FRUH Arthur

# TP Physique Quantique

## 1 Particule dans un puits avec double marche de potentiel

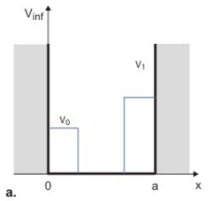



FIGURE 1: marches de potentiel dans un puits infini

On considère la situation représentée sur la Figure 1 où une particule de masse $m$ est confinée dans un puits de potentiel infini de largeur $a$ dans lequel deux marches de potentiels sont ajoutées dans le puits, l'une de hauteur $V_0$ entre $x = 0$ et $x = 0.3a$ et l'autre de hauteur $V_1$ entre $x = 0.7a$ et $x = a$
On introduira $\nu_0 = \frac{V_0}{E_1^0}$ et $\nu_1 = \frac{V_1}{E_1^0}$, et l'on prendra $a = 1$, $\nu_0 = 2000$ et $\nu_1 = 4000$

1. Pour déterminer les fonctions propres des trois premiers niveaux, il faut déterminer l'Hamiltonien du problème. Le potentiel normalisé s'écrit :

$$\nu(x) = \begin{cases} \nu_0 & \text{si } x \in [0, 0.3a] \\ 0 & \text{si } x \in [0.3a, 0.7a] \\ \nu_1 & \text{si } x \in [0.7, a] \\ +\infty & \text{sinon} \end{cases} \tag{1}$$

Par le travail préparatoire, on a montré que les coefficients $h_{n_m}$ s'écrivent :

$$h_{nm} = n^2\delta_{nm} + \frac{2}{a}\int_0^a \sin(n\pi\frac{x}{a})\nu(x)\sin\Big(m\pi\frac{x}{a}\Big)dx \tag{2}$$

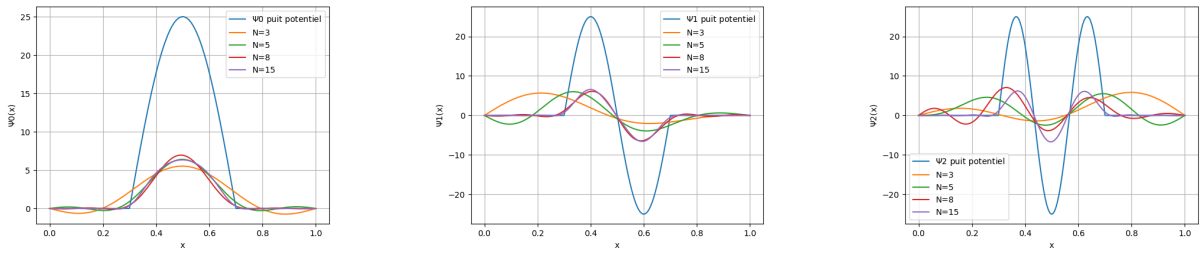
En diagonalisant la matrice $H$ et en prenant la base des $\{|\varphi_n\rangle\}$ on trouve les fonctions propres :

$$\Psi_n(x) = \langle x|\Psi_n\rangle \tag{3}$$

Pour des faibles niveaux d'énergies on peut faire l'hypothèse que la particule se comportera comme si elle était dans un puits infini entre $x = 0.3a$ et $X = 0.7a$

$$\Psi_n^{Th}(x) = \begin{cases} \sqrt{\frac{2}{0.4}}\sin\frac{(x-0.3)n\pi}{0,4} & \text{si } x \in [0.3a, 0.7a] \\ 0 & \text{sinon} \end{cases} \tag{4}$$

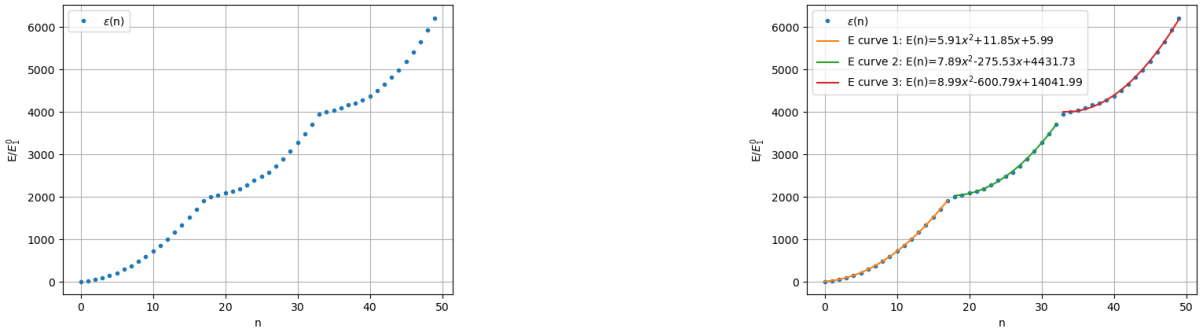
La Figure 2. montre que l'on peut approximer de façon correcte les fonctions $\psi$ de niveaux d'énergie faibles par les fonctions correspondantes pour un puits de potentiel infini centrée en x=0,5 et de largeur a'=0,4, i.e. à l'endroit du puits où le potentiel est nul. Ces approximations sont correctes pour N suffisamment grand, évitant les artefacts si le niveaux d'énergie est proche de N.




