_SF11, n_air, diametre=3)
LC2969 = S1 + S2

plan_image = Dioptre(100, 1e10, 30, n_air, n_air)

system = LC2969 + plan_image

```

- Ecrire une méthode `translated(self, d)` pour `Dioptre` et `SystemeOptique`. Elle renvoie un nouvel objet déplacé de la distance `d`.