OB01-TP

Projet 2 : Calcul de R et T pour deux dioptres

SEGHIR-DELAITRE Célibond

16/10/23

1 La methode analytique

1.1 Calcul pour un dioptre

1.1.1 Polarisation TE

Dans cette première partie, nous souhaitons optenir r, t, R et T, pour une configuration avec $n_1 = 1$ et $n_2 = 1.5$.

En utilisant les expressions données dans le sujet de TP, on peux obtenir ce code :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power



def chapitre_3_1_1():
    n1 = 1 #On defini les deux indices des deux milieux
    n2 = 1.5

    theta = np.linspace(0, np.pi/2, 200) #on cree un linspace pour
        theta

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
         #On defini wa comme etant une fonction, en veillant bien a
        ajouter pour 0j pour prendre en compte la racine carre de
        nombre negatif
    r_ab = lambda w_a, w_b: (w_a-w_b)/(w_a+w_b) #on defini une
        fonction r
    t_ab = lambda w_a, w_b : 2*w_a/(w_a+w_b) #on defini une fonction t

    rm_ab = lambda na, nb, w_a, w_b : ((nb**2)*w_a-(na**2)*w_b)/((nb
        **2)*w_a+(na**2)*w_b)
    tm_ab = lambda na, nb, w_a, w_b : np.sqrt((na**2)/(nb**2))*(2*(nb
        **2)*w_a)/((nb**2)*w_a+(na**2)*w_b)

    rad_to_deg = lambda rad: int(rad*180/np.pi) #On fait une fonction
        pour transformer les radians en degree

    r = r_ab(wa(n1, n1, theta), wa(n2, n1, theta)) #le systeme n'as qu
        'un seul dioptre donc il est r duit    r et    t
    t = t_ab(wa(n1, n1, theta), wa(n2, n1, theta))

    rm = rm_ab(n1, n2, wa(n1, n1, theta), wa(n2, n1, theta))
    tm = tm_ab(n1, n2, wa(n1, n1, theta), wa(n2, n1, theta))


    R = np.square(np.abs(r)) #On calcule la reflexion du systeme
```

```
33      T = np.square(np.abs(t))*(wa(n2, n1, theta)+np.conj(wa(n2, n1,
            theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta))) #On
            calcule la transmission du systeme
34
35      Rm = np.square(np.abs(rm))
36      Tm = np.square(np.abs(tm))*(wa(n2, n1, theta)+np.conj(wa(n2, n1,
            theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta)))
37
38      #plt.plot(theta, r)
39      #plt.plot(theta, t)
40      #plt.plot(theta, R)
41      plt.plot(theta, T)
42      plt.xlim([0, np.pi/2]) #On fait les bornes du graphique
43      degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]] #On change l
            'echelle de radians en degree
44      plt.xticks(plt.xticks()[0], labels=(degree_lab))
45      plt.show() # On affiche
```

Listing 1: Code 1 dioptre TE n_1:1 n_2 : 1.5

Ce qui nous permet d'obtenir ces courbes :

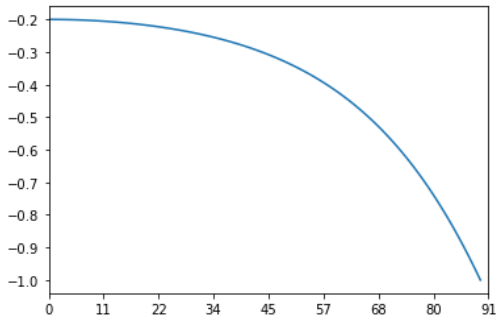

Figure 1: r - TE

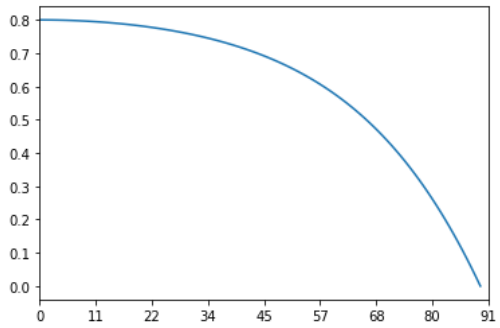

Figure 2: t - TE

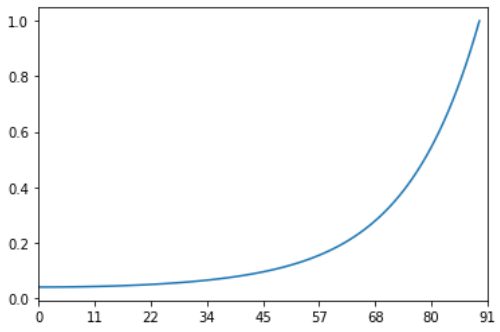

Figure 3: R - TE

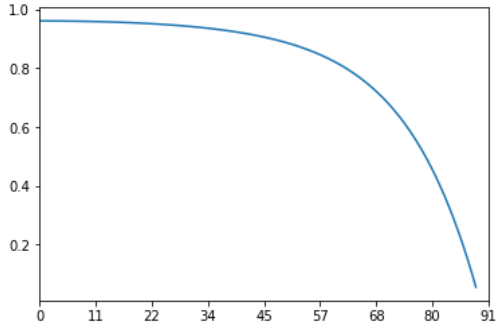

Figure 4: T - TE

Pour obtenir la configuration inverse, nous avons juste à changer les valeures de n_1 et n_2, ce qui donne :

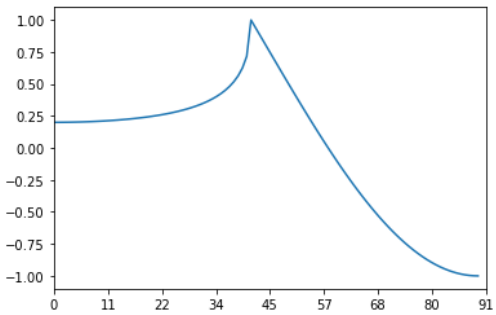

Figure 5: r - TE

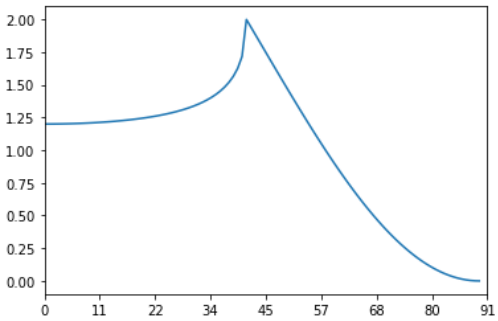

Figure 6: t - TE

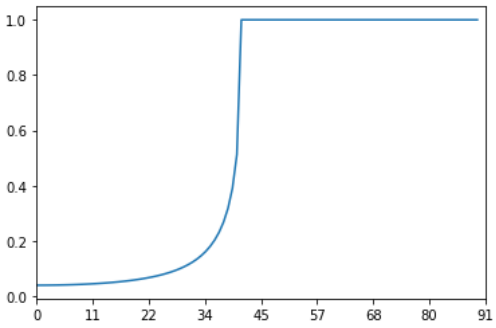

Figure 7: R - TE

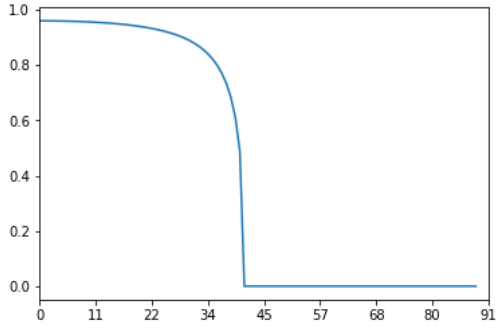

Figure 8: T - TE

Pour connaitre l'angle maximal de reflexion, on peux rajouter ce petit bout de code :

