Segona entrege de Dinàmica Quàntica

Jordi Torrents 12/03/2020

En aquesta pràctica simulem la col·lisió d'un paquet d'ones gaussià contra una barrera de potencial quadrada i en mesurem els coeficients de transmissió i de rebot, T i R respectivament.

El mecanisme per la construcció del paquet així com el mètode de Backward Euler per integrar l'equació de Schrödinger ja van ser descrites en la pràctica anterior.

En aquesta pràctica canviem el mètode de integració i utilitzem el mètode Runge-Kutta de 4t ordre ja que dona uns resultats molt més bons.

La idea serà executar la quantitat suficient de passos de temps com perquè el paquet s'hagi dividit entre la part que travessa la paret i la part que en rebota. Això ho fem utilitzant les simulacions animades de la pràctica anterior i apuntant a quin temps t es dona aquest fenomen. Un cop ho tinguem inicialitzarem el la funció d'ona Ψ, crearem la barrera desitjada i calcularem $int(t/dt)$ passos de temps.

Els resultats es mostren en les següents figures

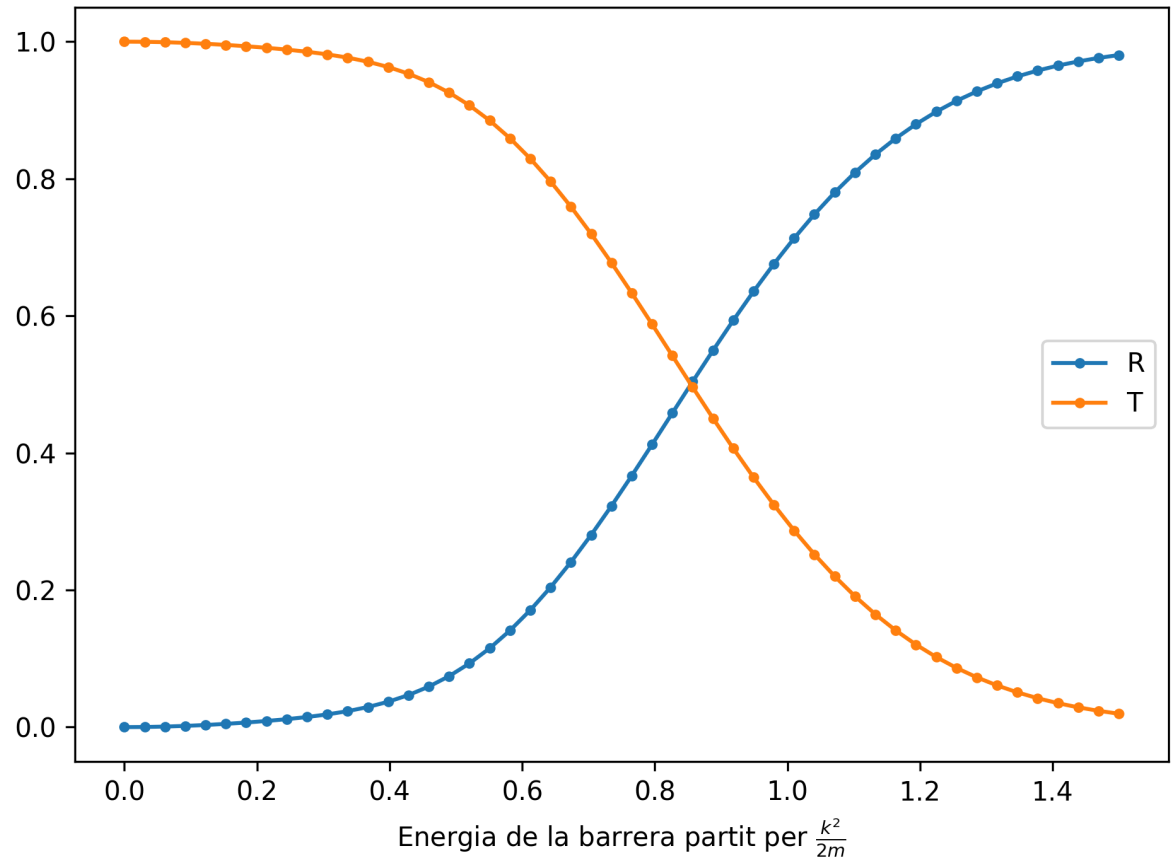

Figure 1: Escanjat dels coeficients R i T variant l'energia de la barrera per una amplada igual a $L/20$

