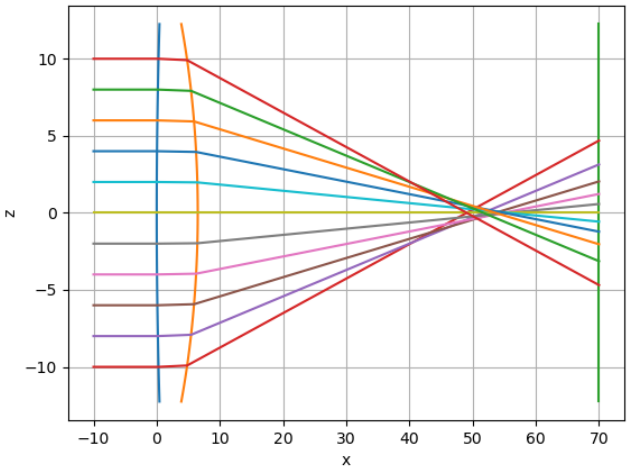
Traceur de rayons

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 Code V, Zemax ou OSLO) permettent de faire de tels calculs. Dans ce TP nous allons programmer un traceur de rayons avec Python. Identifions dans un premier temps les objets élémentaires dans ce problème. Il faut pouvoir décrire des dioptres et des rayons. Nous allons donc commencer par écrire une classe *Diotre*, et une classe *Ray*. Puis nous introduirons des méthodes permettant l'interaction entre ces classes. Dans la suite les variables à utiliser dans Python seront notées en italique. Nous utiliserons les millimètres pour mesurer les longueurs.



Exemple de tracé de rayon pour une lentille convergente composée de deux dioptres sphériques.

1 Dioptres sphériques

Un dioptre est une interface entre deux milieux d'indices différents. Dans ce TP nous allons nous limiter aux dioptres sphériques. Un dioptre est caractérisé par son intersection avec l'axe optique : *position*, par son rayon de courbure : *radius_of_curvature*, par les indices de réfraction de part et d'autre : *refractive_index* (tuple à deux éléments, le premier étant l'indice à gauche et le second l'indice à droite), par son diamètre (extension suivant z) : *diameter*.

1. Créer la classe *Dioptre*, avec comme argument obligatoire les variables *position*, *radius_of_curvature*, *refractive_index*, et comme argument optionnel *diameter*, dont vous fixerez la valeur par défaut à 25.4 mm (1 pouce). On pourra utiliser une dataclass
2. Créer une property *center* qui renvoit la position du centre de la sphère du dioptre.

Le dioptre étant sphérique, il a pour équation $z^2 + (x - c)^2 = R^2$, où R est le rayon de courbure et c le centre. La position du dioptre est donc $x_0 = c + R$, le rayon de courbure pouvant-être positif ou négatif, suivant la concavité du dioptre. Avec cette convention, on a

$$x = c - \text{sign}(R)\sqrt{R^2 - z^2}.$$

3. Ajouter une méthode *shape* qui prend en entrée une valeur de z et renvoie la valeur x donnée par l'équation précédente. Cette méthode devra fonctionner avec des tableau numpy.
4. Ajouter une méthode *plot* qui trace le dioptre sur une figure.

2 Rayon lumineux

Un rayon lumineux peut être décrit par un point $\vec{p}$ (vecteur à deux composantes) et par un vecteur directeur $\vec{k}$ (à deux composantes également).

5. Créer la classe *Ray*, ayant pour argument obligatoire *starting_point* (le point $\vec{p}$), *direction* (le vecteur $\vec{k}$), et pour argument optionnel *refractive_index* dont la valeur par défaut sera l'objet *None*. On s'assurera que le point $\vec{p}$ et le vecteur $\vec{k}$ sont enregistré sous forme de tableau numpy.
6. Comme tous les vecteurs d'ondes sont proportionnels à l'indice de réfraction du milieu, on peut choisir de normaliser les vecteurs d'ondes par l'indice du milieu (c'est à dire que l'on prend $2\pi/\lambda = 1$). Ajouter une méthode *normalize_direction_to_refractive_index* qui permet de normaliser $\vec{k}$ par l'indice de réfraction.
7. Normaliser $\vec{k}$ par l'indice de réfraction dans l'initialiseur, uniquement si l'indice de réfraction est passé comme argument.

