Compte Rendu : Physique numérique

Boris Baudel - Université du Maine- L3 Physique

TP3 : Physique numérique Mécanique Quantique

Introduction

Le but de ce travail pratique est d'évaluer une solution numérique à l'équation de shrodinger dépendent du temps. Pour cela, nous allons nous aider d'un programme python déjà fourni et nous allons globalement étudier les méthodes de discrétisation. Notamment comment l'on discrétise l'équation de Shrodinger.

Après nous allons étudier les méthodes numériques de numérisation par la méthode d'Euler. La méthode qui utiliser et critiquer le fait que l'approximation d'une fonction par sa pente. Nous allons en apercevoir ses erreurs.

Finalement, nous allons utiliser les méthodes de Runge-Kutta. Pour en déduire aussi les solutions de l'équations de shrodinger avec cette méthode. Par ailleurs, nous allons aussi voir l'influence d'un mur de potentiel sur la particule.

Question 1 :

L'équation de schrodinger dépendant du temps est :

$$i\hbar\frac{\partial\Psi(t,\vec{r})}{\partial t} = -\frac{\hbar^2}{2m}\Delta\Psi(t,\vec{r}) + V(\vec{r})\Psi(t,\vec{r})$$

$$i\hbar\frac{\mathrm{d}}{\mathrm{d}t}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle$$

Question 2 :

En faisant une discrétisation sur t, on a, d'après l'équation de shrodinger :

$$i\hbar\frac{\partial\psi(t,x)}{\partial t} = i\frac{\psi(i+1,j)-\psi(i,j)}{t(i+1)-ti} = H(x)\psi(t,x)$$

En faisant une discrétisation sur x, on a en remplaçant :

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi(x,t)}{\partial x^2}+V(x)\psi(x,t)$$

$$\frac{-1}{2}\frac{\psi(i,j-1)-\ 2\psi(i,j)+\psi(i,j+1)}{x(j+1)-x(j)}\ +V(j)\psi(i,j)$$

Finalement, en rejoignant les équations, on obtient :

$$\frac{-1}{2}\frac{\psi(i,j-1)-\ 2\psi(i,j)+\psi(i,j+1)}{x(j+1)-x(j)}\ +V(j)\psi(i,j)\ =i\frac{\psi(i+1,j)-\psi(i,j)}{t(i+1)-ti}$$

On a donc avec i l'unité imaginaire :

$$\psi(i+1,j)=\ \psi(i,j)\ +\ i(t(i+1)-ti)(\frac{1}{2}\frac{\psi(i,j-1)-\ 2\psi(i,j)+\psi(i,j+1)}{x(j+1)-x(j)}\ -\ V(j)\psi(i,j))$$

Les indices (x-1) et (x+1) sont : $x\ -\ \Delta x\ et\ x\ +\Delta x$. Pour l'équation, 16 on a :

$$\psi(0,x)\ =\ \int_{-\infty}^{\infty}\psi(k)\,e^{-i(kx-wt)}dk$$

$$\psi(0,x)\ =\ \sqrt{\frac{2\sigma}{\sqrt{2\pi\sigma}}}\ e^{-\frac{x^2}{4\sigma*\sigma}+ik\sigma}$$

$$\psi(0,j)\ =\ \sqrt{\frac{2\sigma}{\sqrt{2\pi\sigma}}}\ e^{-\frac{j^2}{4\sigma*\sigma}+ik\sigma}$$

Question 3 :

Dans le programme python, la forme de ψ est :

$$\psi(0,x)\ =\ \sqrt{\frac{2\sigma}{\sqrt{2\pi\sigma}}}\ e^{-\frac{x^2}{4\sigma*\sigma}+ik\sigma}$$

Les significations des variables it, nt, t, ix, x ,nx est :

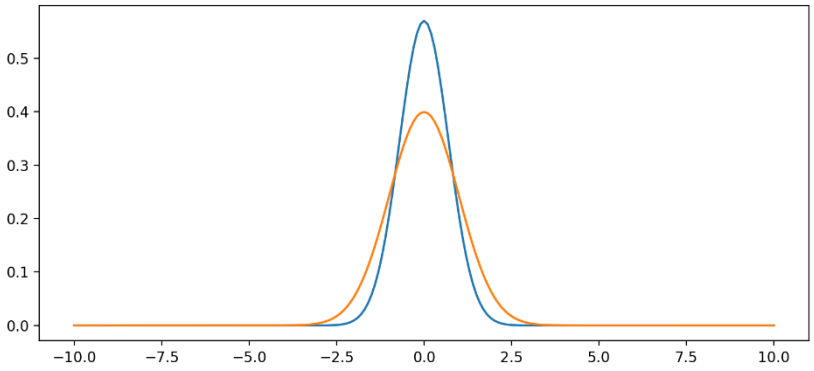
Pour x, il s'agit du vecteur avec tout les coordonnée x à parcourir, elles vont de [-10 ;10] d'un pas de dx. nx est la longueur du vecteur x, c'est le nombre de valeurs du vecteur x Pour ix, il s'agit de l'indice i qui varie utilisée dans les équations précédentes. C'est l'indice qui augmente dans x grâce à la boucle while.

Pour t, il s'agit du vecteur contenant l'ensemble des durées pour lequel on veut modéliser notre équation, les durées sont comprises entre [0.1] et le pas est de dt.

Pour nt, il s'agit du nombre de valeurs dans le vecteur t, c'est la longueur, c'est la différence de valeurs finale et initiale divisé par le pas.

Pour it , il s'agit de l'indice j, dans l'équation précédente. C'est l'indice de la durée temporelle et prend ses valeurs dans t à l'aide de la boucle while. it et ix sont donc l'indice à un lieu x et un temps t donnée du vecteur.