```
print(theta[R.tolist().index(max(R))]) #Cela permet de trouver l'index
    du maximum de la fonction R, puis de recupererl'angle en
   injectant l'index dans theta
print(T.tolist().index(min(T))) #C'est la meme chose avec le minimum
```

Listing 2: Code maximal R et minimal T

Ce qui donne bien un angle proche de 42° ce qui fait bien un sinus de 0.666 ce qui vaut bien à $\frac{n_2}{n_1}$

1.1.2 Polarisation TM

Pour la polarisation TM, il faut juste changer les formules et les fonctions, comme elles sont marquées dans l'énoncé de TD, ce qui donne à peut-près le même code :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power



def chapitre_3_1_1():
    n1 = 1 #On defini les deux indices des deux milieux
    n2 = 1.5

    theta = np.linspace(0, np.pi/2, 200) #on cree un linspace pour
        theta

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
         #On defini wa comme etant une fonction, en veillant bien a
        ajouter pour 0j pour prendre en compte la racine carre de
        nombre negatif

    rm_ab = lambda na, nb, w_a, w_b : ((nb**2)*w_a-(na**2)*w_b)/((nb
        **2)*w_a+(na**2)*w_b) #on defini une fonction r en
        polarisation TM
    tm_ab = lambda na, nb, w_a, w_b : np.sqrt((na**2)/(nb**2))*(2*(nb
        **2)*w_a)/((nb**2)*w_a+(na**2)*w_b) #on defini une fonction t
        en polarisation TM

    rad_to_deg = lambda rad: int(rad*180/np.pi) #On fait une fonction
        pour transformer les radians en degree


    rm = rm_ab(n1, n2, wa(n1, n1, theta), wa(n2, n1, theta))#le
        systeme n'as qu'un seul dioptre donc il est r duit    rm et
```

```
            tm
25      tm = tm_ab(n1, n2, wa(n1, n1, theta), wa(n2, n1, theta))
26
27
28      Rm = np.square(np.abs(rm))
29      Tm = np.square(np.abs(tm))*(wa(n2, n1, theta)+np.conj(wa(n2, n1,
            theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta)))
30
31      #plt.plot(theta, rm)
32      #plt.plot(theta, tm)
33      #plt.plot(theta, Rm)
34      plt.plot(theta, Tm)
35      plt.xlim([0, np.pi/2]) #On fait les bornes du graphique
36      degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]] #On change l
            'echelle de radians en degree
37      plt.xticks(plt.xticks()[0], labels=(degree_lab))
38      plt.show() # On affiche
```

Listing 3: Code 1 dioptre TM n_1:1 n_2 : 1.5

Ce qui donne ces courbes :

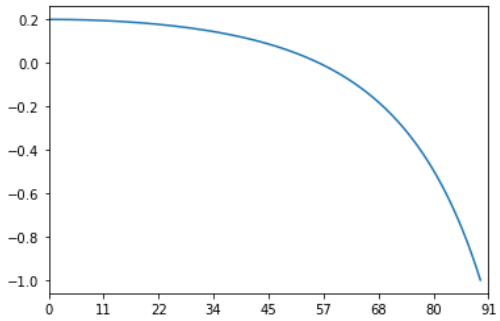

Figure 9: r - TM

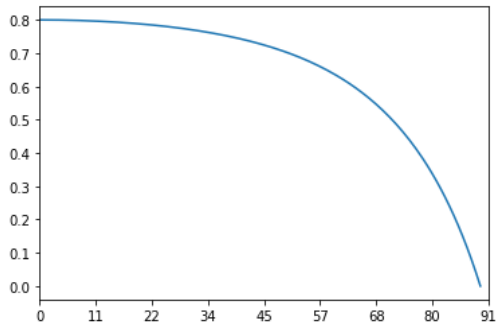

Figure 10: t - TM

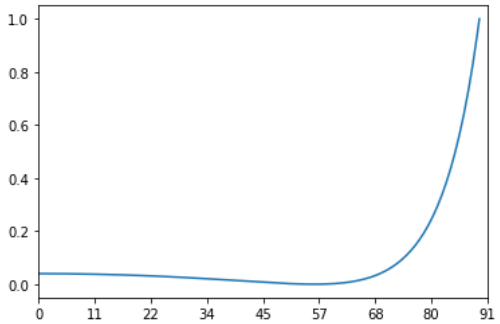

Figure 11: R - TM

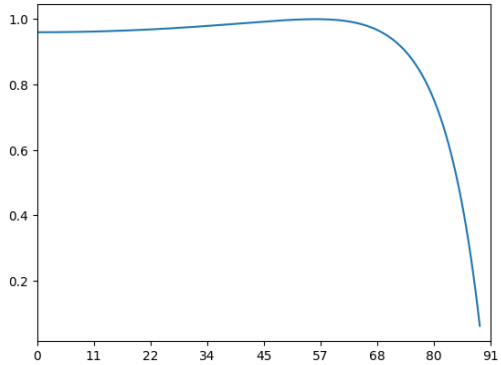

Figure 12: T - TM

On peux réutiliser le meme code que en TM pour trouver le minimum de R et le maximum de T en polarisation TM.

Ce qui nous donne nous donne un angle de 0.9866811097957893 radians ou 56.53266331658291

degree. Ce qui une fois mis dans la fonction $tan(x)$ nous donne bien $\frac{n_2}{n_1}$

Si on inverse les indices n_1 et n_2 on obtiens ces courbes

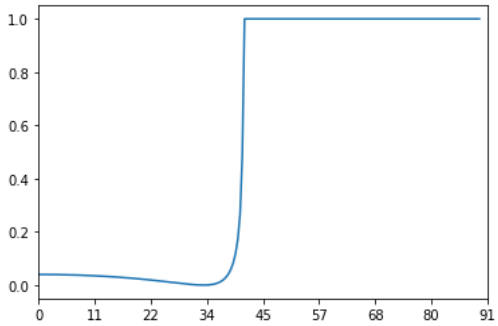

Figure 13: R - TM

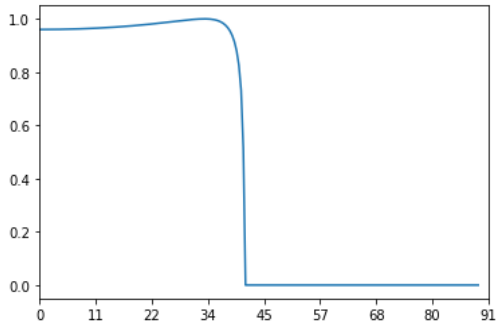

Figure 14: T - TM

Ici cela nous fais un angle de 0.5841152169991073 radians soit 33.46733668341709 ce qui nous fais bien, une fois passé la fonction tangente : $\frac{n_2}{n_1}$

1.2 Calcul pour deux dioptres

1.2.1 Epaisseur constante

On utilisant cette fois les formules pour un systeme avec deux dioptres, on obtient ce code là :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power



def chapitre_3_2_1():
    n1 = 1 #On defini les fariables fixes
    n2 = 1.5
    n3 = 1
    lmd = 500
    d = 1000

    theta = np.linspace(0, np.pi/2, 200)

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
    r_ab = lambda w_a, w_b: (w_a-w_b)/(w_a+w_b)
    t_ab = lambda w_a, w_b : 2*w_a/(w_a+w_b)

    f_r = lambda r1, r2, w: (r1+r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))/(1+
        r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))
```

```
    f_t = lambda t1, t2, r1, r2, w1, w2: (t1*t2*np.exp(1j*2*(np.pi/lmd
        )*d*(w1-w2)))/(1+r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w1)) #On
        construit la formule generalisee pour plusieurs dioptres

    rad_to_deg = lambda rad: int(rad*180/np.pi)

    r_12 = r_ab(wa(n1, n1, theta), wa(n2, n1, theta))
    r_23 = r_ab(wa(n2, n1, theta), wa(n3, n1, theta))


    r = f_r(r_ab(wa(n1, n1, theta), wa(n2, n1, theta)), r_ab(wa(n2, n1
        , theta), wa(n3, n1, theta)), wa(n2, n1, theta))
    t = f_t(t_ab(wa(n1, n1, theta), wa(n2, n1, theta)), t_ab(wa(n2, n1
        , theta), wa(n3, n1, theta)), r_ab(wa(n1, n1, theta), wa(n2,
        n1, theta)),
             r_ab(wa(n2, n1, theta), wa(n3, n1, theta)),  wa(n2, n1,
                 theta),  wa(n3, n1, theta))

    R = np.square(np.abs(r))
    T = np.square(np.abs(t))*(wa(n3, n1, theta)+np.conj(wa(n3, n1,
        theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta)))

    #plt.plot(theta, r)
    #plt.plot(theta, t)
    #plt.plot(theta, R)
    plt.plot(theta, T)
    plt.xlim([0, np.pi/2])
    degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]]
    plt.xticks(plt.xticks()[0], labels=(degree_lab))
    plt.show()
```

Listing 4: Code 2 dioptre TE n_1 et $n_3 : 1 | n_2 : 1.5$

Ce qui donne ces courbes :

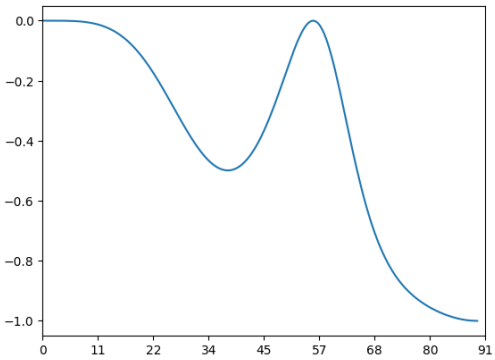

Figure 15: r - TE (2 dioptres)

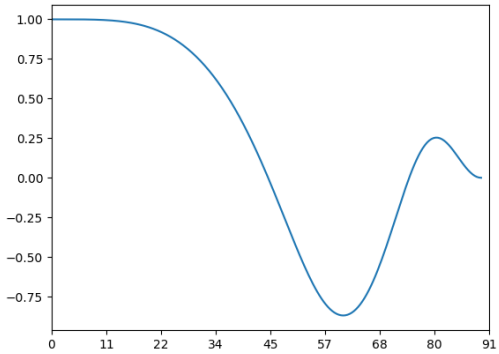

Figure 16: t - TE (2 dioptres)

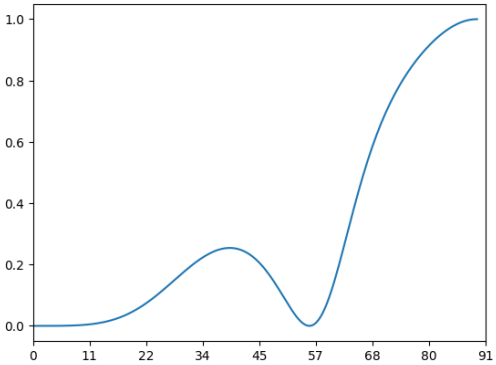

Figure 17: R - TE (2 dioptres)

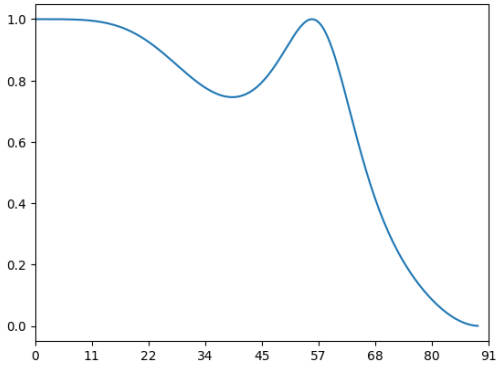

Figure 18: T - TE (2 dioptres)

1.2.2 Epaisseur Variable

C'est à peux prêt le meme code, sauf qu'il faut faire plusieurs r12, r23 et t12, t23, pour chaque épaisseurs, ce qui donne :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power



def chapitre_3_2_2():
    n1 = 1.5
    n2 = 1
    n3 = 1.5
    lmd = 720

    theta = np.linspace(0, np.pi/2, 200)

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
    r_ab = lambda w_a, w_b: (w_a-w_b)/(w_a+w_b)
    t_ab = lambda w_a, w_b : 2*w_a/(w_a+w_b)

    f_r = lambda r1, r2, w, d: (r1+r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))
        /(1+r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))
    f_t = lambda t1, t2, r1, r2, w1, w2, d: (t1*t2*np.exp(1j*2*(np.pi/
        lmd)*d*(w1-w2)))/(1+r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w1))

    rad_to_deg = lambda rad: int(rad*180/np.pi)


    r_10 = f_r(r_ab(wa(n1, n1, theta), wa(n2, n1, theta)), r_ab(wa(n2,
         n1, theta), wa(n3, n1, theta)), wa(n2, n1, theta), 10)
    t_10 = f_t(t_ab(wa(n1, n1, theta), wa(n2, n1, theta)), t_ab(wa(n2,
         n1, theta), wa(n3, n1, theta)), r_ab(wa(n1, n1, theta), wa(n2
        , n1, theta)),
            r_ab(wa(n2, n1, theta), wa(n3, n1, theta)),  wa(n2, n1,
                theta),  wa(n3, n1, theta), 10)

    r_100 = f_r(r_ab(wa(n1, n1, theta), wa(n2, n1, theta)), r_ab(wa(n2
        , n1, theta), wa(n3, n1, theta)), wa(n2, n1, theta), 100)
    t_100 = f_t(t_ab(wa(n1, n1, theta), wa(n2, n1, theta)), t_ab(wa(n2
        , n1, theta), wa(n3, n1, theta)), r_ab(wa(n1, n1, theta), wa(
        n2, n1, theta)),
            r_ab(wa(n2, n1, theta), wa(n3, n1, theta)),  wa(n2, n1,
                theta),  wa(n3, n1, theta), 100)

```

```
    r_1000 = f_r(r_ab(wa(n1, n1, theta), wa(n2, n1, theta)), r_ab(wa(
        n2, n1, theta), wa(n3, n1, theta)), wa(n2, n1, theta), 1000)
    t_1000 = f_t(t_ab(wa(n1, n1, theta), wa(n2, n1, theta)), t_ab(wa(
        n2, n1, theta), wa(n3, n1, theta)), r_ab(wa(n1, n1, theta), wa
        (n2, n1, theta)),
             r_ab(wa(n2, n1, theta), wa(n3, n1, theta)),  wa(n2, n1,
                 theta),  wa(n3, n1, theta), 1000)


    R = lambda r : np.square(np.abs(r))
    T = lambda t : np.square(np.abs(t))*(wa(n3, n1, theta)+np.conj(wa(
        n3, n1, theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta))
        )

    plt.plot(theta, T(t_10), 'b')
    plt.plot(theta, T(t_100), 'r', linestyle='--')
    plt.plot(theta, T(t_1000), 'g', linestyle='--')

    plt.xlim([0, np.pi/2])
    degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]]
    plt.xticks(plt.xticks()[0], labels=(degree_lab))
    plt.show()
```

Listing 5: Code 2 dioptre TE epaisseur variable

Ce qui donne cette courbe, avec en rouge la transmission pour une épaisseur de $10nm$, en bleu de $100nm$ et enfin en rouge de $1000nm$:

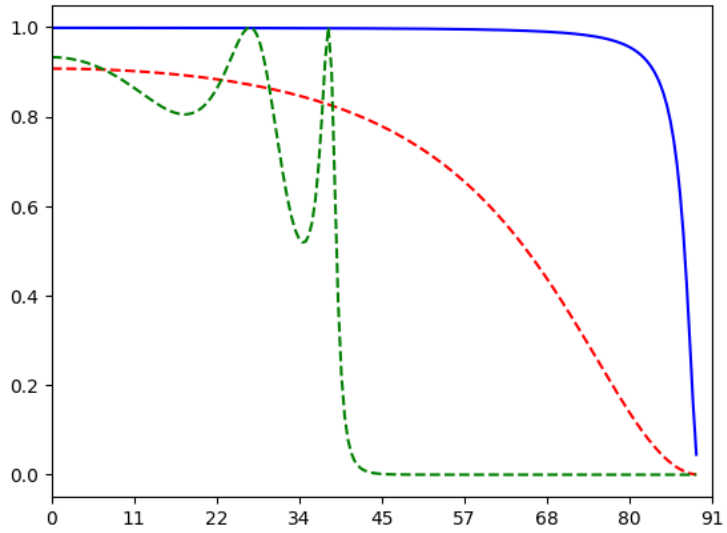

Figure 19: T en polarisation TE avec e variable

1.3 Excitation d'un plasmon de surface

Pour recalculer la reflexion et transmision de ce systeme à 3 indices, on peux utiliser le meme code mais adapté avec les formules de la polarisation TM :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power



def chapitre_3_3():
    n1 = 1.5 #On defini les indices, lambda et l'epaisseur
    n2 = 0.24*(1 + 12.875j)
    n3 = 1
    lmd = 500
    d = 40

    theta = np.linspace(0, np.pi/2, 200)

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
    r_ab = lambda w_a, w_b, na, nb: (w_a*nb**2-w_b*na**2)/(w_a*nb**2+
        w_b*na**2) #On utilise les formules de la polarisation TM pour
         r et t
    t_ab = lambda w_a, w_b, na, nb : np.sqrt(na**2/nb**2)*(2*nb**2*w_a
        /(w_a*nb**2+w_b*na**2))

    f_r = lambda r1, r2, w: (r1+r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))/(1+
        r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w))
    f_t = lambda t1, t2, r1, r2, w1, w2: (t1*t2*np.exp(1j*2*(np.pi/lmd
        )*d*(w1-w2)))/(1+r1*r2*np.exp(1j*2*(np.pi/lmd)*2*d*w1))

    rad_to_deg = lambda rad: int(rad*180/np.pi)

    r = f_r(r_ab(wa(n1, n1, theta), wa(n2, n1, theta), n1, n2), r_ab(
        wa(n2, n1, theta), wa(n3, n1, theta), n2, n3), wa(n2, n1,
        theta))
    t = f_t(t_ab(wa(n1, n1, theta), wa(n2, n1, theta), n1, n2), t_ab(
        wa(n2, n1, theta), wa(n3, n1, theta), n2, n3), r_ab(wa(n1, n1,
         theta), wa(n2, n1, theta), n1, n2),
    r_ab(wa(n2, n1, theta), wa(n3, n1, theta), n2, n3),  wa(n2, n1,
        theta),  wa(n3, n1, theta))

    R = np.square(np.abs(r))
    T = np.square(np.abs(t))*(wa(n3, n1, theta)+np.conj(wa(n3, n1,
        theta)))/(wa(n1, n1, theta)+np.conj(wa(n1, n1, theta)))

```

```
    #plt.plot(theta, np.abs(r))
    #plt.plot(theta, np.abs(t))
    #plt.plot(theta, R)
    #plt.plot(theta, T)
    plt.plot(theta, R+T)
    plt.xlim([0, np.pi/2])
    degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]]
    plt.xticks(plt.xticks()[0], labels=(degree_lab))
    plt.show()
```

Listing 6: Code 2 dioptres TM argent

Ce qui donne :

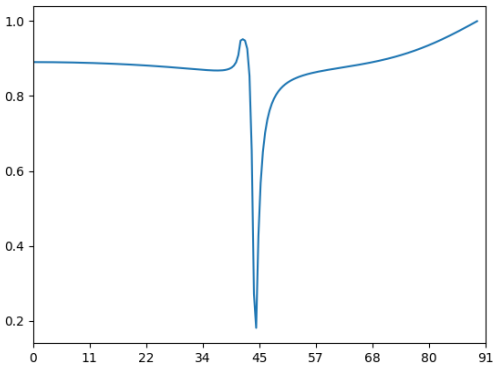

Figure 20: $|r|$ - TM (Argent)

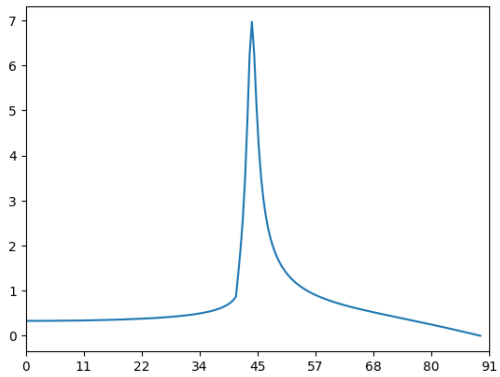

Figure 21: $|t|$ - TM (Argent)

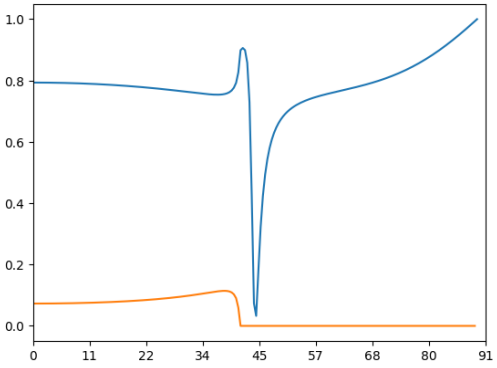

Figure 22: R (bleu) et T(orange) - TM (Argent)

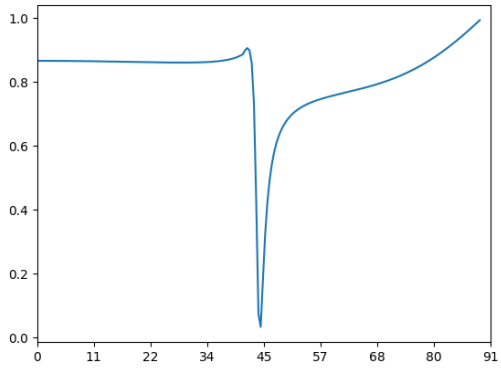

Figure 23: R+T - TM (Argent)

On peux faire varier l'epaisseur grâce à un *np.linspace* puis faire les opération avec meshgrid, ce qui nous donne une projection 3D, de —t—. On peux aussi projeter cette image sur le plan XY pour faire comme sur l'énoncé.

C'est une basique modification du code qui donne :

```
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power


   def chapitre_3_3_2():
    n1 = 1.5
    n2 = 0.24*(1 + 12.875* 1j)
    n3 = 1
    lmd = 500

    d = np.linspace(10, 100, 1000) #On cree un linspace pour d
    theta = np.linspace(0, np.pi/2, 1000)

    THETA, D = np.meshgrid(theta, d) #On utilise meshgrid, pour
        faciliter les operations matricielles

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
    r_ab = lambda w_a, w_b, na, nb: (w_a*nb**2-w_b*na**2)/(w_a*nb**2+
        w_b*na**2)
    t_ab = lambda w_a, w_b, na, nb : np.sqrt(na**2/nb**2)*(2*nb**2*w_a
        /(w_a*nb**2+w_b*na**2))

    f_r = lambda r1, r2, w: (r1+r2*np.exp(1j*2*(np.pi/lmd)*2*D*w))/(1+
        r1*r2*np.exp(1j*2*(np.pi/lmd)*2*D*w))
    f_t = lambda t1, t2, r1, r2, w1, w2: (t1*t2*np.exp(1j*2*(np.pi/lmd
        )*D*(w1-w2)))/(1+r1*r2*np.exp(1j*2*(np.pi/lmd)*2*D*w1))

    rad_to_deg = lambda rad: int(rad*180/np.pi)


    r = f_r(r_ab(wa(n1, n1, THETA), wa(n2, n1, THETA), n1, n2), r_ab(
        wa(n2, n1, THETA), wa(n3, n1, THETA), n2, n3), wa(n2, n1,
        THETA))
    t = f_t(t_ab(wa(n1, n1, THETA), wa(n2, n1, THETA), n1, n2), t_ab(
        wa(n2, n1, THETA), wa(n3, n1, THETA), n2, n3), r_ab(wa(n1, n1,
         THETA), wa(n2, n1,THETA), n1, n2),
    r_ab(wa(n2, n1, THETA), wa(n3, n1, THETA), n2, n3),  wa(n2, n1,
        THETA),  wa(n3, n1, THETA))


```

```
35      fig, ax = plt.subplots(subplot_kw={"projection": "3d"}) #On defini
            une figure
36
37
38
39      surf = ax.plot_surface(THETA, D,np.abs(t), cmap='plasma',
40                             linewidth=0, antialiased=False, alpha=0.4,
                                   rcount=300, ccount=300) #On affiche la
                                   surface en la rendant un peux transparante
                                   et une resolution de 300x300 carres
41      ax.contourf(THETA, D, np.abs(t), levels=50, zdir='z', offset=0,
            cmap='plasma', alpha = 0.8) #On cree le graphique 2D sur l'axe
             XY, avec 50 niveaux et precision et un peu moins transparante
42      plt.xlim([0, np.pi/2])
43      degree_lab = [rad_to_deg(i) for i in plt.xticks()[0]]
44      plt.xticks(plt.xticks()[0], labels=(degree_lab))
45      plt.savefig("abs_t_3d.svg") #On enregistre au format svg pour
            garder le maximum de qualite
46      plt.savefig("abs_t_3d.png", format="png", dpi=800)
47      plt.show()
```

Listing 7: Code 1 dioptre TM n_1:1 n_2 : 1.5

Malheuresment, mon éditeur ne suporte pas le .svg donc nous allons nous contenter d'un fichier de type bitmap, ce qui donne cette belle courbe:

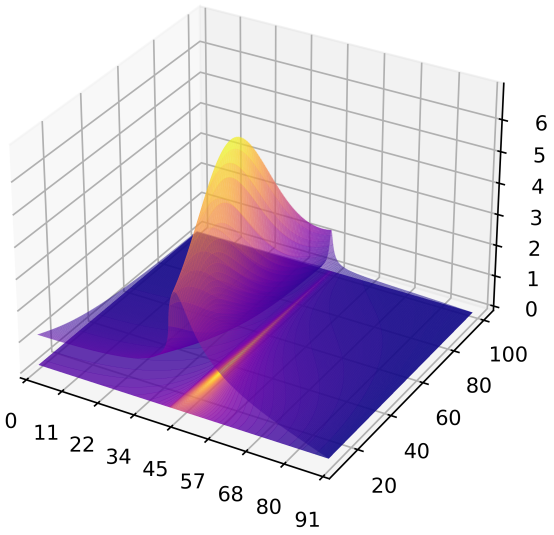

Figure 24: $|t|$ en 3d

Pour avoir le maximum de —t—, on peux rajouter ces lignes aux programme :

```python
import numpy as np

 print(np.max(np.abs(t))) #Cela nous permet d'avoir le max de |t|
print(np.where(np.abs(t)==np.max(np.abs(t)))) #Cela nous permet de d'
    avoir les coordonnees d et theta pour le max d'|t|
d_max, theta_max = np.where(np.abs(t)==np.max(np.abs(t)))
print(d[d_max[0]], theta[theta_max[0]]) #On cherche quel sont les
    valeures qui correspondent aux coordonnees ce qui donne d et theta
     pour |t| max
```

Listing 8: Code 1 dioptre TM n_1:1 n_2 : 1.5

Grâce à ce programme, on obtient les valeures de 6.969457060908964 pour $|t|$ max ce qui correspond à $d = 39.90990990990991$ nm et $\theta = 0.7814672416587223$ rad ou 44.77477477477477 degrées.

2 La methode matricielle

Avec la methode matricielle, il faut faire très attention, car la bibliothèque *numpy* fais parfois du calcul matriciel pur, comme le produit scalaire, le produit matriciel, mais aussi parfois du calcul scalaire, comme le fait de mettre tous les élements d'une matrice au carré.

Il faut juste faire très attention à la notation et ne pas se tromper. Ce qui une fois après recopier et transformer les formules en ligne de code donne ceci :

```python
import numpy as np
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from numpy.linalg import inv, matrix_power


def chapitre_4():
    n1 = 1.5 #On fixe les indice puis theta
    n2 = 1
    N = 2 #On aura plus qu'a changer le N pour les differentes courbes
    theta =  np.pi/4

    sigmad = np.linspace(0.48, 0.66, 200) # On fais un linspace avec
        sigma*d ce qui repressente d/lambda

    wa = lambda na, ni, theta: np.sqrt(na**2-(ni*np.sin(theta))**2+0j)
    Ma = lambda wa: np.array([[1, 1], [-wa, wa]]) #On defini les
        calcul de matricie
    Pa = lambda wa, sigmad: np.array([[np.exp(1j*2*np.pi*wa*sigmad),
        0], [0, np.exp(-1j*2*np.pi*wa*sigmad)]])
```

```
20
21
22      M1 = Ma(wa(n1, n1, theta))
23      M2 = Ma(wa(n2, n1, theta))
24      P1_list = [Pa(wa(n1, n1, theta), i) for i in sigmad] #On defnin
            des liste de matrice pour pouvoir tracer une courbe
25      P2_list = [Pa(wa(n2, n1, theta), i) for i in sigmad]
26
27      Mtot_list = [inv(M1) @ M2 @ inv(P2_list[i]) @ inv(M2) @
            matrix_power((M1 @ inv(P1_list[i]) @ inv(M1) @ M2 @ inv(
            P2_list[i]) @ inv(M2)), (N-1)) @ M1 for i in range(len(sigmad)
            )] #On calcul Mtot en fesant attention d'utiliser une fonction
             inv() pour faire l'inverse, l'@ pour faire les produits
            matriciels et matrix_power() pour mettre    la puissance tous
            les elements d'une matrice
28
29      list_r = [x[1][0]/x[0][0] for x in Mtot_list] #On fait la liste
            des r
30
31
32      plt.plot(sigmad, np.square(np.abs(list_r))) #On affiche la
            reflexion, ici grand R. np.square() permet de mettre tous les
            elements d'une matrice au carre, c'est la verision limitee de
            matrix_power().
```

Listing 9: Code 1 dioptre TM n_1:1 n_2 : 1.5

Ce qui nous donne ces courbes :

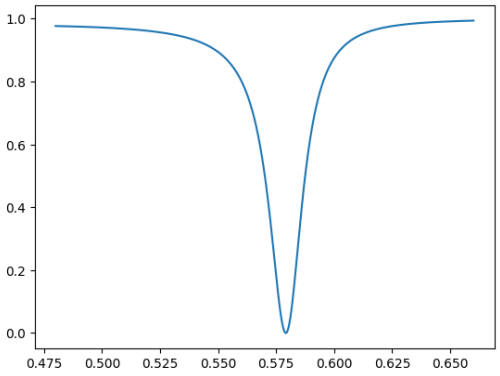

Figure 25: R pour N = 2

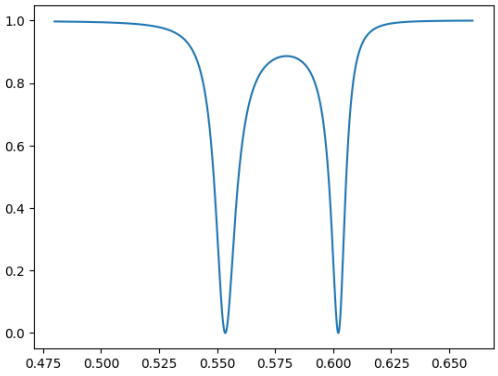

Figure 26: R pour N = 3

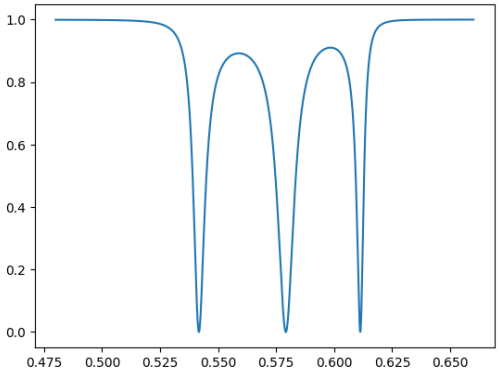

Figure 27: R pour N = 4

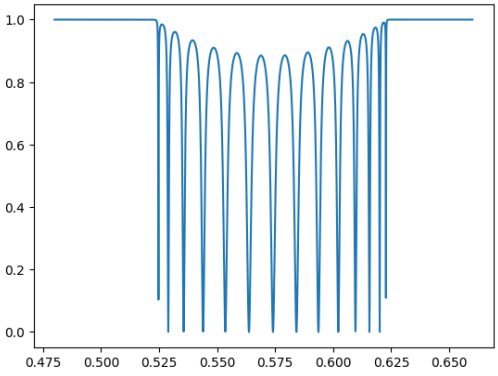

Figure 28: R pour N = 15