3 Loi de Snell et Descartes

Nous avons maintenant deux classes pour décrire les objets élémentaire du problème considéré. Il faut écrire des méthodes de ces classes qui décrivent la réfraction des rayons par les dioptres. Il y a plusieurs manières de procéder. Ici, on propose d'ajouter ces méthodes à la classe *Dioptre*.

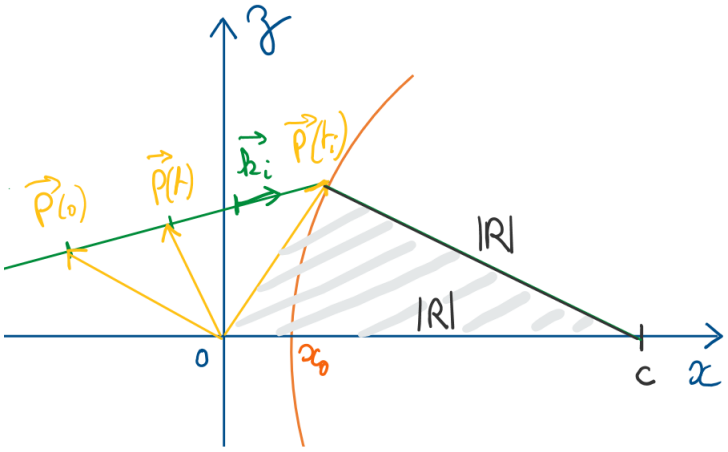
8. Ajouter une méthode *get_refracted_ray* à la classe *Dioptre*. Cette méthode prendra comme argument la variable *incident_ray*, qui devra être le rayon incident (une instance de la classe *Ray*). Dans un premier temps, ajouter l'unique instruction *pass* à cette méthode.

L'objectif est que cette méthode renvoie le rayon réfracté. Pour commencer il faut d'abord déterminer le point d'intersection entre le rayon incident et le dioptre. Nous allons écrire une méthode spécifique pour cela. L'équation paramétrique du rayon incident est $\vec{p}(0) + t\vec{k}_i$. On cherche le paramètre t_i, tel que $\vec{p}(t_i) = \vec{p}(0) + t_i\vec{k}_i$ soit le point d'intersection entre le rayon incident et le dioptre. Dans le triangle grisé, on doit donc résoudre :

$$\begin{aligned} \|\vec{p}(t) - \vec{c}\|^2 &= R^2 \\ \|\vec{p}(0) - \vec{c} + t\vec{k}_i\|^2 &= R^2 \\ \alpha t^2 + \beta t + \gamma &= 0, \end{aligned}$$

avec

$$\begin{aligned} \alpha &= \|\vec{k}_i\|^2 \\ \beta &= 2\vec{k}_i \cdot (\vec{p}(0) - \vec{c}) \\ \gamma &= \|\vec{p}(0) - \vec{c}\|^2 - R^2. \end{aligned}$$



Intersection entre un rayon et un dioptre sphérique. Le dioptre est représenté en orange, le rayon incident en vert.

9. Ajouter une méthode *intersection* à la classe *Dioptre*, qui prend en argument un rayon, et renvoie un tableau numpy à deux composantes contenant les coordonnées de $\vec{p}(t_i)$. Dans cette méthode, il faudra donc déterminer les racines du trinôme du second degré ci-dessus.

On utilise ensuite la loi de Snell et Descartes : la composante parallèle du vecteur d'onde (projection du vecteur d'onde sur la tangente au dioptre, au point d'incidence) est conservée lors de la réfraction. On peut déterminer cette composante parallèle en soustrayant à $\vec{k}_i$ sa composante perpendiculaire :

$$\vec{k}_{\parallel} = \vec{k}_i - \vec{k}_{i,\perp} = \vec{k}_i - (\vec{k}_i \cdot \vec{n}) \cdot \vec{n}.$$

On a introduit $\vec{n}$, le vecteur normal à la tangente au point d'incidence (cf. figure ci-contre). Par ailleurs, sur la figure précédente on voit que

