Modèle de Snell-Descartes

September 21, 2024

1 Modèle de Snell-Descartes

On commence par importer les library utiles pour réaliser les simulations numériques.

[1]:
```
#on commence classiquement par importer la library numpy (sous l'alias np) pour␣
 ↪le calcul numérique
#son sous module numpy.random pour effectuer les tirages aléatoires selon des␣
 ↪lois bien controlées
#et la library matplotlib.pyplot (sous l'alias pl) pour la réalisation de␣
 ↪graphique.
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as pl
```

1.1 Evaluation de l'incertitude sur sin(i0) et sin(i1)

Pour évaluer l'incertitude sur le sinus d'un angle, connaissant l'incertitude sur l'angle lui même, on va à nouveau exploiter une simulation de Monte Carlo. Cette démarche sera obligatoire dès que le lien entre la grandeur directement issue de l'expérience et la grandeur à évaluer est complexe. Un petit nombre de situation donnera lieu à un traitement direct sans passer forcément par l'outil de simulation (cas d'une formule de type somme ou produit).

[11]:
```
#on créée une liste de 10000 tirages aléatoires uniformément répartis sur␣
 ↪l'intervalle étudié.
#ce qu'on appelle généralement simulation Monte-Carlo.
N=10000

#valeur expérimentale de i0 l'angle d'incidence et i1 l'angle de réfraction
angle_incidence=40
angle_refraction=75


#Demi largeur de l'intervalle de mesure
l_angle_inc=1
l_angle_refrac=2

# simultation montéCarlo pour le sin(i0) et le sin(i1)
angle_incidence_MC=angle_incidence+rd.uniform(-l_angle_inc,l_angle_inc,N)
```

```
sin_incidence_MC=np.sin(np.pi/180*angle_incidence_MC)

angle_refraction_MC=angle_refraction+rd.
 ↪uniform(-l_angle_refrac,l_angle_refrac,N)
sin_refraction_MC=np.sin(np.pi/180*angle_refraction_MC)

#on calcul les valeurs moyennes sur le tirage aléatoire réalisé
moyenne_incidence=round(np.average(sin_incidence_MC),3)
print("la valeur moyenne du sin(i0) est égale à ",moyenne_incidence)

moyenne_refraction=round(np.average(sin_refraction_MC),3)
print("la valeur moyenne du sin(i1) est égale à ",moyenne_refraction)

#on calcul les écarts type sur le tirage aléatoire réalisé
u_incidence=round(np.std(sin_incidence_MC,ddof=1),3)
print("l'incertitude associée à sin(i0) est égale à ",u_incidence)
u_refraction=round(np.std(sin_refraction_MC,ddof=1),3)
print("l'incertitude associée à sin(i1) est égale à ",u_refraction)

#on obtient enfin une évaluation de n et l'incertitude associé
indice_MC=sin_refraction_MC/sin_incidence_MC
moyenne_indice=round(np.average(indice_MC),3)
print("l'évaluation de l'indice obtenu est", moyenne_indice)
u_indice=round(np.std(indice_MC),3)
print("l'évaluation de l'incertitude sur l'indice est", u_indice)
```

```
la valeur moyenne du sin(i0) est égale à  0.643
la valeur moyenne du sin(i1) est égale à  0.966
l'incertitude associée à sin(i0) est égale à  0.008
l'incertitude associée à sin(i1) est égale à  0.005
l'évaluation de l'indice obtenu est 1.503
l'évaluation de l'incertitude sur l'indice est 0.02
```