(a) $\Psi_0(x)$ (b) $\Psi_1(x)$ (c) $\Psi_2(x)$

FIGURE 2: tracé des $\Psi(x)$ pour différents N et comparaison avec les fonctions trouvées analytiquement

2. et 3. En ordonnant dans l'ordre croissant les valeurs propres de $H$, on a directement les valeurs de $\varepsilon(n)$. On peut alors en faire un tracé comme dans la Figure 3.




(a) tracé de $\varepsilon(n)$ (b) "fit" de $\varepsilon(n)$

FIGURE 3: tracé des valeurs $\varepsilon(n)$

On remarque trois zones sur le graphes correspondant chacun à une parabole. On peut analyser que pour des faibles valeurs de $n$, la particule est confiné entre les potentiels $V_0$ et $V_1$, lorsque l'énergie est suffisante la particule peut accéder à l'espace occupé par le potentiel $V_0$. Finalement pour une très grande énergie la particule peut aussi accéder à l'espace du potentiel $V_1$, on retrouve presque le cas du puits infini entre 0 et $a$ .
On peut faire l'hypothèse que chaque branche de la courbe $\varepsilon(n)$ correspond à un puits de potentiel infini différent (Figure 4.). Pour vérifier cette hypothèse on va modéliser chacune des zones 1,2 ou 3 par : $\varepsilon_i^{approx} = an^2 + bn + C_i$

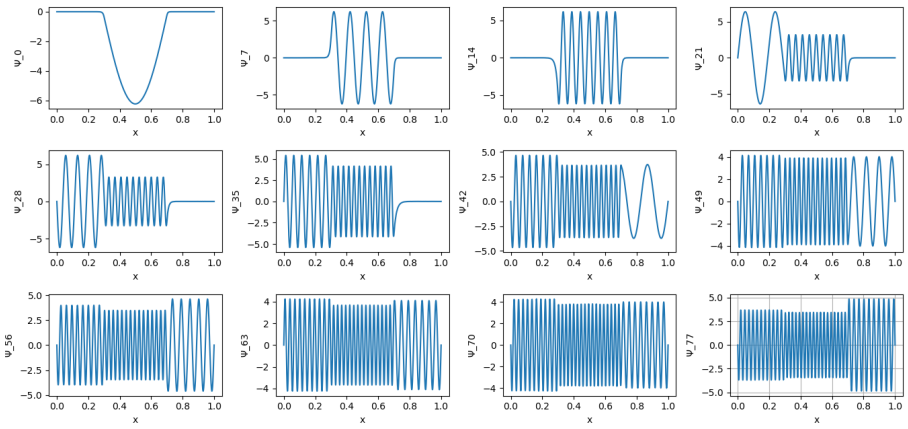



FIGURE 4: différents tracés de $\psi_i$ pour différents niveaux d'énergie i

# 2 Résonance Paramagnétique Electronique dans le rubis

1. Dans ce problème (le cristal $Al_2O_3$ avec $0,1\%$ de $Cr^{3+}$, spin total S=3/2), l'Hamiltonien s'écrit :

$$H = -\frac{2\pi D}{\hbar}(S_z^2 - \frac{S^2}{3}) + g\frac{eB_0}{2m}(S_z\cos\theta + S_x\sin\theta)$$

- $-D(S_z^2 - \frac{S^2}{3})$ : terme de champ cristallin. Ici l'axe z correspond à l'axe c du cristal.On donne D=5.73 GHz en unité de fréquence
- $g\frac{eB_0}{2m}(S_z\cos\theta + S_x\sin\theta)$ : le terme correspondant à l'effet Zeeman qui dépend de $\theta$,l'angle entre l'axe z et le champ $\vec{B_0}$.