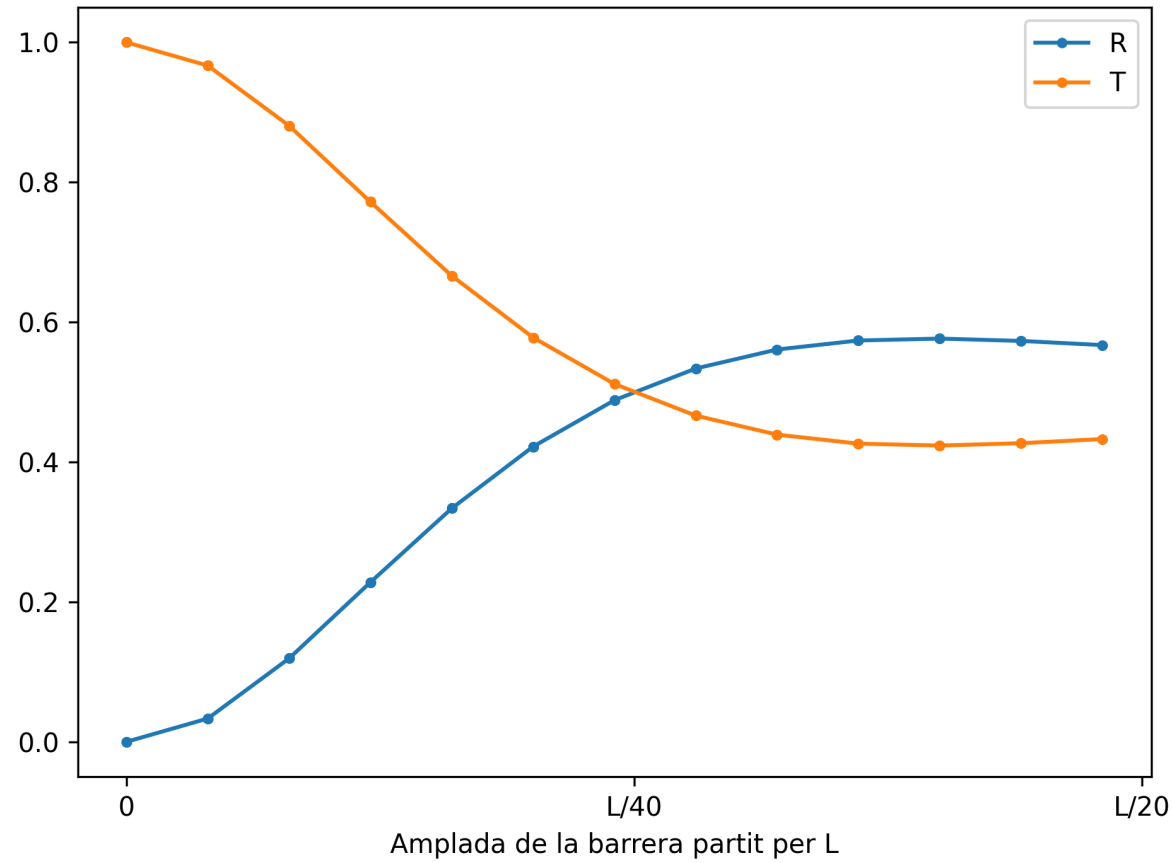

Figure 2: Escanjat del coeficients R i T variant l'amplada de la barrera per una energia igual a $0.9E$ on E és l'energia del paquet d'ones. Al ser una barrera estreta, es fa notoria la discretització de l'espai i com això afecta als possibles valors de l'amplada que es poden simular

Seguidament es presenta el codi utilitzat en Python i la llibreria personal *numfor* utilitzada.

```
import numpy as np
import matplotlib.pyplot as plt
import numfor as nf

def psi_0(x):
    return (a**2/(2*np.pi))**(1/4)*np.exp(-0.25*(a*(x-x0))**2)*np.exp(1j*k*x)

def R():
    return np.sum(prob[:int(n/2)])*dx

def T():
    return np.sum(prob[int(n/2):])*dx

a = 5
n = 1000
k = 20
```

```
w = 2

m  = 1
L  = 5
x0 =-L/2
V  = np.zeros(n)
x  = np.linspace(-L, L, n)
dx = (2*L)/(n-1)
dt = 2/(2/(m*dx**2)+np.max(V))

R_arr, T_arr = np.empty(50), np.empty(50)
ample  = int(0.5*(n/2)/20)
alcada = np.linspace(0, 1.5, 50)
for i in range(50):
    psi = psi_0(x)
    V   = np.zeros(n)
    V[int(n/2)-ample:int(n/2)+ample]=alcada[i]*k**2/(2*m)
    nf.euler_rk4(psi=psi, steps=int(0.35/dt), dt=dt, m=m, v=V, dx=dx)
    prob = np.abs(psi)**2
    R_arr[i], T_arr[i] = R(), T()


plt.figure(dpi=300)
... ... ...
plt.savefig('barrera_energia.png')


ample = np.linspace(0, 0.5*(n/2)/20, 50).astype(int)
for i in range(50):
    psi = psi_0(x)
    V   = np.zeros(n)
    V[int(n/2)-ample[i]:int(n/2)+ample[i]]=0.9*k**2/(2*m)
    nf.euler_rk4(psi=psi, steps=int(0.35/dt), dt=dt, m=m, v=V, dx=dx)
    prob = np.abs(psi)**2
    R_arr[i], T_arr[i] = R(), T()

plt.figure(dpi=300)
... ... ...
plt.savefig('barrera_amplada.png')
```

Listing 1: Fortran personal library (numfor)

```
subroutine Euler_RK4(psi, n, steps, V, dt, m, dx)
  integer                    :: n, steps, i
  complex(8), dimension(n)   :: psi, Psi_temp
  complex(8), dimension(n-2) :: k1, k2, k3, k4
  complex(8)                 :: ii
  real(8), dimension(n)      :: V
  real(8)                    :: dt, m, dx
  ii = cmplx(0,1)

  Psi_temp(1) = cmplx(0)
  Psi_temp(n) = cmplx(0)
  do i=1,steps

    k1 = ii*dt*((Psi(:n-2)+Psi(3:)-2.d0*Psi(2:n-1))/(2.d0*m*dx**2)-V(2:n-1)*Psi(2:n-1))

    Psi_temp(2:n-1) = Psi(2:n-1) + 0.5d0*k1

    k2 = ii*dt*((Psi_temp(:n-2)+Psi_temp(3:)-2.d0*Psi_temp(2:n-1))/(2.d0*m*dx**2)-V(2:n-1)*Psi_temp(2:n-1))

    Psi_temp(2:n-1) = Psi(2:n-1) + 0.5d0*k2

    k3 = ii*dt*((Psi_temp(:n-2)+Psi_temp(3:)-2.d0*Psi_temp(2:n-1))/(2.d0*m*dx**2)-V(2:n-1)*Psi_temp(2:n-1))

    Psi_temp(2:n-1) = Psi(2:n-1) + k3

    k4 = ii*dt*((Psi_temp(:n-2)+Psi_temp(3:)-2.d0*Psi_temp(2:n-1))/(2.d0*m*dx**2)-V(2:n-1)*Psi_temp(2:n-1))

    psi(2:n-1) = psi(2:n-1)+(k1+2.d0*k2+2.d0*k3+k4)/6.d0
  end do

end subroutine
```