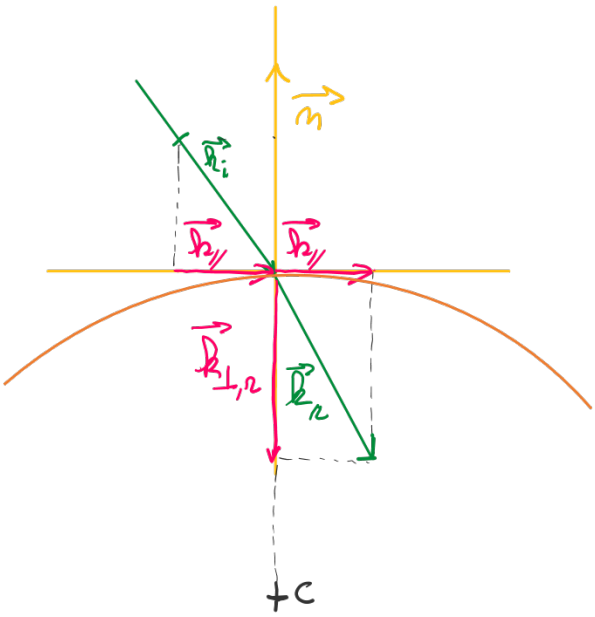
$$\vec{n} \propto \vec{p}(t_i) - \vec{c}.$$

La composante perpendiculaire du vecteur d'onde du rayon réfracté s'obtient ensuite en utilisant le fait que $\|\vec{k}_r\| = n_2$, où n_2 est l'indice de réfraction du côté droit du dioptre. On a donc

$$\vec{k}_{r,\perp} = \sqrt{n_2^2 - \|\vec{k}_{\parallel}\|^2}\ \vec{n}.$$

On en déduit le vecteur d'onde réfracté

$$\vec{k}_r = \vec{k}_{\parallel} - \mathrm{sign}(R)\vec{k}_{r,\perp}.$$



Réfraction du rayon incident sur le dioptre sphérique. Par rapport à la figure précédente, une rotation a été effectuée pour faciliter le tracé et la visualisation.

10. Compléter la méthode *get_refracted_ray* :
    (a) Déterminer $\vec{p}(t_i)$ en appelant la méthode *intersection*.
    (b) Déterminer le vecteur unitaire $\vec{n}$.
    (c) Calculer $\vec{k}_{\parallel}$.
    (d) Calculer $\vec{k}_{r,\perp}$.
    (e) En déduire $\vec{k}_r$, et retourner l'objet rayon réfracté.

4 Trajet complet d'un rayon et système optique

L'essentiel du travail est fait. Pour faciliter la programmation du tracé de rayon nous pouvons écrire une classe *OpticalSystem* pour décrire un système optique comprenant plusieurs dioptres. Cette classe aura pour attribut une liste de Dioptre. De même afin de décrire un trajet complet de rayons à travers les différents dioptres, on peut regrouper les différents rayons dans une liste.

11. Créer la classe *OpticalSystem*.
12. Ajouter à la classe *OpticalSystem* une méthode *plot* qui trace les dioptres de la liste.

13. Créer la classe *RayList*.
14. Ajouter à la classe *RayList* une méthode *plot* qui trace des segments entre les points de départ des rayons de la liste.
15. Créer une class *Screen* qui est un dipotre plan ne faisant rien. On changera la class *Dipotre* en *SphericalDioptre* et on fera hériter *Screen* et *SphericalDioptre* d'un classe vide *Dipotre*.
16. Ajouter à la classe *OpticalSystem* une méthode *ray_tracing* qui prend comme argument un rayon (instance de la classe *Ray*) incident sur le système optique, et qui renvoie une liste de rayons (instance de la classe *Raylist*) correspondant au trajet de la lumière à travers les différents dioptres du système optique.
17. Tester le programme en générant une figure similaire à celle donnée en introduction.
18. (Bonus) Ecrire les méthodes spéciales afin que le code suivant puisse fonctionner (méthode *__add__* pour assembler le dioptre ou le système optique, *__matmul__* pour déplacer le dioptre ou le système optique, *__neg__* pour inverser le sens d'un dioptre ou système optique) :

```
n_LAH64 = 1.77694
n_SF11 = 1.76583
n_BK7 = 1.5112
n_air = 1.0002992

#Reference from https://www.thorlabs.com/
s1 = Dioptre(0, -4.7 , (n_air, n_SF11), diameter=3)
s2 = Dioptre(1.5, 1E10, (n_SF11, n_air), diameter=3)
LC2969 = s1 + s2 # Return an OpticalSystem

s1 = Dioptre(0, 1E10 , (n_air, n_BK7))
s2 = Dioptre(4.1, -38.6, (n_BK7, n_air))
LA1608 = s1 + s2 # Return an OpticalSystem

system = -LC2969 + LA1608@50
```