On écrit ensuite les différentes matrice $S_x$, $S_z$ et $S^2$ dans la base $|S, m_S\rangle$ :
On sait que $S = \frac{3}{2}$ et $m \in \{\frac{3}{2}; \frac{1}{2}; -\frac{1}{2}; -\frac{3}{2}\}$. Comme $\langle S', m'|S_z|S, m|S', m'|S_z|S, m\rangle = m\hbar\delta_{S,S'}\delta_{m,m'}$ et $\langle S', m'|S^2|S, m|S', m'|S^2|S, m\rangle = S(S+1)\hbar^2\delta_{S,S'}\delta_{m,m'}$. On en déduit les matrices :

$$S_z = \frac{\hbar}{2}\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -3 \end{pmatrix} \qquad S^2 = \frac{15\hbar^2}{4}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pour calculer $S_x$ et $S_y$, on calcule d'abord $S_+$ et $S_-$.
On a $\langle S, m|S_+|S', m|S, m|S_+|S', m\rangle = \hbar\sqrt{S(S+1) - m(m+1)}\delta_{S,S'}\delta_{m,m'+1}$
et $\langle S', m'|S_-|S, m|S', m'|S_-|S, m\rangle = \hbar\sqrt{S(S+1) - m(m-1)}\delta_{S,S'}\delta_{m,m'-1}$.

$$S_+ = \hbar\begin{pmatrix} 0 & \frac{\sqrt{12}}{2} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{12}}{2} \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad S_- = \hbar\begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{\sqrt{12}}{2} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{12}}{2} & 0 \end{pmatrix}$$

On a ensuite $S_x = \frac{S_+ + S_-}{2}$ et $S_y = \frac{S_+ - S_-}{2}$.

$$S_x = \hbar\begin{pmatrix} 0 & \frac{\sqrt{12}}{2} & 0 & 0 \\ \frac{\sqrt{12}}{2} & 0 & 1 & 0 \\ 0 & 1 & 0 & \frac{\sqrt{12}}{2} \\ 0 & 0 & \frac{\sqrt{12}}{2} & 0 \end{pmatrix} \qquad S_y = \hbar\begin{pmatrix} 0 & \frac{\sqrt{12}}{2} & 0 & 0 \\ -\frac{\sqrt{12}}{2} & 0 & 1 & 0 \\ 0 & -1 & 0 & \frac{\sqrt{12}}{2} \\ 0 & 0 & -\frac{\sqrt{12}}{2} & 0 \end{pmatrix}$$

La matrice H s'écrit finalement dans la base $|S, m_S\rangle$ :

$$H = \begin{pmatrix} -2\pi\hbar D - \frac{3geB_0\hbar}{4m}cos(\theta) & \frac{\sqrt{12}geB_0\hbar}{8m}sin(\theta) & 0 & 0 \\ \frac{\sqrt{12}geB_0\hbar}{8m}sin(\theta) & 2\pi\hbar D - \frac{geB_0\hbar}{8m}cos(\theta) & \frac{geB_0\hbar}{2m}sin(\theta) & 0 \\ 0 & \frac{geB_0\hbar}{2m}sin(\theta) & 2\pi\hbar D - \frac{geB_0\hbar}{8m}cos(\theta) & \frac{\sqrt{12}geB_0\hbar}{8m}sin(\theta) \\ 0 & 0 & \frac{\sqrt{12}geB_0\hbar}{8m}sin(\theta) & -2\pi\hbar D - \frac{3geB_0\hbar}{4m}cos(\theta) \end{pmatrix}$$

2. (a) Pour un angle $\theta = 50°$, on obtient la figure 5. comme représentation des énergies en fonctions du champs $B_0$
Le spectromètre RPE disponible en travaux pratiques à une cavité telle que sa fréquence de résonance est $f_{cav} = 9.19GHz$. En calculant les valeurs, on trouve des transitions possibles sur le spectre RPE pour $B_0 = [482G, 1837G, 2209G, 4131G]$

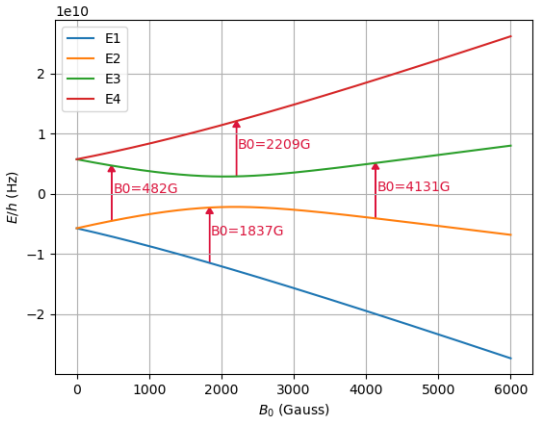



FIGURE 5: niveau d'énergie et les transitions calculées numériquement pour $\theta = 50°$