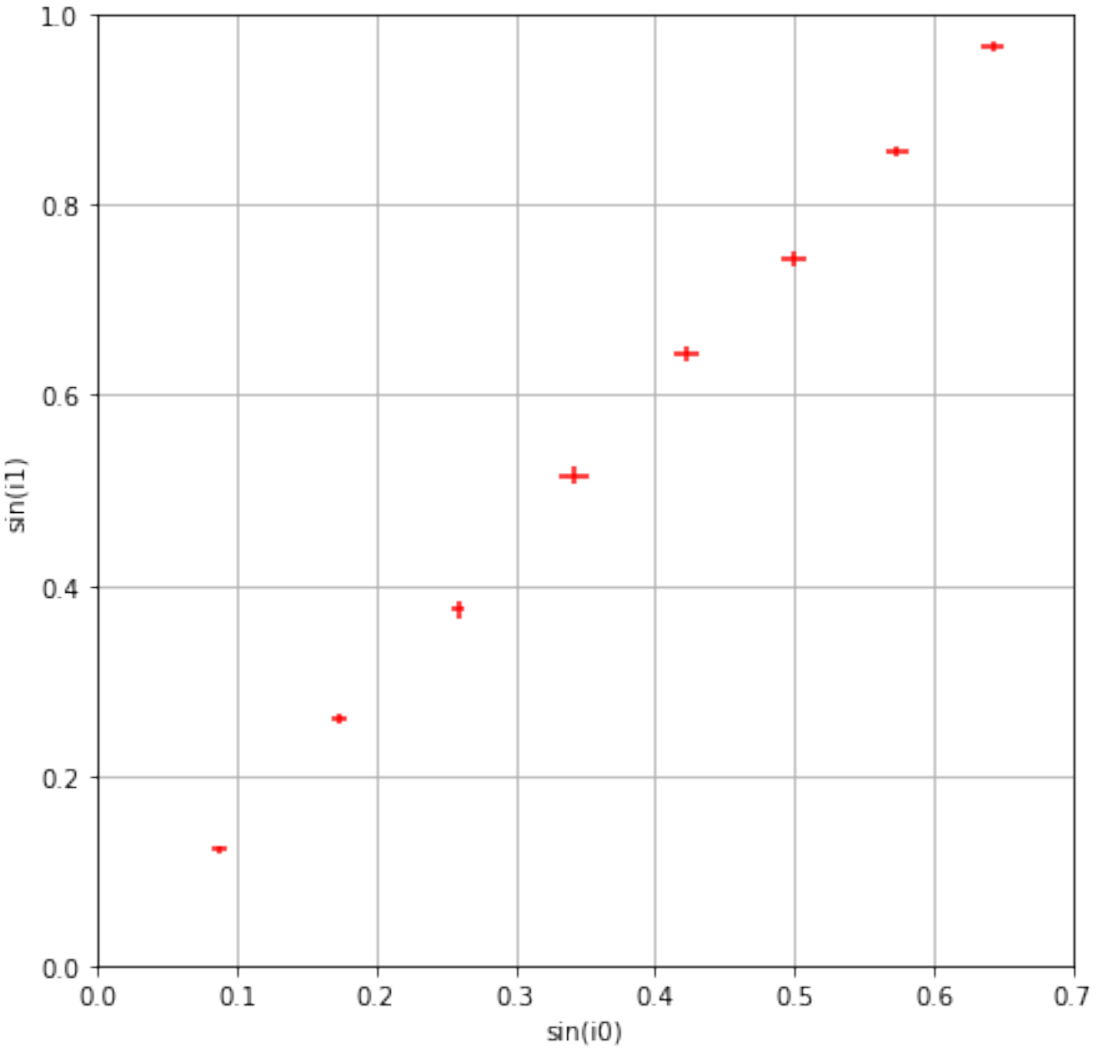
1.2 exemple de tableau de résultats obtenus

On reprend alors les résultats obtenus par la simulation sous python pour établir le tableau complet des moyennes et incertitudes sur toutes les grandeurs évaluées dans cette expérience :

i0	i1	sin(i0)	u(sin(i0))	sin(i1)	u(sin(i1))	n	u(n)
5	7	0,087	0,005	0,122	0,005	1,403	0,10
10	15	0,174	0,005	0,259	0,005	1,492	0,051
15	22	0,259	0,005	0,375	0,009	1,448	0,045
20	31	0,342	0,01	0,515	0,009	1,507	0,049
25	40	0,423	0,009	0,643	0,008	1,522	0,038
30	48	0,5	0,009	0,743	0,007	1,487	0,029
35	59	0,573	0,008	0,857	0,005	1,495	0,023
40	75	0,643	0,008	0,966	0,005	1,503	0,020