Voici ce qui est tracé dans le programme :



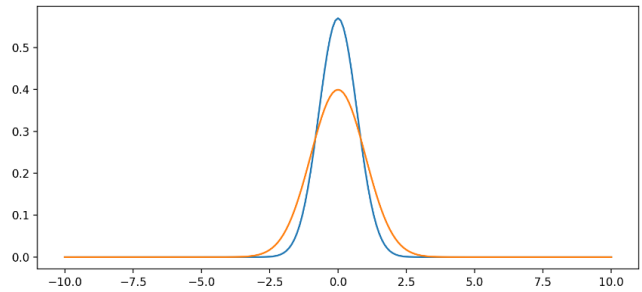
Le courbe bleu représente la fonction précédente, solution donc théorique :

$$\psi(0,x) = \sqrt{\frac{2\sigma}{\sqrt{2\pi\sigma}}}\, e^{-\frac{x^2}{4\sigma*\sigma}+ik\sigma}$$

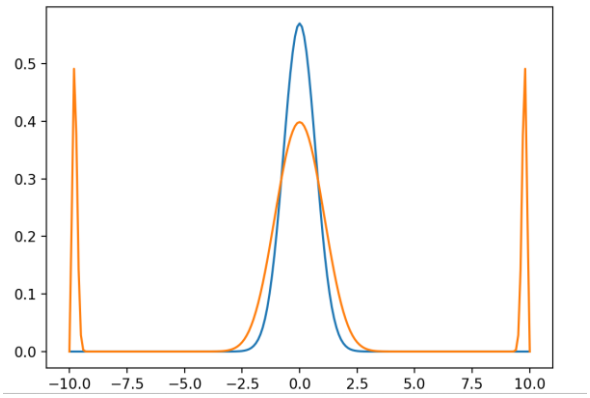
Le courbe orange représente la solution à l'aide de la méthode d'Euler en discrétisant l'équation de shrodinger.

Question 4 :

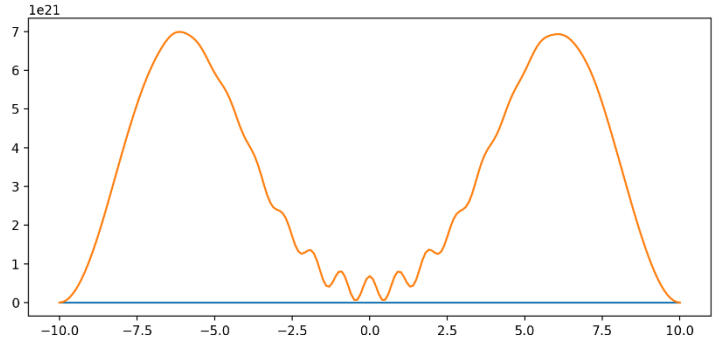
En considérant ψ sur une unité atomique avec des paramètres du pas temporelle de 0.001 unités atomiques. L'espace est défini dans l'intervalle [-10,10] et on le discrétise en un pas 0.1, avec σ = 0.7. et ko = 0. On observe :

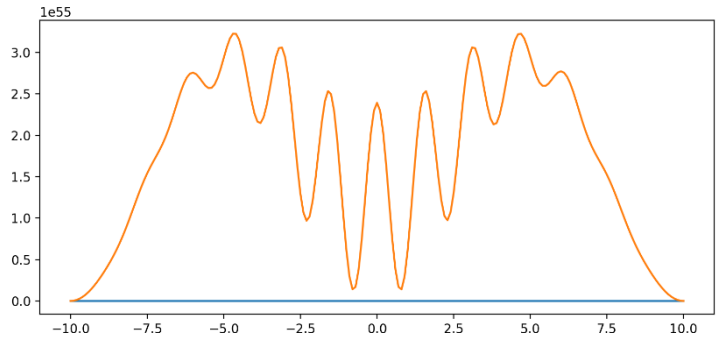


En prenant un pas temporel de 0.1, on observe des problèmes dans les bords. Des incohérences à ce niveau.



Par ailleurs, si l'on change l'intervalle temporelle à [0,3] on obtient ceci, avec le pas qui été à l'initiale, puis [0,5] :

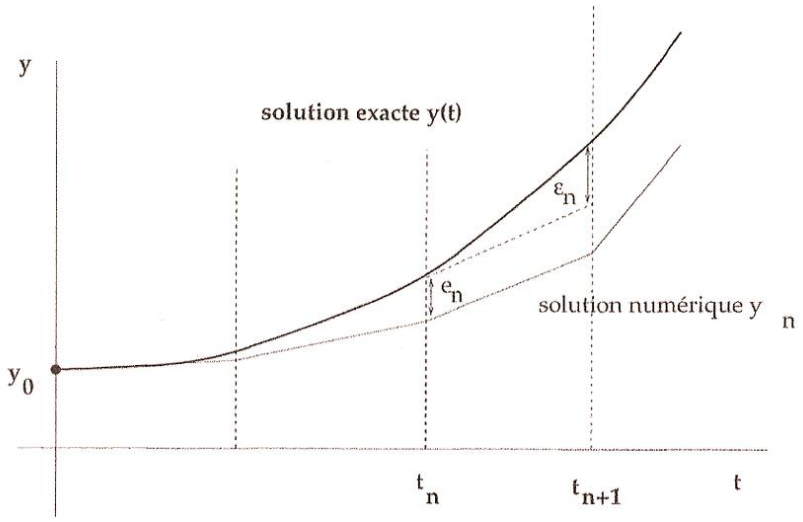




On remarque, qu'il y a une incohérence. En effet, la propriété fondamentale n'est pas respectée :

$$1 = \int_{-\infty}^{\infty} |\psi|^2 \, dx$$

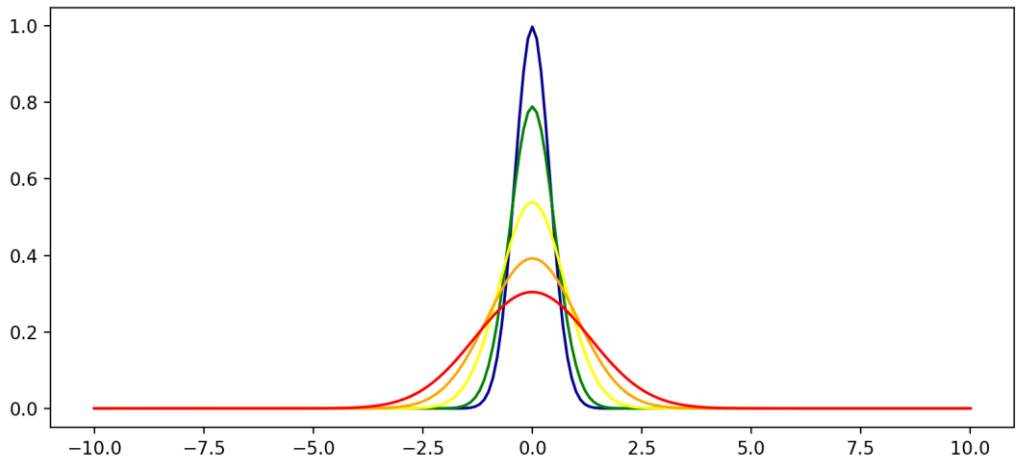
Cette erreur, est principalement due à l'approximation qui avait été effectuer précédemment dans le programme par la méthode d'Euler.



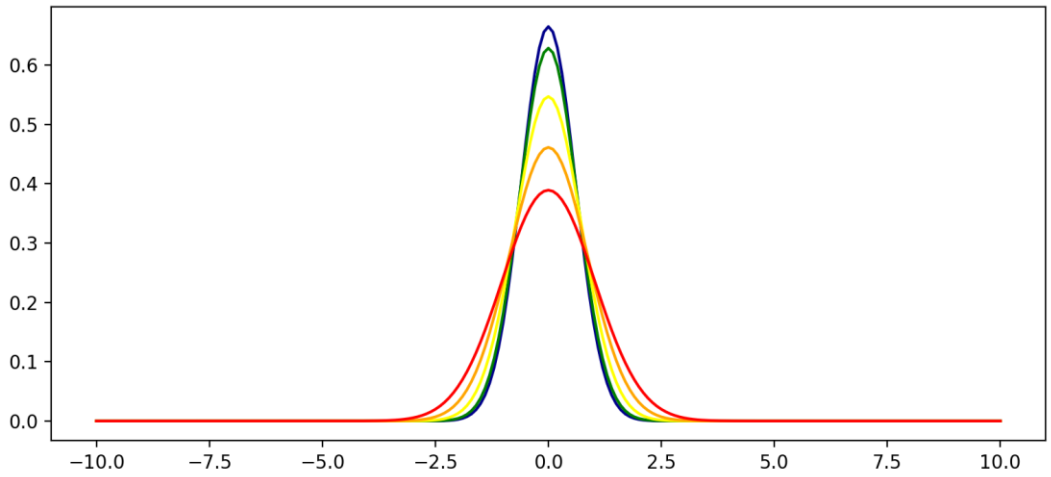
Pour corriger l'erreur, il faut diminuer le pas. Toutefois, il faut que le temps de calcule soit raisonnable. Voici le programme python utilisée précédemment fourni :

```
def euler():

    #import pylab
    #pylab.plot(range(10), 'o')
    import numpy as np
    import matplotlib.pyplot as pplot

    dx=0.1
    x=np.arange(-10.,10.+dx,dx)
    nx=len(x)
    #print(nx)
    lapsi=np.zeros(nx)+1j
    dpsi=np.zeros(nx)+1j
    dt=0.001
    t=np.arange(0.,3.+dt,dt)
    nt=len(t)
    #print(nt)
    psi=np.zeros((nt,nx))+1j

    sig=0.7
    k0=0.

    rac2pi=np.sqrt(2*np.pi)
    normfac=np.sqrt(1/(rac2pi*sig))
    psi[0,:]=normfac*np.exp(-x*x/(4*sig*sig)+1j*k0*x)
    psi[0,:]=psi[0,:]*1e9   #else precision of float/complex insufficient
    #pylab.plot(psi[0,:], 'o')

    V=0.

    it=0
    while it < nt-1:
        ix=1
        while ix < nx-1:
            lapsi[ix]=(psi[it,ix-1]-2*psi[it,ix]+psi[it,ix+1])/dx**2
            ix=ix+1
        # end while ix
        dpsi=dt*(-0.5*lapsi+V*psi[it,:])/1j
        psi[it+1,:]=psi[it,:]+dpsi
        it=it+1
        #pylab.plot(psi[it,:], 'o')
    # end while it
    #print(psi)
```

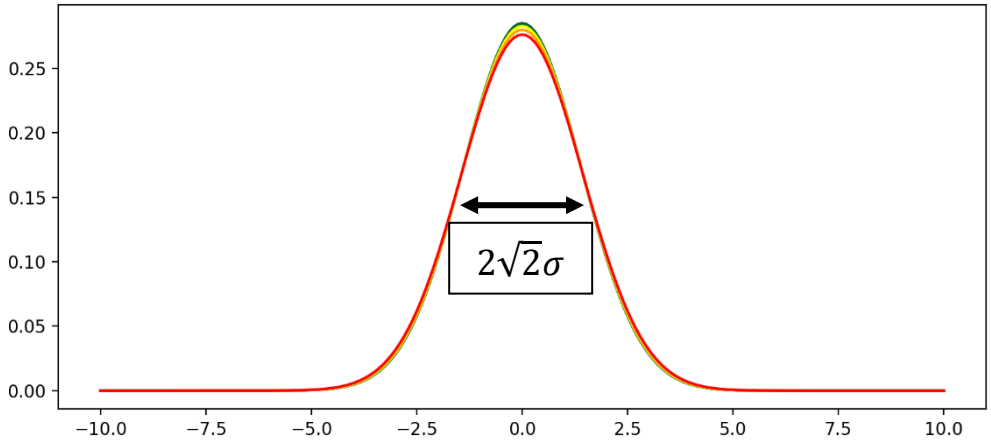
En utilisant le modèle de Runge-kutta et prenant $\sigma = 0.4$:



En utilisant le modèle de Runge-kutta et prenant $\sigma = 0.5$:

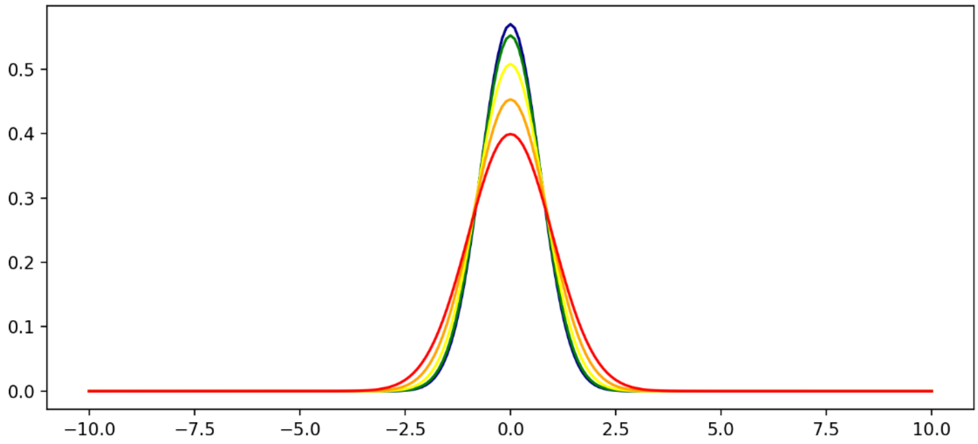


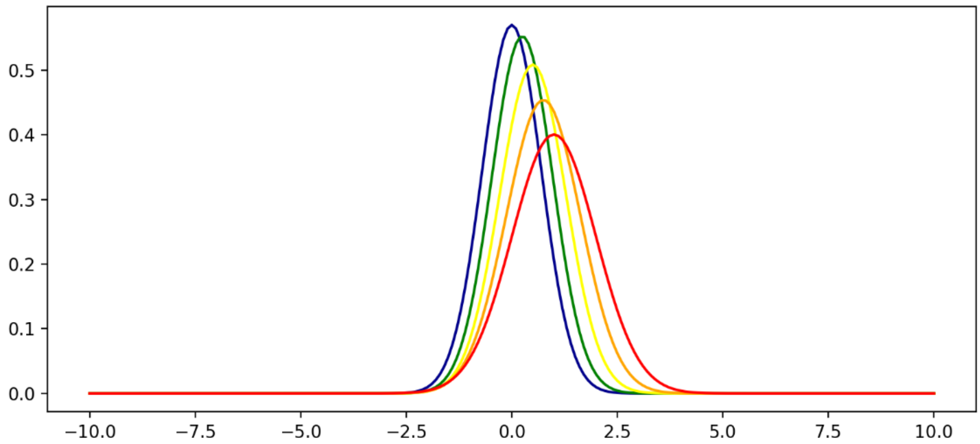
En utilisant le modèle de Runge-kutta et prenant $\sigma = 1.5$:

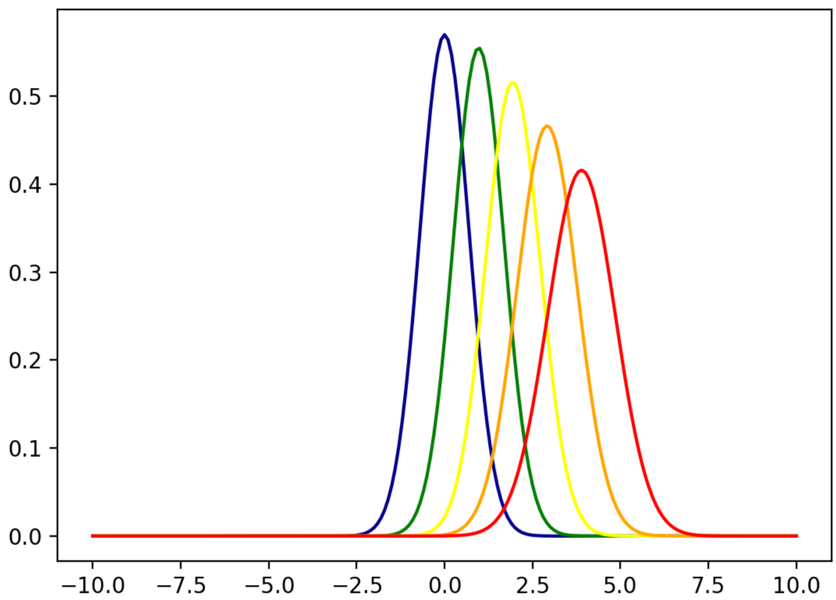


On remarque alors que σ représente l'écart type associée à la loi de probabilité de trouver la particule. Si σ augmente alors l'incertitude de trouver la particule augmente.

En utilisant le modèle de Runge-kutta et prenant $\sigma = 0.7$, on a successivement pour les valeurs k = 0 ; 1 ; 4 :

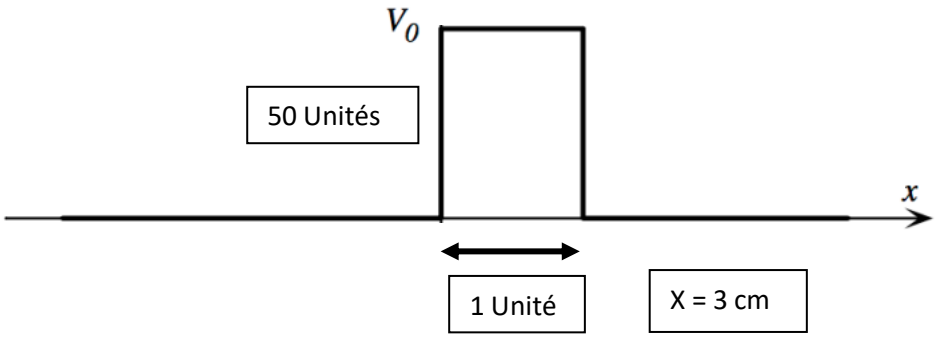


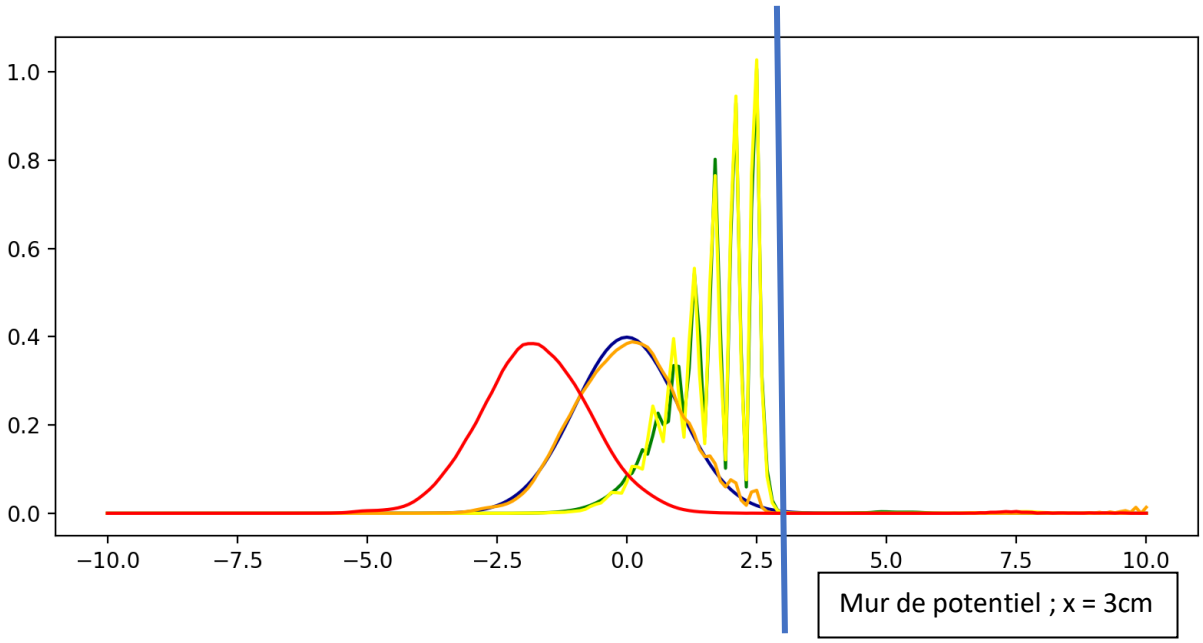




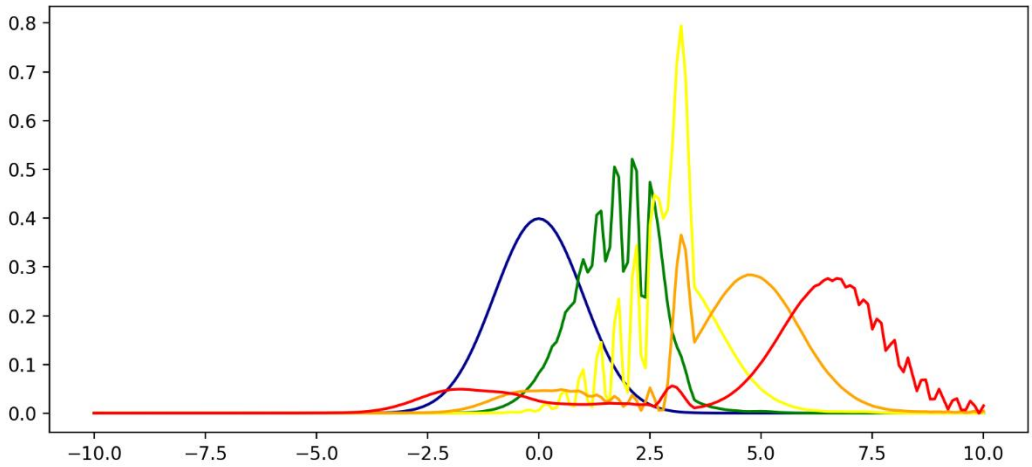
La valeur k représente le nombre d'onde, donc la relation de dispersion, la manière dont se disperse l'onde. On observe donc que, en variant k, on fait varier

Maintenant, nous allons mettre σ = 1 et k= 8. On considère alors une barre de potentielles de largeur 0.1 située à 3 du centre. Si l'on rajoute un potentiel, la fonction d'onde va changer.

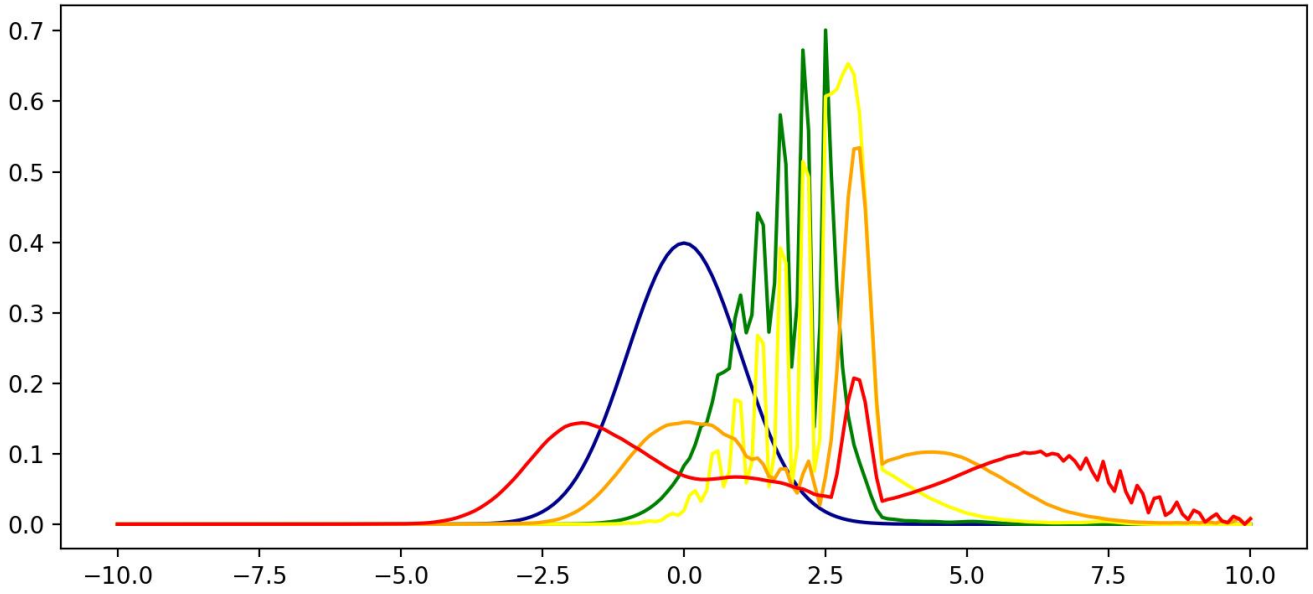




On observe que pour plusieurs temps, la fonction d'onde, et donc la particule est réfléchie par le mur de potentiel à x = 3cm, pour une Energie potentiel associée à 50.

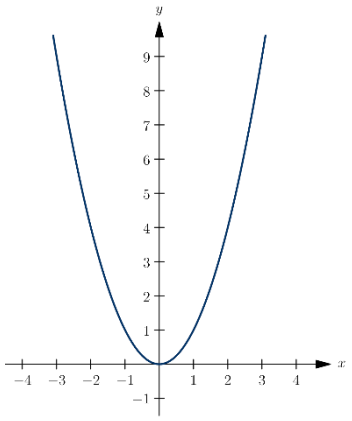


On observe que pour plusieurs temps, la fonction d'onde, et donc la particule traverse mur de potentiel à x = 3cm, pour une Energie potentiel associée à V= 20.

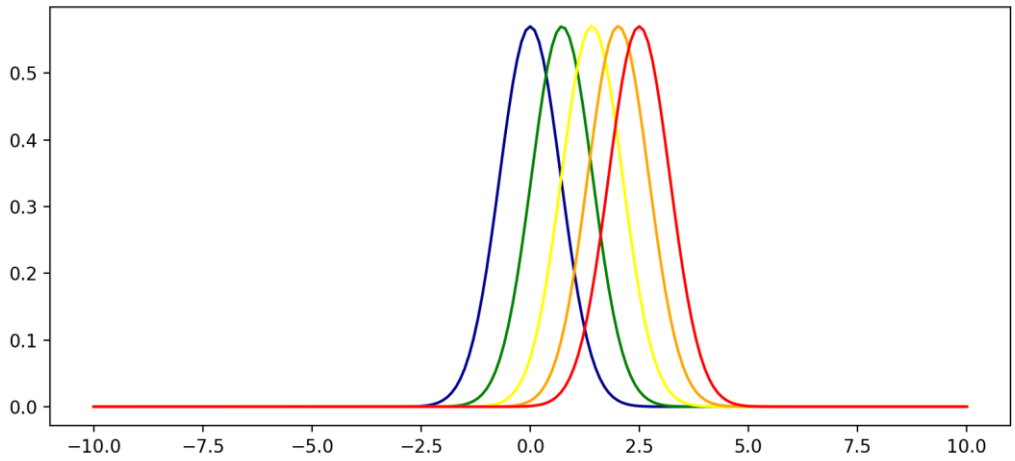


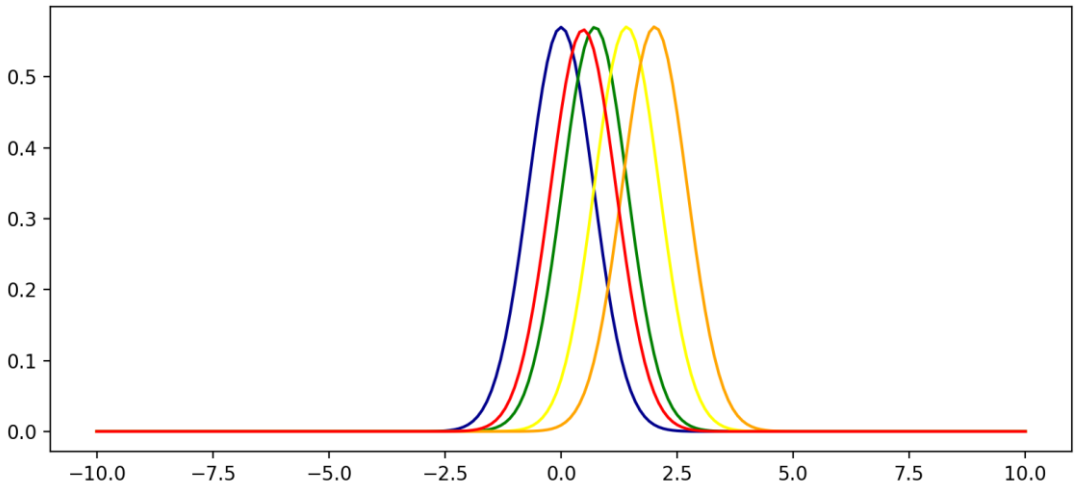
On observe que pour plusieurs temps, la fonction d'onde associe une probabilité de passer le mur du potentiel et une probabilité d'être réfléchie (fonction d'onde finale rouge) à x = 3 : , pour une Energie potentiel associée à V= 28.

Nous nous plaçons à un potentiel maintenant sous la forme V(x) = 0.5x*x ; et sig=0,7 ; ko = 3

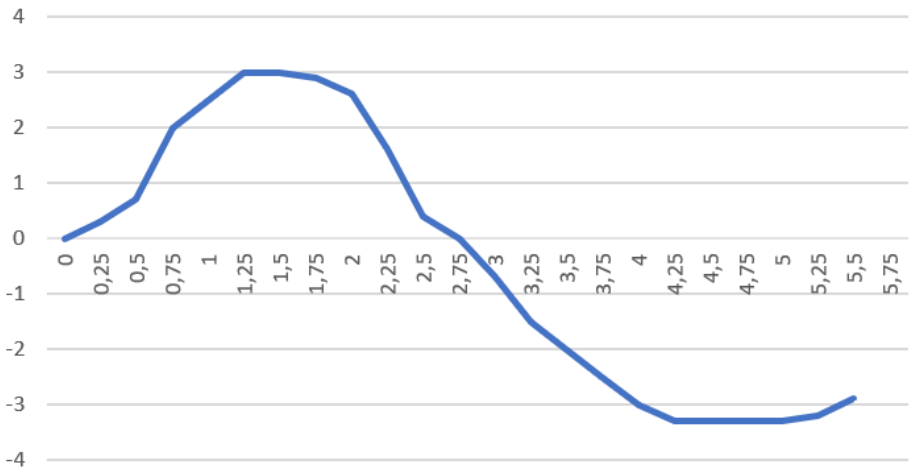


Ainsi, nous obtenons alors les fonctions d'onde suivantes. On remarque alors que le potentiel frêne la particule sans modifier sa répartition de probabilité. (I= [0,1] ; I=[0,3])

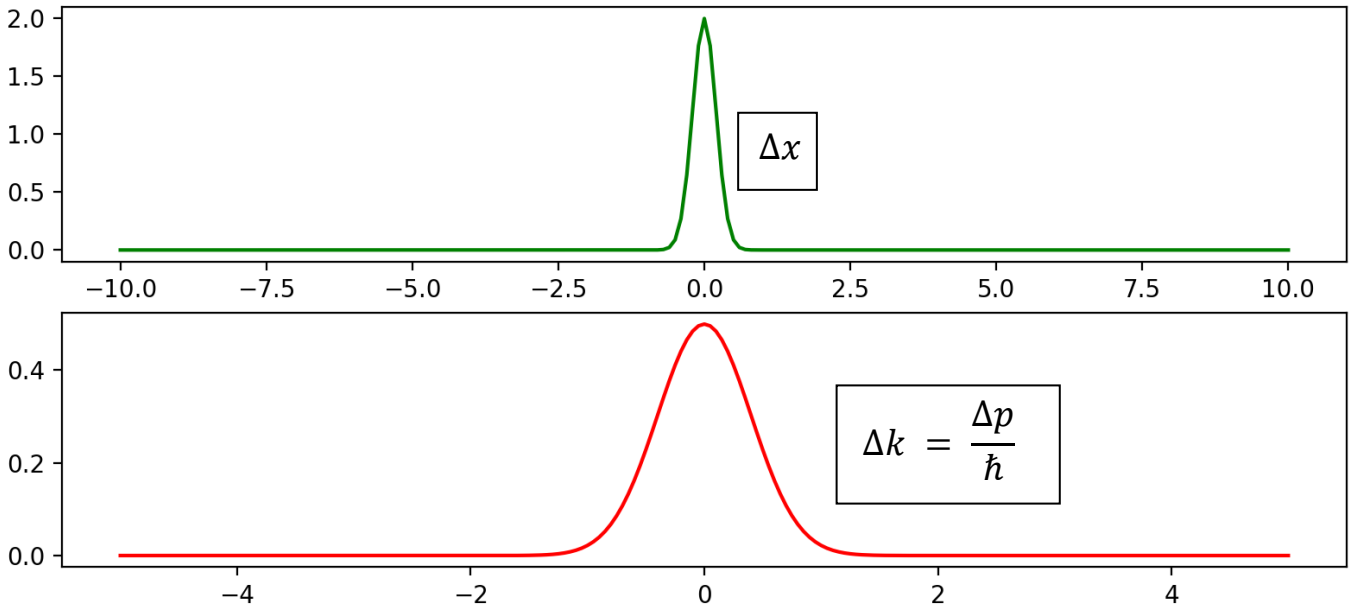




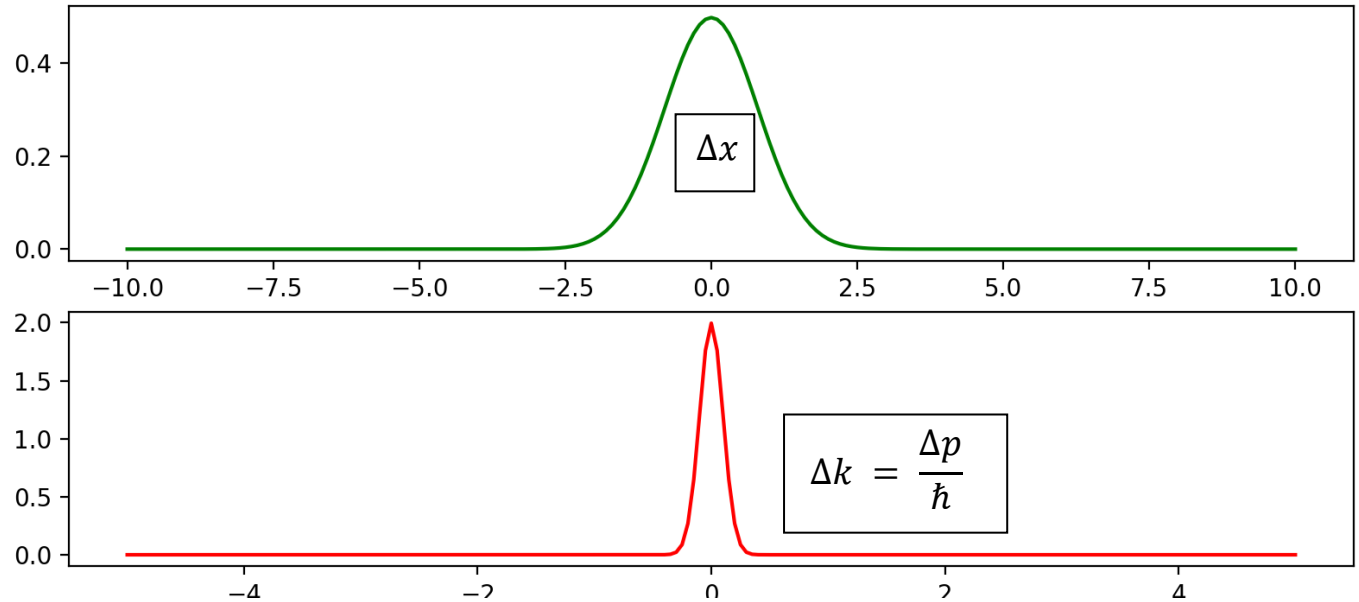
La trajectoire de la particule est donc de période 5 et de fréquence 0,20 Hz.



En utilisant le dernier programme, on obtient la transformée de Fourier de la fonction d'onde en fonction de t et de k.



Si l'on change la distribution, avec sigma $\sigma = 0{,}8$; l'écart type de $\psi(k)$ diminue et $\psi(t)$ augmente.



Ce que l'on observe içi c'est le principe d'Incertitude d'Heisenberg. En effet, si l'on augmente l'incertitude sur x alors on a la diminue sur p et inversement.

```
def rungekutta4_ft():

    #import pylab
    import numpy as np
    import matplotlib.pyplot as pplot

    dx=0.1
    xmin=-10.
    xmax=10.
    x=np.arange(xmin,xmax+dx,dx)
    nx=len(x)
    lapsi1=np.zeros(nx)+1j
    lapsi2=np.zeros(nx)+1j
    lapsi3=np.zeros(nx)+1j
    lapsi4=np.zeros(nx)+1j
    k1=np.zeros(nx)+1j
    k2=np.zeros(nx)+1j
    k3=np.zeros(nx)+1j
    k4=np.zeros(nx)+1j
    psi1=np.zeros(nx)+1j
    psi2=np.zeros(nx)+1j
    psi3=np.zeros(nx)+1j

    V=np.zeros(nx)

    dt=0.001
    t=np.arange(0.,1.+dt,dt)
    nt=len(t)
    psi=np.zeros((nt,nx))+1j

    sig=0.2
    k0=0.
    rac2pi=np.sqrt(2*np.pi)
    normfac=np.sqrt(1/(rac2pi*sig))
    psi[0,:]=normfac*np.exp(-x*x/(4*sig*sig)+1j*k0*x)
    psi[0,:]=psi[0,:]*1e9   #else precision of float/complex insufficient
    #pylab.plot(psi[0,:], 'o')

    #ix=0
    #while ix < nx:
    #   if (x[ix] >  and x[ix] <  ):
    #       V[ix]=
    #    else:
    #        V[ix]=
    #    #end if
    #    ix=ix+1
    # end while ix

    #case f' has no explicite dependence on t
    it=0
    while it < nt-1:
        ix=1
        while ix < nx-1:
            lapsi1[ix]=(psi[it,ix-1]-2*psi[it,ix]+psi[it,ix+1])/dx**2
            ix=ix+1
        # end while ix
        lapsi1[0]=2*lapsi1[1]-lapsi1[2]
        lapsi1[nx-1]=2*lapsi1[nx-2]-lapsi1[nx-3]
        k1=dt*(-0.5*lapsi1+V*psi[it,:])/1j
        psi1=psi[it,:]+0.5*k1
```

```
    ix=1
    while ix < nx-1:
        lapsi2[ix]=(psi1[ix-1]-2*psi1[ix]+psi1[ix+1])/dx**2
        ix=ix+1
    # end while ix
    lapsi2[0]=2*lapsi2[1]-lapsi2[2]
    lapsi2[nx-1]=2*lapsi2[nx-2]-lapsi2[nx-3]
    k2=dt*(-0.5*lapsi2+V*psi1)/1j
    psi2=psi[it,:]+0.5*k2

    ix=1
    while ix < nx-1:
        lapsi3[ix]=(psi2[ix-1]-2*psi2[ix]+psi2[ix+1])/dx**2
        ix=ix+1
    # end while ix
    lapsi3[0]=2*lapsi3[1]-lapsi3[2]
    lapsi3[nx-1]=2*lapsi3[nx-2]-lapsi3[nx-3]
    k3=dt*(-0.5*lapsi3+V*psi2)/1j
    psi3=psi[it,:]+k3

    ix=1
    while ix < nx-1:
        lapsi4[ix]=(psi3[ix-1]-2*psi3[ix]+psi3[ix+1])/dx**2
        ix=ix+1
    # end while ix
    lapsi4[0]=2*lapsi4[1]-lapsi4[2]
    lapsi4[nx-1]=2*lapsi4[nx-2]-lapsi4[nx-3]
    k4=dt*(-0.5*lapsi4+V*psi3)/1j

    psi[it+1,:]=psi[it,:]+k1/6+k2/3+k3/3+k4/6
```

```
psi=psi/1e9
psiq=psi*psi.conjugate()

pplot.subplot(2, 1, 1)
pplot.plot(x,psiq[0,:], color='green')
#pplot.plot(x,psiq[nt-1,:], color='red')

kmin=-1/(2*dx)
kmax=1/(2*dx)
dk=1/(xmax-xmin)
k=np.arange(kmin,kmax+dk,dk)
#k=np.linspace(kmin,kmax+dk,nk)
nk=len(k)

psikinitmp=np.zeros(nk)+1j
psikfintmp=np.zeros(nk)+1j
psikini=np.zeros(nk)+1j
psikfin=np.zeros(nk)+1j

psikinitmp=np.fft.fft(psi[0,:])/np.sqrt(nx)
psikfintmp=np.fft.fft(psi[nt-1,:])/np.sqrt(nx)
ik=0
while ik < nk:
    shiftk=int(ik-np.floor(nk/2))
    #print shiftk
    psikini[ik]=psikinitmp[shiftk]
    psikfin[ik]=psikfintmp[shiftk]
    ik = ik+1
#end while
psikiniq=psikini*psikini.conjugate()
psikfinq=psikfin*psikfin.conjugate()
```

```
    psikinitmp=np.fft.fft(psi[0,:])/np.sqrt(nx)
    psikfintmp=np.fft.fft(psi[nt-1,:])/np.sqrt(nx)
    ik=0
    while ik < nk:
        shiftk=int(ik-np.floor(nk/2))
        #print shiftk
        psikini[ik]=psikinitmp[shiftk]
        psikfin[ik]=psikfintmp[shiftk]
        ik = ik+1
    #end while
    psikiniq=psikini*psikini.conjugate()
    psikfinq=psikfin*psikfin.conjugate()

    pplot.subplot(2, 1, 2)
    pplot.plot(k,psikiniq, color='red')
    #pplot.plot(k,psikfinq, color='red')
    pplot.show()

# end def rungekutta4_ft

# main program starts here
rungekutta4_ft()
```

Conclusion

Nous avons mis en évidences certains phénomènes du monde quantique, dans ce cadre. Nous avons étudié une particule et les effets de son environnement. Cela m'a permis de m'introduire au monde quantique et ses caractéristiques.