(b) On détermine toutes les transitions possibles correspondant à la fréquence de la cavité en fonction de l'angle $\theta \in [0°, 190°]$ et du champ $B_0 \in [0G, 8000G]$(on a augmenté la plage de $B_0$ pour ne pas couper la courbe)

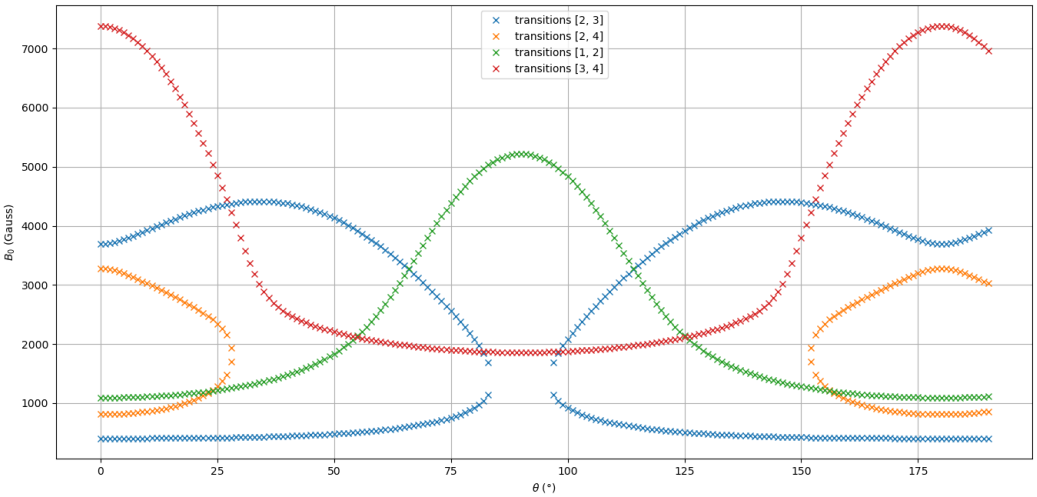



FIGURE 6: Graphe des transitions possibles pour $\theta \in [0°, 190°]$

On remarque que les transitions sont $\pi$-périodique.

(c) On peut comparer les valeurs trouvées expérimentalement et celles calculées numériquement ($B_0 \in [0G, 6000G]$) en se limitant à la plage angulaire $[0°, 90°]$.

Les valeurs sont assez cohérente si ce n'est que l'on calcule pour chaque $\theta$ (excepté $\theta = 40°$) une transition non mesurée expérimentalement.Une hypothèse est que l'on mesure seulement une des deux transitions, quand leurs valeurs de $B_0$ sont très proches.
De plus on peut connaître la nature de chacune des transitions :

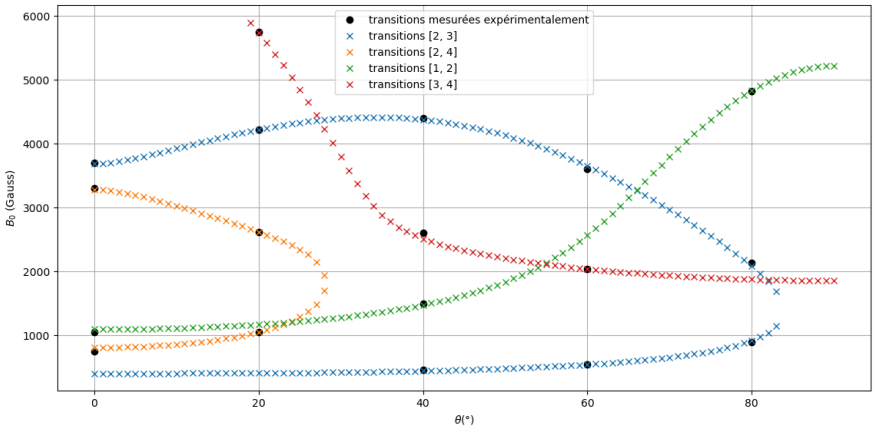



FIGURE 7: Graphe des transitions possibles (calcul numérique) et des transitions mesurées

- Elles sont des transitions $E_i \to E_{i+1}$ ce qui était attendu avec la condition $\Delta m_s = \pm 1$ sur les transitions entre états de spin.
- Pour $\theta = 0°$ et $\theta = 20°$, il exite pour chaque angle deux transitions $E_2 \to E_4$. Cela peut être dû aux photons piégés qui excitent plusieurs fois les électrons avant de s'échapper du cristal.