1.3 Tracé de la courbe expérimentale sin(i1)=f(sin(i0))

[14]:
```
#On construit les tableaux des valeurs des angles mesurés.
angle_inc=np.array([5,10,15,20,25,30,35,40])
angle_ref=np.array([7,15,22,31,40,48,59,75])

# On construit les tableaux des valeurs d'angle associés.
sin_inc=np.sin(np.pi/180*angle_inc)
sin_ref=np.sin(np.pi/180*angle_ref)

#on construit les tableaux des incertitudes sur chaque valeur de sinus
u_inc=np.array([0.005,0.005,0.005,0.01,0.009,0.009,0.008,0.008])
u_ref=np.array([0.005,0.005,0.009,0.009,0.008,0.007,0.005,0.005])

# création d'une figure
pl.figure(figsize = (7,7))
# représentation des points expérimentaux avec barres d'incertitude à \pm u
pl.errorbar(sin_inc, sin_ref, xerr = u_inc, yerr = u_ref, fmt = ',', color =␣
 ↪'r')

#déco axe des abscisses
pl.xlim(0,0.7), pl.xlabel("sin(i0)")
#déco axe des ordonnées
pl.ylim(0,1.), pl.ylabel("sin(i1)")

#axe des abscisses et axe des ordonnées sur la même échelle
#pl.axis("equal")
#Affichage d'une grille
pl.grid()

#affichage de la figure
pl.show()
```



On peut observer alors que : - L'allure de la courbe semble bien corroborrée un modèle linéaire (ou affine). On illustrera par la suite la vérification avec des critère plus précis de cette affirmation. - La pente de cette droite donne alors la valeur expérimentale de l'indice optique mais elle ne donne pas accès à l'incertitude sur cette mesure.

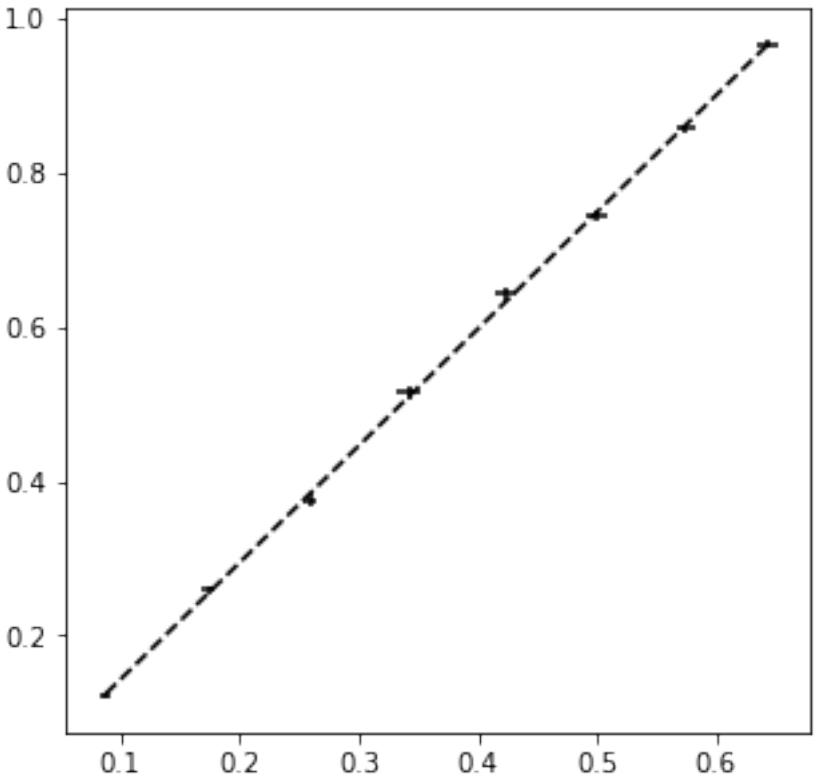
1.4 Vérification du modèle linéaire par évaluation des résidus.

Pour vérifier le modèle linéaire, on va : - effectuer une régression linéaire avec python pour déterminer la meilleure droite modèle. - Evaluer et représenter les résidus avec les barres d'erreur. - Discuter de l'allure de la courbe obtenue.

