Équation de Schrödinger à une dimension

1. Introduction

Ce document montre comment faire la résolution numérique de l'équation de Schrödinger dépendante du temps, en suivant une méthode similaire à celle utilisée pour l'équation de diffusion. On verra comment appliquer la méthode à un paquet d'onde diffusé par un potentiel.

Voir aussi la simulation Équation de Schrödinger à une dimension.

2. Équation de Schrödinger

On considère l'équation de Schrödinger dépendante du temps pour une fonction d'onde $\psi(x,t)$:

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi \tag{1}$$

On s'intéresse à la diffusion de particules dans un potentiel $V(x)$. La résolution est limitée à l'intervalle fini $[0,L]$. Sur les bords de cet intervalle, la fonction d'onde est supposée nulle. La condition de normalisation s'écrit :

$$\int_0^L \psi\psi^*\,dx = 1 \tag{2}$$

La fonction d'onde initiale $\psi(x,0)$ est connue. Elle prendra la forme d'un paquet d'onde localisée à l'intérieur de l'intervalle $[0,L]$.

On cherche à résoudre numériquement l'équation par la méthode des différences finies. La méthode utilisée est similaire à celle développée dans Équation de diffusion à une dimension.

Soit V_0 l'échelle du potentiel. La fonction $V(x)$ utilisée sera sans dimensions. On introduit l'échelle de temps :

$$\tau = \frac{\hbar}{V_0} \tag{3}$$

et l'échelle de longueur :

$$\lambda = \frac{\sqrt{2mV_0}}{\hbar} \tag{4}$$

En utilisant un temps et un x réduits par ces deux échelles, on obtient l'équation :

$$i\frac{\partial\psi}{\partial t} = -\frac{\partial^2\psi}{\partial x^2} + V(x)\psi \tag{5}$$

On utilise l'opérateur hamiltonien :

$$H = -\frac{\partial^2}{\partial x^2} + V(x) \tag{6}$$

La solution de l'équation de Schrödinger peut s'écrire formellement à l'aide de l'exponentielle de l'opérateur hamiltonien ([1]) :

$$\psi(x,t) = e^{-iHt}\psi(x,0) \tag{7}$$

La conservation de la condition (2) au cours du temps vient du caractère unitaire de l'opérateur e^{-iHt}.

3. Méthode numérique

3.a. Discrétisation

Soit N le nombre de points dans l'intervalle $[0, L]$. On définit le pas d'espace :

$$\Delta x = \frac{L}{N-1} \tag{8}$$

Soit Δt le pas de temps. On pose :

$$\psi_j^n = \psi(j\Delta x, n\Delta t) \tag{9}$$

$$V_j = V(j\Delta x) \tag{10}$$

où n est un entier positif ou nul repésentant le temps.

La dérivée temporelle est discrétisée de la manière suivante :

$$\frac{\partial \psi}{\partial t} \to \frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} \tag{11}$$

La dérivée seconde par rapport à x est discrétisée par :

$$\frac{\partial^2 \psi}{\partial x^2} \to \frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^{n+1}}{(\Delta x)^2} \tag{12}$$

Le schéma explicite qui en découle est :

$$i\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = -\frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{(\Delta x)^2} + V_j\psi_j^n \tag{13}$$

Comme pour l'équation de diffusion, ce schéma explicite a une condition de stabilité très contraignante. On lui préfère donc un schéma implicite.

Pour alléger les notations, on introduit l'opérateur hamiltonien discret défini par :

$$H\psi_j^n = -\frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{(\Delta x)^2} + V_j\psi_j^n \tag{14}$$

3.b. Schéma implicite unitaire

En suivant la démarche adoptée pour l'équation de diffusion à une dimension, on est amené à considérer le schéma implicite suivant :

$$i\frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = H\psi_j^{n+1} \tag{15}$$

ou encore :

$$(1 + i\Delta t H)\psi_j^{n+1} = \psi_j^n \tag{16}$$

Ce schéma conduit à la résolution, pour chaque pas de temps, d'un système linéaire tridiagonal, pour lequel il existe une méthode directe efficace. Il a toutefois l'inconvénient de ne pas conserver la norme de la fonction d'onde.