# Appendices

```
#### Annexe :TP de Physique Quantique
from math import *
import numpy as np
from scipy.integrate import *
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.constants import hbar,h

#### Partie A
##fonction Hamiltonien
def v(x,a,v0,v1):
    """calcul le potentiel harmonique normalisé et adimensioné"""
    if x>=0 and  x<=0.3*a:
        return v0
    elif x>=0.7*a and x<=a:
        return v1
    elif x>0.3*a and x<0.7*a:
        return 0

def h_coeff(n,m) :
    """fonction qui calcul les coefficients de l'hamiltonien normalisé et adimensionés,
    prend en argument la ligne n et la colonne m"""
    def Integrande(x):
        return np.sin(np.pi*n*x/a)*v(x,a,v0,v1)*np.sin(np.pi*m*x/a)
    coef, eps= quad(Integrande,0,a)
    coef*=2/a
    if n==m:
        coef+=n**2
    #si le coefficient trouvé est plus petit que la précision, on le considère nul
    if abs(coef)<abs(eps):
        return 0
    return coef

def calc_H(N):
    """renvoi matrice H"""
    H=np.zeros((N,N),dtype=float)
    for m in range(1,N+1):
        for n in range(1,N+1):
            H[n-1,m-1]=h_coeff(n,m)
    return H

def vecteurpropre(H):
    """fontion qui renvoie les vecteurs propres correspondants aux energies 0,1 et 2"""
    #on récupère les valeurs propres et vecteurs propres de H,
    #ces valeurs propres sont toutes les valeurs possibles pour l'énergie
    vp,vect=np.linalg.eig(H)
    vect = np.transpose(vect)
    index = vp.argsort()
    return vp[index], vect[index]

##fonction Psi
def phiHarm(x,n,a):
    """retourne la fonction phi_harmonique en x"""
    return np.square(2/a)*np.sin(np.pi*(n+1)*x/a)

def prodScal(x,N,a,vectPropre,s):
```

```
    """calcul de <x|psi>"""
    ps=0
    for n in range(N):
        ps+=phiHarm(x,n,a)*vectPropre[n]*s
    return ps

##Parametres
a=1
v0=2000
v1=4000

##Energie
N=50
H1=calc_H(N)
epsilon ,vect = vecteurpropre(H1)

#graphe des énergies
plt.figure(0)
X=np.arange(50)
plt.plot(X,epsilon,label ='$\epsilon$(n)',marker='.',linestyle='')
plt.xlabel("n")
plt.ylabel("E/$E^{0}_{1}$")
plt.legend()
plt.grid()
plt.show()

#modélisation eps=n^2+C
E_Q1=[]# liste des énergies quadratiques
C1=0
for k in range(0,18):
     E_Q1.append(((k)**2)/.4**2+C1)
for k in range(0,18):
    C1+=abs(((k)**2)/.4**2-epsilon[k])/N
print(C1)
for k in range(0,18):
     E_Q1[k]+=C1

E_Q2=[]# liste des énergies quadratiques
C2=0
for k in range(0,16):
     E_Q2.append(((k+18)**2)/.7**2+C2)
for k in range(0,16):
    C2+=abs(((k+18)**2)/.7**2-epsilon[k+18])/N
print(C2)
for k in range(0,16):
     E_Q2[k]+=C2

E_Q3=[]# liste des énergies quadratiques
C3=0
for k in range(0,17):
     E_Q3.append(((k+33)**2)+C3)
for k in range(0,17):
    C3+=abs(((k+33)**2)-epsilon[k+33])/N
print(C3)
for k in range(0,17):
     E_Q3[k]+=C3

plt.figure(1)
X1=np.arange(0,18)
```

```
X2=np.arange(18,33)
X3=np.arange(33,50)
plt.plot(X,epsilon,label ='$\epsilon$(n)',marker='.',linestyle='')

plt.plot(X1,E_Q1, label ='E_Quadratique',marker='.',linestyle='')
plt.plot(X2,E_Q2, label ='E_Quadratique',marker='.',linestyle='')
plt.plot(X3,E_Q3, label ='E_Quadratique',marker='.',linestyle='')

#En utilisant la fonction curve_fit de python :
def f(x,a,b,c):
    return a*x**2+b*x+c
Y1=epsilon[0:18]
popt1,pcov1=curve_fit(f,X1,Y1)
print("Zone 1 : ",popt1)
Yopt1=[f(x,popt1[0],popt1[1],popt1[2]) for x in X1]

Y2=epsilon[18:33]
popt2,pcov2=curve_fit(f,X2,Y2)
print("Zone 2 : ",popt2)
Yopt2=[f(x,popt2[0],popt2[1],popt2[2]) for x in X2]

Y3=epsilon[33:50]