[16]:
```
#on détermine les paramètres de la modélisation affine se rapprochant le plus␣
 ↪possible du nuage de points expérimental.
# la liste p produite par la fonction polyfit de la librairy numpy est la liste␣
 ↪des coefficients polynomiaux
```

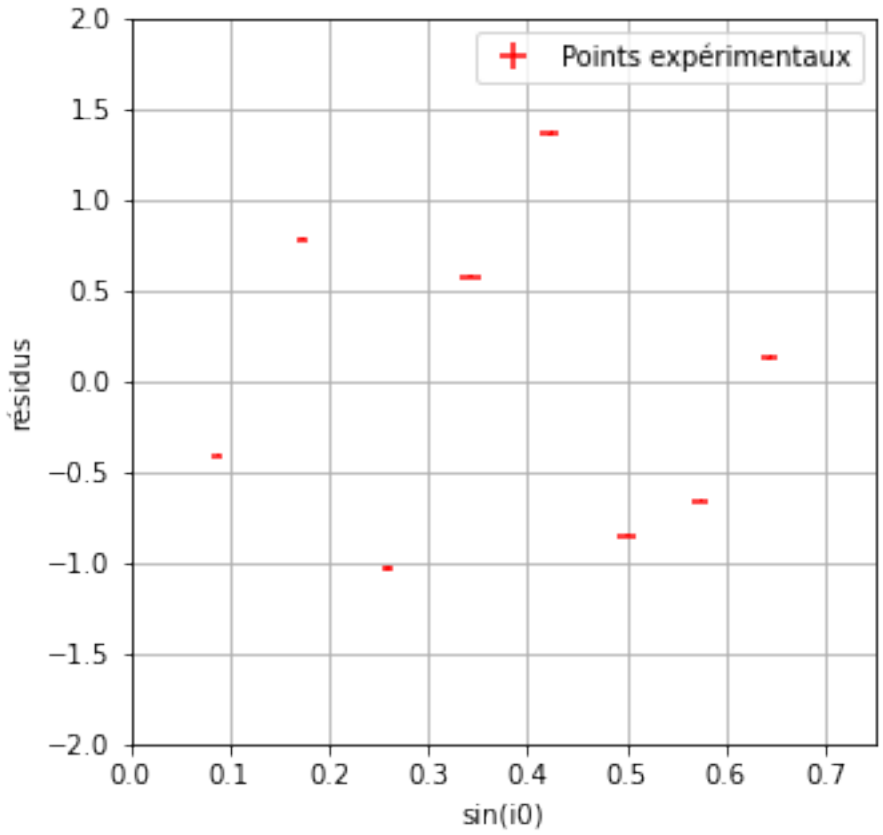
```
p = np.polyfit(sin_inc, sin_ref, 1)

#affichage de la valeur de la pente donnant la valeur estimée de n
print("l'indice optique est estimé par la pente qui vaut : ",p[0])
print("mais il manque l'évaluation de l'incertitude sur cette pente !!!")


# On trace sur une figure le nuage de points expérimentaux et la drotie␣
 ↪d'ajustement
pl.figure(figsize = (5,5))

# on effectue une représentation des points expérimentaux
pl.errorbar(sin_inc, sin_ref, xerr = u_inc, yerr = u_ref, fmt = ',', color =␣
 ↪'k', label = 'Points expérimentaux')

# représentation de la droite d'ajustement
pl.plot(sin_inc, np.polyval(p,sin_inc), 'k--', label = 'Modèle affine')

pl.show()
```

```
l'indice optique est estimé par la pente qui vaut :  1.5142301917314245
mais il manque l'évaluation de l'incertitude sur cette pente !!!
```



La droite modèle et le nuage de points expérimentaux semble bien correspondre, mais les barres d'erreur étant très petites, cette représentation graphique ne permet pas de conclure avec une sécurité totale. On va donc étudier la représentation graphique des résidus, et des écarts normalisés.

[17]:
```
#on calcul les résidus c'est à dire l'écart entre les points expérimentaux et␣
 ↪le modèle affine
residus = (sin_ref - np.polyval(p,sin_inc))

## Vérification graphique de la pertinence (ou non !) de la modélisation par␣
 ↪représentation des écarts normalisés
# c'est à dire le rapport des résidus et de l'incertitude type sur le point de␣
 ↪mesure associé
pl.figure(figsize = (5,5))


# représentation des points expérimentaux
pl.errorbar(sin_inc, residus/u_ref, xerr = u_inc, yerr = u_ref, fmt = ',',␣
 ↪color = 'r', label = 'Points expérimentaux')


pl.xlim(0,0.75), pl.xlabel("sin(i0)")
pl.ylim(-2,2), pl.ylabel("résidus")
pl.grid(), pl.legend(loc = 0)

pl.show()
```



- L'affichage de l'écart normalisé, raport des résidus avec l'incertitude sur la valeur de sin(i1) donne des valeurs qui sont toutes dans l'intervalle [-2,2].
- Les points sont distribués autour de la valeur nulle de manière qui semble aléatoire, on n'identifie par d'évolution continue de la position des points dans le graphique représentant les résidus. **On peut alors conclure que le modèle affine est bien vérifié.