Pour un intervalle de temps, on a :

$$\psi(x,t+\Delta t) = e^{-iH\Delta t}\psi(x,t) \tag{17}$$

Le schéma défini ci-dessus revient à utiliser l'approximation au premier ordre suivante :

$$e^{-iH\Delta t} \simeq 1 - iH\Delta t \tag{18}$$

Cette transformation n'est pas unitaire, c'est pourquoi la norme de la fonction d'onde n'est pas conservée. On la remplace par l'approximation de Cayley ([2]), qui est unitaire :

$$e^{-iH\Delta t} \simeq \frac{1 - i\frac{1}{2}H\Delta t}{1 + i\frac{1}{2}H\Delta t} \tag{19}$$

Le schéma numérique s'écrit alors :

$$(1 + i\frac{1}{2}\Delta t H)\psi_j^{n+1} = (1 - i\frac{1}{2}\Delta t H)\psi_j^n \tag{20}$$

On obtient ainsi le schéma implicite suivant :

$$\psi_{j-1}^{n+1} + (i\alpha - (\Delta x)^2 V_j - 2)\psi_j^{n+1} + \psi_{j+1}^{n+1} = -\psi_{j-1}^n + (i\alpha + (\Delta x)^2 V_j + 2)\psi_j^n - \psi_{j+1}^n \tag{21}$$

avec :

$$\alpha = \frac{2(\Delta x)^2}{\Delta t} \tag{22}$$

On le met sous la forme générale suivante :

$$a_j\psi_{j-1}^{n+1} + b_j\psi_j^{n+1} + c_j\psi_{j+1}^{n+1} = d_j\psi_{j-1}^n + e_j\psi_j^n + f_j\psi_{j+1}^n + s_j^n \tag{23}$$

Pour chaque pas de temps, le système tridiagonal est résolu par l'algorithme d'élimination décrit dans Équation de diffusion à une dimension.

Les conditions limites sont de type Dirichlet, puisqu'elles consistent à annuler la fonction d'onde sur les bords de l'intervalle :

$$\psi_0^n = \psi_{N-1}^n = 0 \tag{24}$$

3.c. Paquet d'onde

Pour obtenir l'évolution d'un paquet d'onde, on considère comme condition initiale un paquet d'onde gaussien défini par :

$$\psi(x,0) = e^{ik_0 x} e^{-\frac{(x-x_0)^2}{4\sigma_0^2}} \tag{25}$$

Le paquet d'onde est initialement en $x = x_0$ et se déplace dans le sens de l'axe x. La densité de propabilité (non normée) associée à ce paquet est :

$$\rho(x,0) = \psi(x,0)\psi(x,0)^* = e^{-\frac{(x-x_0)^2}{2\sigma_0^2}} \tag{26}$$

La variance de cette densité de probabilité est σ_0. Le spectre en nombre d'onde de ce paquet a une largeur donnée par :

$$\sigma_0 \Delta k = \frac{1}{2} \tag{27}$$

Considérons l'évolution du paquet d'onde en l'absence de potentiel (particule libre). La largeur Δk reste constante au cours du temps. Notons σ la largeur du paquet à l'instant t. L'inégalité d'Heisenberg s'écrit :

$$\sigma \Delta k \geq \frac{1}{2} \tag{28}$$

Pour ce paquet gaussien, l'égalité est réalisée à l'instant $t = 0$. On peut calculer précisément la largeur du paquet à l'instant t ([3]) :

$$\sigma = \sigma_0 \sqrt{1 + \frac{\hbar^2 t^2}{4m^2 \sigma_0^4}} \tag{29}$$

Cette relation montre l'étalement du paquet d'onde lorsqu'on s'éloigne de l'origine du temps.

La vitesse du paquet d'onde (la vitesse de groupe) est :

$$v_0 = \frac{\hbar k_0}{m} \tag{30}$$

Soit T la durée de la simulation. La distance parcourue par le paquet pendant cette durée doit être de l'ordre de la largeur L. Avec les unités utilisées ($m = 1/2$ et $\hbar = 1$), la condition s'écrit :

$$T \simeq \frac{L}{2k_0} \tag{31}$$

Si l'on souhaite que l'étalement du paquet d'onde soit négligeable pendant cette durée, il faut respecter la condition suivante :

$$T \ll \sigma_0 \tag{32}$$

qui s'écrit aussi :

$$2\sigma_0^2 k_0 \gg L \tag{33}$$

Par exemple, pour $L = 1$ et $\sigma_0 = 0.05$, il faut que $k_0 > 200$.

Voyons comment choisir l'échantillonnage spatial et temporel. Le pas d'espace doit vérifier :

$$k_0 \Delta x \ll 1 \tag{34}$$

La condition de Nyquist-Shannon impose une valeur maximale égale à π, mais il faut prévoir une marge à cause de la dispersion Δk. L'évolution dans le temps de la fonction d'onde se fait à la pulsation moyenne :

$$\omega = \frac{E}{\hbar} = \frac{\hbar k_0^2}{2m} \tag{35}$$

Avec les unités utilisées, le pas de temps doit donc vérifier :

$$k_0^2 \Delta t \ll 1 \tag{36}$$

Pour fixer les pas de temps et d'espace, on peut adopter la relation suivante :

$$\Delta t = 2\Delta x^2 \tag{37}$$

Il suffit alors de vérifier la relation :

$$k_0 \Delta x \ll 1 \tag{38}$$

Par exemple, pour $k_0 = 200$ et $L = 1$, il faut $N = 2000$ points. Le nombre de pas de temps nécessaire est :

$$P = \frac{T}{\Delta t} = \frac{L}{4k_0 \Delta x^2} \tag{39}$$

Avec les valeurs précédentes $P = 5000$.

On voit donc que la simulation d'un paquet d'onde présentant un faible étalement est très exigeante en terme de quantité de calculs.

4. Implémentation python

```
import numpy
import math
import cmath


class Schrodinger1D:
    def __init__(self,L,N):
        self.L = L
        self.N = N
        self.dx = L/(N-1)
        self.b = numpy.zeros(N,dtype=numpy.complex)
        self.e = numpy.zeros(N,dtype=numpy.complex)
        self.x = numpy.zeros(N,dtype=numpy.complex)
        self.beta = numpy.zeros(N,dtype=numpy.complex)
        self.gamma = numpy.zeros(N,dtype=numpy.complex)

    def config(self,dt,V):
        N = self.N
        self.dt = dt
        dx2 = self.dx*self.dx
        alpha = 2*dx2/dt
        for k in range(1,self.N-1):
            self.b[k] = 1j*alpha-dx2*V[k]-2.0
            self.e[k] = 1j*alpha+dx2*V[k]+2.0
        self.b[0] = 1.0
        self.b[N-1] = 1.0
        self.beta[0] = self.b[0]
```

```
        self.gamma[0] = 1.0/self.beta[0]
        for k in range(1,N):
            self.beta[k] = self.b[k]-self.gamma[k-1]
            if self.beta[k]==0:
                raise Exception("Impossible de resoudre le systeme ligne "+str(k))
            self.gamma[k] = 1.0/self.beta[k]

    def iterations(self,psi0,ti,tf):
        if psi0.size!=self.N:
            raise Exception("Taille de U incorrecte")
        psi = psi0.copy()
        t = ti
        while t<tf:
            self.x[0] = (self.e[0]*psi[0]-psi[1])/self.beta[0]
            for k in range(1,self.N-1):
                self.x[k] = (-psi[k-1]+self.e[k]*psi[k]-psi[k+1]-self.x[k-1])/self.be
            k = self.N-1
            self.x[k] = (-psi[k-1]+self.e[k]*psi[k]-self.x[k-1])/self.beta[k]
            psi[self.N-1] = self.x[self.N-1]
            for k in range(self.N-2,-1,-1):
                psi[k] = self.x[k]-self.gamma[k]*psi[k+1]
            t += self.dt
        return [psi,t]


    def V_marche(self,V0):
        V = numpy.zeros(self.N)
        P = int(self.N/2)
        V[P:self.N-1] = V0
        return V

    def V_barriere(self,V0,largeur):
        q = int(largeur/self.dx/2)
        V = numpy.zeros(self.N)
        P = int(self.N/2)
        V[P-q:P+q] = V0
        return V


    def paquet(self,x0,k0,sigma0):
        psi = numpy.zeros(self.N,dtype=numpy.complex)
        for k in range(1,self.N-1):
            x = self.dx*k
            psi[k] = cmath.exp(1j*k0*x)*math.exp(-(x-x0)*(x-x0)/(4*sigma0**2))
        return psi
```

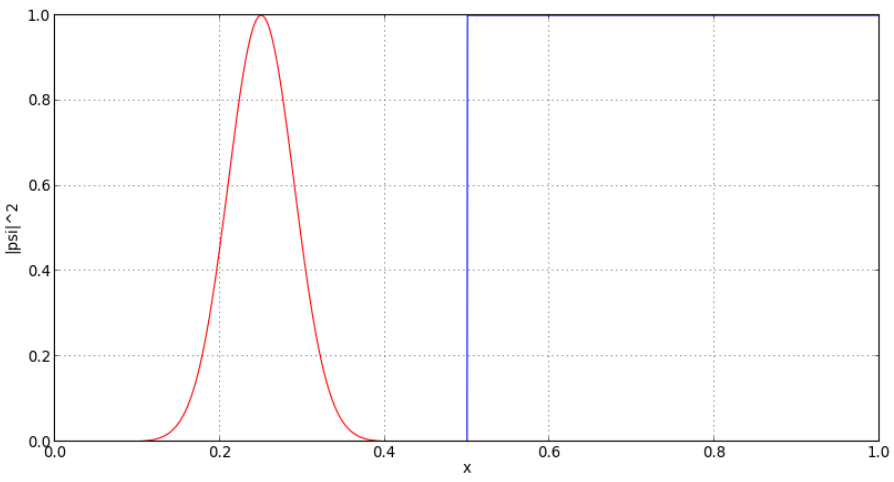
5. Exemples

5.a. Diffusion de particules sur une marche de potentiel

```
import numpy
from Schrodinger1D import Schrodinger1D
from matplotlib.pyplot import *

N=2000
L=1.0
dx=L/(N-1)
x = numpy.arange(N)*dx
schrodinger = Schrodinger1D(L,N)
V = schrodinger.V_marche(1.0)
l0 = 2e-2
k0 = numpy.pi*2/l0
E=k0**2
x0 = 0.25
sigma0 = 0.04
psi = schrodinger.paquet(x0,k0,sigma0)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```

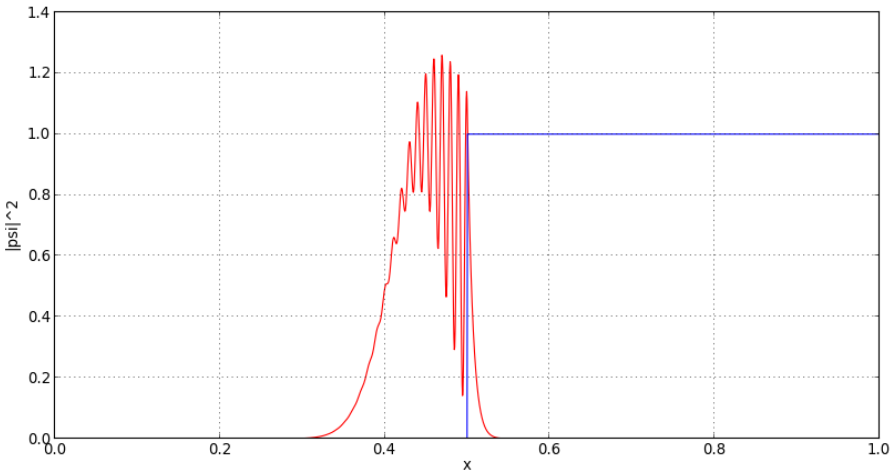


On trace le module de la fonction d'onde (densité de probabilité) pour différents instants.

```
t=0.0
dt=2*dx*dx
```

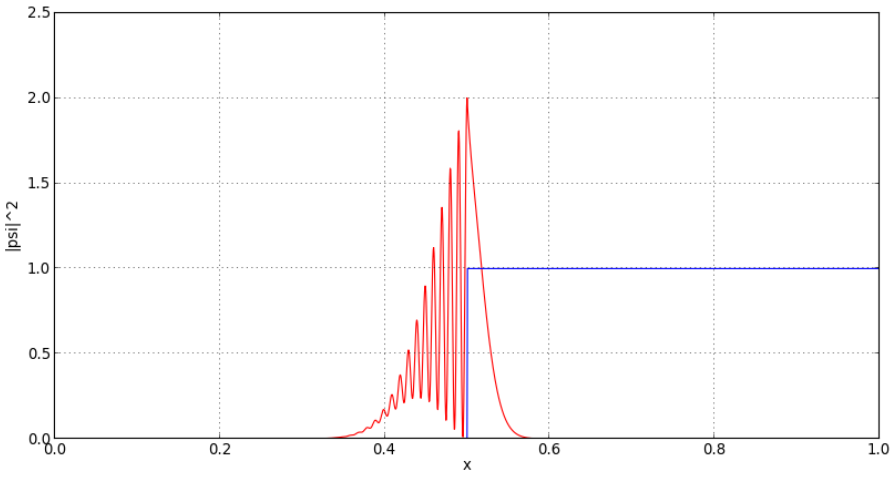
```
v=2*k0
T = 1.0/v
V0 = E
schrodinger.config(dt,V*V0)
[psi,t] = schrodinger.iterations(psi,t,T*0.2)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```



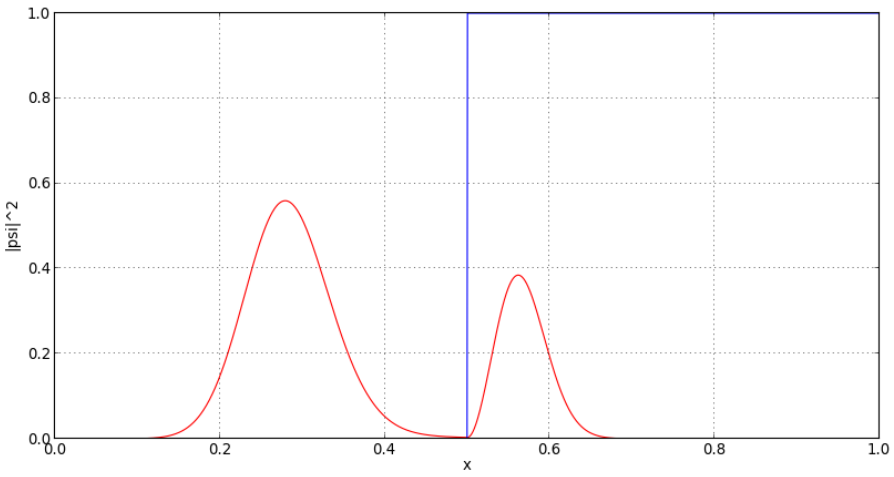

```
[psi,t] = schrodinger.iterations(psi,t,T*0.3)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```




```
[psi,t] = schrodinger.iterations(psi,t,T*0.5)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```



5.b. Diffusion de particules sur une barrière de potentiel

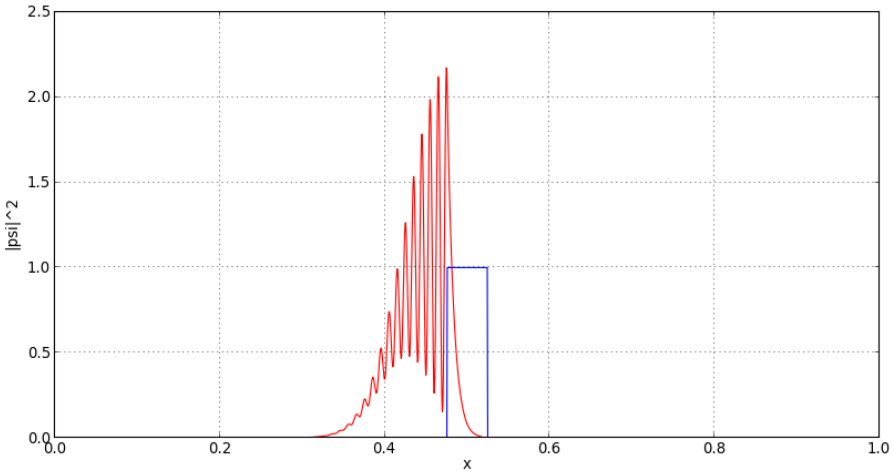
```
N=2000
L=1.0
dx=L/(N-1)
x = numpy.arange(N)*dx
schrodinger = Schrodinger1D(L,N)
V = schrodinger.V_barriere(1.0,0.05)
l0 = 2e-2
```

```
k0 = numpy.pi*2/10
E=k0*k0
x0 = 0.25
sigma0 = 0.04
psi = schrodinger.paquet(x0,k0,sigma0)
```

On trace le module de la fonction d'onde (densité de probabilité) pour différents instants.

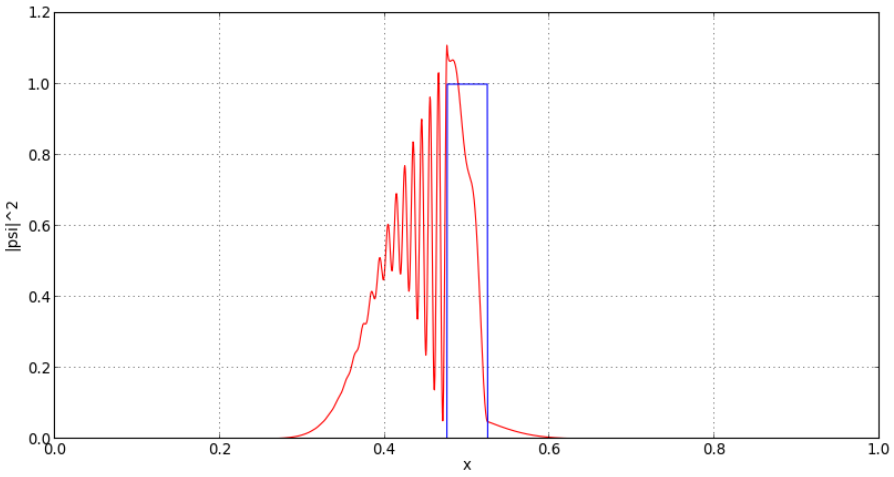
```
t=0.0
dt=2*dx*dx
v=2*k0
T = 1.0/v
V0=E
schrodinger.config(dt,V*V0)
[psi,t] = schrodinger.iterations(psi,t,T*0.2)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```



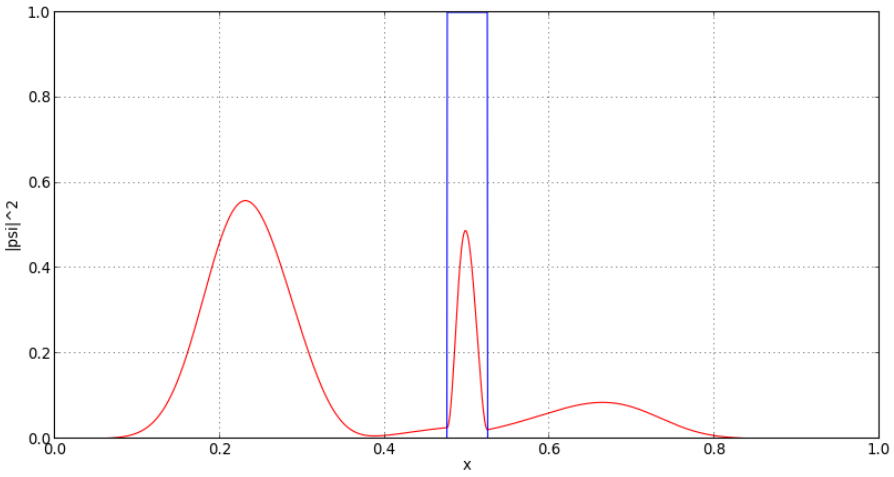

```
[psi,t] = schrodinger.iterations(psi,t,T*0.3)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```



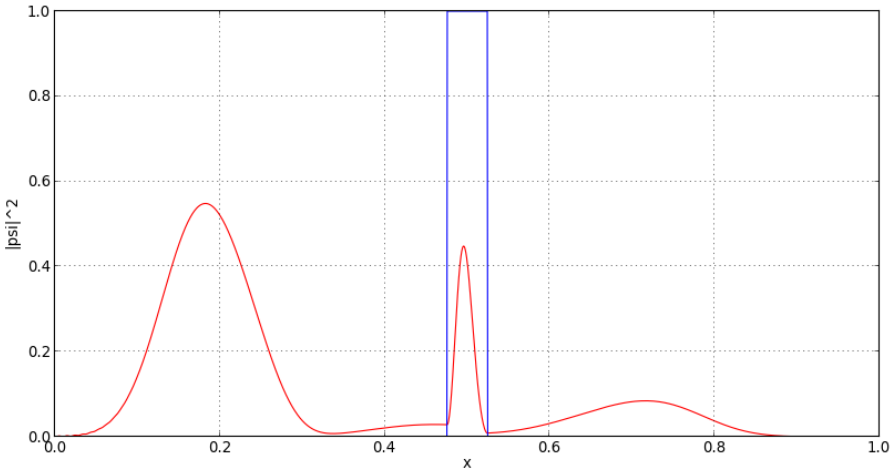

```
[psi,t] = schrodinger.iterations(psi,t,T*0.5)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```




```
[psi,t] = schrodinger.iterations(psi,t,T*0.55)

figure(figsize=(12,6))
plot(x,psi*numpy.conj(psi),'r')
plot(x,V,'b')
xlabel("x")
ylabel("|psi|^2")
grid()
```



Références

[1] J.J. Sakurai, *Modern quantum mechanics*, (Addison-Wesley, 1985)
[2] A. Goldberg, H.M. Schey, J.L. Schwartz, *Computer-generated motion pictures of one-dimensional quantum-mechanical transmission and reflection phenomena*, (Am. J. Phys, 35(3), 1967)
[3] C. Cohen-Tannoudji, B. Diu, F. Laloë, *Mécanique quantique*, (Hermann, 1980)