popt3,pcov3=curve_fit(f,X3,Y3,maxfev=2000)
print("Zone 3 : ",popt3)
Yopt3=[f(x,popt3[0],popt3[1],popt3[2]) for x in X3]

plt.plot(X1,Yopt1,label=f'E curve 1: E(n)={popt1[0]:.2f}$x^2${popt1[1]:+.2f}$x${popt1[2]:+.2f}')
plt.plot(X2,Yopt2,label=f'E curve 2: E(n)={popt2[0]:.2f}$x^2${popt2[1]:+.2f}$x${popt2[2]:+.2f}')
plt.plot(X3,Yopt3,label=f'E curve 3: E(n)={popt3[0]:.2f}$x^2${popt3[1]:+.2f}$x${popt3[2]:+.2f}')
plt.xlabel("n")
plt.ylabel("E/$E^{0}_{1}$")
plt.legend()
plt.grid()
plt.show()

##Question 3

#calculons le produit scalaire <x|psi>
X=np.linspace(0,a,1000)

#Calculons les fonctions Psi0,Psi1etPsi2 pour differents N
N=3
H=calc_H(N)
vp,vect =vecteurpropre(H)

Psi0_3=[prodScal(x,N,a,vect[0],1) for x in X]
Psi1_3=[prodScal(x,N,a,vect[1],1) for x in X]
Psi2_3=[prodScal(x,N,a,vect[2],-1) for x in X]

N=5
H=calc_H(N)
vp,vect =vecteurpropre(H)

Psi0_5=[prodScal(x,N,a,vect[0],1) for x in X]
Psi1_5=[prodScal(x,N,a,vect[1],1) for x in X]
Psi2_5=[prodScal(x,N,a,vect[2],-1) for x in X]

N=8
H=calc_H(N)
```

```
H=calc_H(N)
vp,vect =vecteurpropre(H)

Psi0_8=[prodScal(x,N,a,vect[0],1) for x in X]
Psi1_8=[prodScal(x,N,a,vect[1],1) for x in X]
Psi2_8=[prodScal(x,N,a,vect[2],1) for x in X]

N=15
H=calc_H(N)
vp,vect =vecteurpropre(H)

Psi0_15=[prodScal(x,N,a,vect[0],-1) for x in X]
Psi1_15=[prodScal(x,N,a,vect[1],1) for x in X]
Psi2_15=[prodScal(x,N,a,vect[2],1) for x in X]

#calculons les psi_theoriques:
def phiAdapt(x,n):
    if x>0.3*a and x<0.7*a:
        return phiHarm(x-0.3,n,0.4)
    return 0

PsiT0=[phiAdapt(x,0) for x in X]
PsiT1=[phiAdapt(x,1) for x in X]
PsiT2=[phiAdapt(x,2) for x in X]

#traçage des graphes
#psi0
plt.figure(2)
plt.plot(X,PsiT0,label ='$\Psi$0 puit potentiel')
plt.plot(X,Psi0_3, label ='N=3')
plt.plot(X,Psi0_5, label ='N=5')
plt.plot(X,Psi0_8, label ='N=8')
plt.plot(X,Psi0_15, label ='N=15')
plt.xlabel("x")
plt.ylabel("$\Psi$0(x)")
plt.legend()
plt.grid()
plt.show()

#psi1
plt.figure(3)
plt.plot(X,PsiT1,label ='$\Psi$1 puit potentiel')
plt.plot(X,Psi1_3, label ='N=3')
plt.plot(X,Psi1_5, label ='N=5')
plt.plot(X,Psi1_8, label ='N=8')
plt.plot(X,Psi1_15, label ='N=15')
plt.xlabel("x")
plt.ylabel("$\Psi$1(x)")
plt.legend()
plt.grid()
plt.show()

#psi2
plt.figure(4)
plt.plot(X,PsiT2,label ='$\Psi$2 puit potentiel')
plt.plot(X,Psi2_3, label ='N=3')
plt.plot(X,Psi2_5, label ='N=5')
plt.plot(X,Psi2_8, label ='N=8')
plt.plot(X,Psi2_15, label ='N=15')
```

```
plt.xlabel("x")
plt.ylabel("$\Psi$2(x)")
plt.legend()
plt.grid()
plt.show()

##courbe phi
plt.figure(5)
N=200
a=1
H=calc_H(N)
vp,vect =vecteurpropre(H)

i=0
axe=[0 for x in range(12)]
X=np.linspace(0,a,1000)
for k in range(12):
        #b=str(3)+str(3)+str(k+1)
        axe[k]=plt.subplot(3,4,k+1)
        Psi=[prodScal(x,N,a,vect[i],1) for x in X]
        axe[k].plot(X,Psi)
        plt.xlabel("x")
        plt.ylabel("$\Psi$_{}".format(str(i)))
        i+=7

plt.grid()
plt.show()



#### Partie B
D=5.73e9
g=2
e=1.6e-19
m=9.1e-31
fcav=9.19e9

def elementsPropres_RPE(theta,B0):
    '''calcul des éléments propres de l'hamiltonien normalisé (divisé par h_bar),
    pour une particule de spin 3/2 dans un champ B0 en Gauss
    vu depuis un angle theta par rapport à l'axe du champ.
    Le champ s'exprime en Gauss et theta en degré
    La base est |3/2,-3/2>;|3/2,-1/2>;|3/2,1/2>;|3/2,3/2>'''
    Sz=hbar/2*np.array([[-3,0,0,0],[0,-1,0,0],[0,0,1,0],[0,0,0,3]])
    Sx=hbar/2*np.array([[0,np.sqrt(3),0,0],[np.sqrt(3),0,2,0],[0,2,0,np.sqrt(3)],[0,0,np.sqrt(3),0]])