1.5 Simulation Monte Carlo pour la loi de Snell-Descartes.

La simulation Monte Carlo peut aussi s'appliquer à la construction de la droite de régression étudiée précédemment. En effectuant N tirages aléatoires sur cette droite de régression, on aura accès à N valeurs de la pente. On pourra alors appliquer un traitement statistique pour obtenir une évaluation de l'indice optique n qui sera complète, avec une valeur éstimée obtenue par évaluation de la moyenne de ces pentes, et une incertitude type obtenue par évaluation de l'écart-type standard.

[18]:
```
#on identifie le nombre de mesures effectuées
n_mesure=len(angle_inc)

#on initialise un tableau de tirages aléatoires sur l'angle incident
angle_inc_MC=np.zeros([N,n_mesure])

for i in range(n_mesure):
```

```
#on construit alors le tableau des tirages aléatoire pour l'angle d'incidence
    angle_inc_MC[:,i]=angle_inc[i]+rd.uniform(-l_angle,l_angle,N)
#on construit le tableau des tirages aléatoire pour le sinus de l'angle␣
 ↪d'incidence
    sin_inc_MC=np.sin(np.pi/180*angle_inc_MC)

#on reprend la démarche pour l'angle de réfraction
#on initialise un tableau de tirages aléatoires sur l'angle incident
angle_ref_MC=np.zeros([N,n_mesure])

for i in range(n_mesure):
#on construit alors le tableau des tirages aléatoire pour l'angle de réfraction
    angle_ref_MC[:,i]=angle_ref[i]+rd.uniform(-l_angle,l_angle,N)
#on construit le tableau des tirages aléatoire pour le sinus de l'angle de␣
 ↪réfraction
    sin_ref_MC=np.sin(np.pi/180*angle_ref_MC)


#On intitalise une liste des évaluations de l'indice en la prenant vide.
list_indice=[]

for i in range(N):
#on effectue la régression linéaire sur les N simulations de nuages de points␣
 ↪avec polyfit
    p = np.polyfit(sin_inc_MC[i,:], sin_ref_MC[i,:], 1)
#On extrait la pente de la droite pour chaque simulation et on l'enregistre␣
 ↪dans la liste.
    list_indice=list_indice+[p[0]]

# On évalue la moyenne des pente pour obtenir une estimation de la valeur de␣
 ↪l'indice.
moyenne_list_indice=np.average(list_indice)
#on évalue l'acert type pour avoir accès à l'incertitude type
u_list_indice=np.std(list_indice)

print("La moyenne des pentes sur les tirages aléatoires␣
 ↪est",moyenne_list_indice)
print("l'incertitude entre les tirages aléatoires est", u_list_indice)
```

```
La moyenne des pentes sur les tirages aléatoires est 1.5117031707412938
l'incertitude entre les tirages aléatoires est 0.030333073138613005
```

La conclusion à laquelle on aboutit est alors la suivante par exploitation de cette simulation Monte Carlo - L'indice optique du plexiglas est n=1,51 avec une incertitude type u(n)=0,03 (avec un seul chiffre pour l'incertitude) - L'indice optique du plexiglas est n=1,512 avec une incertitude type u(n)=0,031 (avec deux chiffres pour l'incertitude)

[19]:
```
n_plexiglas=1.5
Z=np.abs((moyenne_list_indice-n_plexiglas)/u_list_indice)
if Z<=2:
    print("le Z score est de",Z,", il est inférieur ou égale à 2")
    print("la mesure effectuée est cohérente avec la valeur théorique tabulée")
else :
    print("le Z score est de",Z,", il est supérieur à 2")
    print("la mesure effectuée n'est pas cohérente avec la valeur théorique␣
 ↪tabulée")
```

```
le Z score est de 0.385822125170563 , il est inférieur ou égale à 2
la mesure effectuée est cohérente avec la valeur théorique tabulée
```

[ ]: