

PHYSIQUE ATOMIQUE ET NUCLÉAIRE
DEVOIR #4 - NUMÉRO 5
CODE PYTHON

class PB:

def __init__(self):

self.A = 208

Nombre de masse

self.Z = 82

Nombre atomique

self.Vo = 50

MeV

self.hw = 41 * self.A**(-1/3)

MeV

self.D = 0.0225 * self.hw

self.C = 0.1 * self.hw

self.protons = [

couches complétées pour 82 protons

[1, 0, 0.5],

[1, 1, 1.5],

[1, 1, 0.5],

[1, 2, 2.5],

[2, 0, 0.5],

[1, 2, 1.5],

[1, 3, 3.5],

[2, 1, 1.5],

[1, 3, 2.5],

[2, 1, 0.5],

[1, 4, 4.5],

[2, 2, 2.5],

[1, 4, 3.5],

[3, 0, 0.5],

[2, 2, 1.5],

[1, 5, 5.5],

82 protons

]

self.neutrons = [

couches complétées pour 126 neutrons

[1, 0, 0.5],

[1, 1, 1.5],

[1, 1, 0.5],

[1, 2, 2.5],

[2, 0, 0.5],

[1, 2, 1.5],

[1, 3, 3.5],

[2, 1, 1.5],

[1, 3, 2.5],

[2, 1, 0.5],

[1, 4, 4.5],

[2, 2, 2.5],

[1, 4, 3.5],

[3, 0, 0.5],

[2, 2, 1.5],

[1, 5, 5.5],

82 neutrons

[2, 3, 3.5],

[1, 5, 4.5],

[1, 6, 6.5],

```

        [3, 1, 1.5],
        [2, 3, 2.5],
        [3, 1, 0.5]
    ]
    # 126 neutrons

    def Enlsj(self,n,l,s,j):
        # Énergie d'un nucléon selon nlsj
        return - self.Vo + self.hw * (2 * (n - 1) + l + 3/2) - self.D * l * (l + 1) - self.C * (j * (j + 1) - l * (l + 1) - s * (s + 1)) / 2

    def bindingEnergy(self):
        # Énergie de liaison
        protonsEnergy = 0
        for shell in self.protons:
            n, l, s, j = shell[0], shell[1], 0.5, shell[2]
            protonsEnergy += self.Enlsj(n, l, s, j) * (2 * j + 1)
            # Ajout de l'énergie d'un proton multipliée par le degré de dégénérescence
        neutronsEnergy = 0
        for shell in self.neutrons:
            n, l, s, j = shell[0], shell[1], 0.5, shell[2]
            neutronsEnergy += self.Enlsj(n, l, s, j) * (2 * j + 1)
            # Ajout de l'énergie d'un neutron multipliée par le degré de dégénérescence
        return - (protonsEnergy + neutronsEnergy)
        # Somme de toutes les énergies

pb = PB()
print('The binding energy is', pb.bindingEnergy(), 'MeV.')

```

Output :

The binding energy is 3422.517322322542 MeV.

PHYSIQUE ATOMIQUE ET NUCLÉAIRE
DEVOIR #4 - NUMÉRO 3
CODE PYTHON

```
import numpy as np
from matplotlib import pyplot as plt
import scipy.special as ss
import scipy
```

```
def f(x):
    return x
```

```
def f0(x, Omega2):
    return - (1 - x**2)**0.5 * (np.tan((Omega2 * (1 - x**2))**0.5))**-1
```

```
def F(L, x, Omega2):
    a = scipy.sqrt((Omega2 * (1 - x**2)))
    b = 1j * Omega2**0.5 * x
    coeff = -1j * scipy.sqrt((1 - x**2))
    firstTerm = ss.spherical_jn(L - 1, a) / ss.spherical_jn(L, a)
    secdTerm = (ss.spherical_jn(L, b) + 1j * ss.spherical_yn(L, b)) / (ss.spherical_jn(L - 1, b) + 1j * ss.spherical_yn(L - 1, b))
    return coeff * firstTerm * secdTerm
```

```
# Omega^2 values
```

```
singlet = 2.23 * 4
```

```
triplet = 3.57 * 4
```

```
# w values knowing w range is [0,1]
```

```
w = np.array([(i + 1)/100 for i in range(99)])
```

```
# left term of the transcendant equation
```

```
f = np.array([f(i) for i in w])
```

```
# right term of the transcendant equation
```

```
f_00 = np.array([f0(i, singlet) for i in w]) # when (L,S) = (0,0)
```

```
f_01 = np.array([f0(i, triplet) for i in w]) # when (L,S) = (0,1)
```

```
F_10 = np.array([F(1,i, singlet) for i in w]) # when (L,S) = (1,0)
```

```
F_11 = np.array([F(1,i, triplet) for i in w]) # when (L,S) = (1,1)
```

```
# when (L,S) = (0,0)
```

```
plt.figure()
```

```
plt.plot(w, f)
```

```
plt.plot(w, f_00)
```

```
idx = np.argwhere(np.diff(np.sign(f_00 - f))).flatten() # find
```

```
print('intersection', w[idx])
```

```
plt.plot(w[idx], f[idx], 'ro')
```

```
plt.ylim([-1, 10])
```

```
plt.savefig('00', bbox_inches = 'tight')
```

```
# when (L,S) = (0,1)
```

```
plt.figure()
```

```
plt.plot(w, f)
```

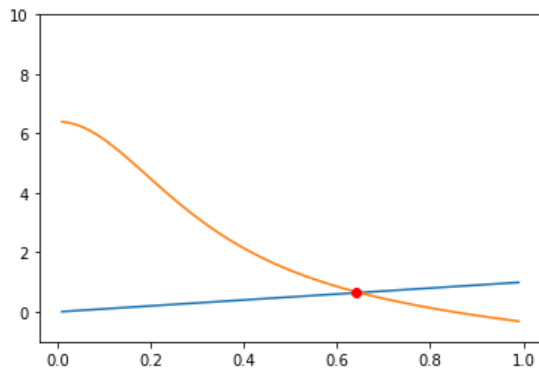
```
plt.plot(w,f_01)
idx = np.argwhere(np.diff(np.sign(f_01 - f))).flatten()
print('intersection', w[idx])
plt.plot(w[idx], f[idx], 'ro')
plt.ylim([-1,10])
plt.savefig('01', bbox_inches = 'tight')
```

```
# when (L,S) = (1,0)
plt.figure()
plt.plot(w,f)
plt.plot(w,np.real(F_10))
idx = np.argwhere(np.diff(np.sign(F_10 - f))).flatten()
print('intersection', w[idx])
plt.plot(w[idx], f[idx], 'ro')
plt.ylim([-10,10])
plt.savefig('10', bbox_inches = 'tight')
```

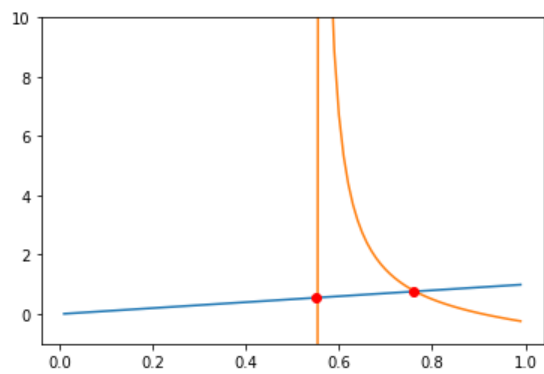
```
# when (L,S) = (1,1)
plt.figure()
plt.plot(w,f)
plt.plot(w,np.real(F_11))
idx = np.argwhere(np.diff(np.sign(F_11 - f))).flatten()
print('intersection', w[idx])
plt.plot(w[idx], f[idx], 'ro')
plt.ylim([-1,10])
plt.savefig('11', bbox_inches = 'tight')
```

Output :

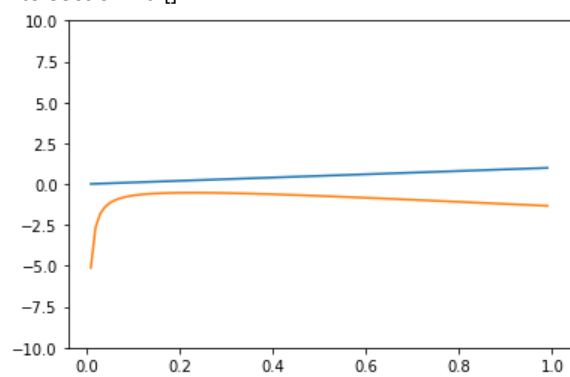
intersection 00 [0.6477]



intersection 01 [0.5557 0.7636]



intersection 10 []



intersection 11 [0.4319]