    S2=15*hbar**2/4*np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])

    B0=B0*1e-4
    H=-2*D*np.pi/hbar*(np.dot(Sz,Sz)-S2/3)+g*e*B0/2/m*(Sz*np.cos(theta/180*np.pi)
    +Sx*np.sin(theta/180*np.pi))

    vap,vect=np.linalg.eig(H)
    np.transpose(vect)
    index=vap.argsort()
    return vap[index]/h, vect[index]

##calcul des transitions possible
def sans_les_presques_doublons(L):
    """enlever les doublons de transitions pour de champs B très proches
```

```
    ex: pour L=[(404, [2,3]),(405, [2,3]),(406, [2,3]),(407, [2,3])],
    on retourne[(405, [2,3])] """
    n=len(L)
    assert n>0
    GrandeListe=[]
    petiteliste=[L[0]]
    i=1
    #On recupere dans GrandeListe les valeurs de B consecutives par morceau
    while i<n:
        while i<n and L[i][0]==petiteliste[-1][0]+1:
            petiteliste.append(L[i])
            i+=1
        GrandeListe.append(petiteliste)
        try:
            petiteliste=[L[i]]
            i+=1
        except:
            pass
    ResultatB=[]
    ResultatTransition=[]
    #On fait pour chaque segment la moyenne des valeurs de B

    for liste in GrandeListe:
        longueur=len(liste)
        s=0
        for i in range(longueur):
            s+=liste[i][0]
        ResultatB.append(int(s/longueur))
        ResultatTransition.append(liste[0][1])
    return ResultatB,ResultatTransition

def nb_transition(theta,N):
    """Pour chaque valeur de B entre 0 et (N-1)G,
    on renvoie la valeur de B qui permet une transition  et les niveaux qui rentrent en jeu.
    On retourne une liste d'élément desvaleur de B,
    et une correspondant à au passage correspondant [niveau initial,niveau final]"""
    trans=[]
    for i in range(N):
        B0=i
        vp,vect=elementsPropres_RPE(theta,B0)
        E1= vp[0]
        E2= vp[1]
        E3= vp[2]
        E4= vp[3]
        if np.abs((E2-E1)-fcav)<10e6:
            delta=[1,2]
            trans.append([i,delta])
        elif np.abs((E3-E2)-fcav)<10e6:
            delta=[2,3]
            trans.append([i,delta])
        elif np.abs((E4-E3)-fcav)<10e6:
            delta=[3,4]
            trans.append([i,delta])
        elif np.abs((E3-E1)-fcav)<5e6:
            delta=[1,3]
            trans.append([i,delta])
        elif np.abs((E4-E2)-fcav)<5e6:
            delta=[2,4]
            trans.append([i,delta])
```

```
        elif np.abs((E4-E1)-fcav)<5e6:
            delta=[1,4]
            trans.append((i,delta))
            #Il y a des riques de doublons pour des B très proches, il faut les enlever
    return sans_les_presques_doublons(trans)

## affichage des transitions sur le diagramme des énergies
def dessiner_transition(ax,B0, E, i, f):
    Ei,Ef=E[i-1],E[f-1]
    plt.arrow(B0, Ei[B0], 0 , Ef[B0]-Ei[B0]-1e9 , head_width=100, head_length=1e9, color="crimson")
    plt.text(B0+20, Ei[B0]+(Ef[B0]-Ei[B0])/2, "B0={}G".format(int(B0)), color="crimson")
def affichage_transitions_diagEnergie(transB,transEnergie,E):
    ax=plt.axes()
    for i in range(len(transB)):
        dessiner_transition(ax,transB[i],E,transEnergie[i][0],transEnergie[i][1])

## traçage de la courbes des transitions pour un intervalle de Bo et de Theta
def affichage_transitions_IntervalleTheta(pas,Bomax,thetamax):
""" calcule les transitions(pour Bo entre 0 et Bomax) possibles pour
chaque valeurs de theta dans l'intervalle [0,thetamax] avec un pas donnée
De plus il affiche les courbes obtenue dans un graphe"""
    trESansDoublons=[] #liste des differentes transitions possibles
                       # format [transition A, transition B, ...]
    Transition=[] #liste des valeurs de Bo possible par transitions
                  # format [[valeurs de Bo pour la transition A],
                  #         [valeurs de Bo pour une transition B], ...]
    Theta=[]     #liste des valeurs de Theta correspondant à un Bo possible
                 #format [[valeur de theta correspont à la transition A],
                 #       [valeur de theta correspont à la transition B], ...]

    #Initialisation à part pour éviter des recherches d'indices dans
    #des listes vides
    theta=0
    trB,trE=nb_transition(pas*theta,Bomax+1)
    trESansDoublons.append(trE[0])
    Transition.append([trB[0]])
    Theta.append([pas*theta])

    for i in range(1,len(trE)):
        if (trE[i] in trESansDoublons)==False:
            trESansDoublons.append(trE[i])
            Transition.append([])
            Theta.append([])
        a=trESansDoublons.index(trE[i]) # a est l'indice de Transition/Theta
                                        # correspondant à la transition i
        Transition[a].append(trB[i])
        Theta[a].append(pas*theta)

    #boucle pour les theta>0
    for theta in range(1,int(thetamax//pas+1)):
        trB,trE=nb_transition(pas*theta,Bomax+1)
        for i in range(len(trE)):
            if (trE[i] in trESansDoublons)==False:
                trESansDoublons.append(trE[i])
                Transition.append([])
                Theta.append([])
            a=trESansDoublons.index(trE[i])
            Transition[a].append(trB[i])
            Theta[a].append(pas*theta)
```

```
    #affichages des courbes
    for i in range(len(trESansDoublons)):
        plt.plot(Theta[i],Transition[i],'x',label ='transitions {}'
                            .format(trESansDoublons[i]),linestyle='')

##Fonctions pour vérifier les transitions pour un theta donné
theta=80
transB,transEnergie=nb_transition(theta,6001)
print('theta = {} | B= {} | transition entre les niveaux {}'
        .format(theta,transB,transEnergie))



## Calcul des énergies E(B) pour theta=50°
plt.figure(6)
theta=50
abs=[]
E1_50=[]
E2_50=[]
E3_50=[]
E4_50=[]
for i in range (6001):
    B0=i
    abs.append(i)
    vp,vect=elementsPropres_RPE(theta,B0)
    E1_50.append(vp[0])
    E2_50.append(vp[1])
    E3_50.append(vp[2])
    E4_50.append(vp[3])
plt.plot(abs,E1_50, label="E1")
plt.plot(abs,E2_50,label="E2")
plt.plot(abs,E3_50,label="E3")
plt.plot(abs,E4_50,label="E4")
plt.legend()

#calcul des transitions possibles
E=[E1_50,E2_50,E3_50,E4_50]
transB,transEnergie=nb_transition(theta,6001)
affichage_transitions_diagEnergie(transB,transEnergie,E)

plt.ylabel('$E/h$ (Hz)')
plt.xlabel('$B_0$ (Gauss)')
plt.grid()
plt.show()

##graphiques des valeurs de B de transitions pour theta=0 à 190° et B=0à8000G

plt.figure(7)
affichage_transitions_IntervalleTheta(1,8000,190)
plt.legend()
plt.grid()
plt.ylabel('$B_0$ (Gauss)')
plt.xlabel(r'$\theta$ (°)')
plt.show()

##comparaisons des valeurs expérimentales et numérique
Theta_exp=[0,0,0,0,20,20,20,20,40,40,40,40,60,60,60,80,80,80]
Transition_exp=[750,1050,3700,3300,1050,2620,4220,5750,
                460,1500,2600,4400,550,2040,3600,900,2140,4820,]
```

```
plt.figure(8)
plt.plot(Theta_exp,Transition_exp,'ko', label ='transitions mesurées expérimentalement',linestyle='')
affichage_transitions_IntervalleTheta(1,6000,90)
plt.legend()
plt.ylabel('$B_0$ (Gauss)')
plt.xlabel(r'$\theta (°)$')
plt.grid()
plt.show()
```