Advanced Simulation Techniques in Physics

EP 438 / PH 810

Simulation of One-Dimensional Quantum Tunneling Effect

Author:	*Roll Numbers:*
Abhinav Kumar	140260021
Soham Mukhopadhyay	16K410002
Anand Kumar	16K410007

Abstract

In this report we simulate the time development of one-dimensional quantum mechanical system. In this study we mainly concern about transmission and reflection phenomena. In order to formulate this problem, we consider a plane wave incidence on a step potential and animate its probability of transmission. Second we consider the phenomena of reflection and transmission of a Gaussian wave packet impinging on a square well potential barrier. The wave equation is solved numerically by reducing the second order ordinary differential equation into set of difference equations which gives a recursion relation. By this recursion relation we estimate variation of wave function with time and space. Photographs of the wave packets vs positions at various time and for different energies are reported.

Contents

1 Introduction

Quantum Tunneling is a phenomenon where particle has a finite probability to tunnel through a barrier and the barrier can be an insulator, a vacuum or it can be a region of high potential.Tunneling is a quantum mechanical phenomena and thus important for small mass particles in which classical laws break down.
The principle of quantum tunneling can be employed to explain various physical process. Alpha particle decay is one of the radioactive decay process which explain can be explain by quantum tunneling. This has important applications to modern devices such as quantum computing and scanning tunneling microscopy.

2 Algorithm used to solve the problem

2.1 Difference Equation

The Schrodinger Equation in one dimension in operator form is represented as:

$$-H\psi = [\partial^2/\partial x^2 + V(x)]\psi(x,t) = -i\partial\psi/\partial t \tag{1}$$

We need to convert equation(1) into a difference equation. For that we denote temporal part by a superscript (n) and spatial part by a subscript (j) to the wavefunction ψ. Thus now, instead of $\psi(x,t)$, we have ψ_j^n.

Taylor expanding in spatial dimension(where ϵ is the infinitesimal spatial displacement):

$$\psi_{j+1} = \psi_j + \epsilon\psi_j' + (1/2)\epsilon^2\psi_j'' + (1/6)\epsilon^3\psi_j''' + O(\epsilon^4) \tag{2}$$

$$\psi_{j-1} = \psi_j - \epsilon\psi_j' + (1/2)\epsilon^2\psi_j'' - (1/6)\epsilon^3\psi_j''' + O(\epsilon^4) \tag{3}$$

From these two equations, we obtain :

$$\psi_j'' = (1/\epsilon^2)[\psi_{j+1} - 2\psi_j + \psi_{j-1}] + O(\epsilon^2) \tag{4}$$

So, now we can write the Hamiltonian operator acting at a point $x = x_j = j\epsilon$ as:

$$-H\psi = (1/\epsilon^2)[\psi_{j+1} - 2\psi_j + \psi_{j-1}] - V_j\psi_j \tag{5}$$

For time development, the formal solution of Eq.(1) is given by:

$$\psi(x,t) = \exp(-i(t-t_0)H)\psi(x,t_0) \tag{6}$$

Using the temporal index (n), we can write:

$$\psi_j^{n+1} = \exp(-i\delta H)\psi_j^n \tag{7}$$

where δ represents infinitesimal time evolution. However, the difference equation arising out of eq.(7) by applying the H operator as outlined in eq.(5) gives rise to an explicit differencing scheme where we obtain ψ_j^{n+1} in terms of an wavefunction at an earlier time ψ_j^n. However, the expansion of $\exp(-i\delta H)$ can grow in time without bounds after each iteration and hence this differencing scheme is very unstable.

In order to circumvent this problem, we rewrite eq.(7) as:

$$\psi_j^n = \exp(i\delta H)\psi_j^{n+1} \tag{8}$$

So, correct to the terms of order δ, we have:

$$\psi_j^n = (1 + (i\delta H))\psi_j^{n+1} = \psi_j^{n+1} + i\delta H\psi_j^{n+1} \tag{9}$$

which leads to the difference equation:

$$\psi_j^n = \psi_j^{n+1} - (i\delta/\epsilon^2)[\psi_{j+1}^{n+1} - 2\psi_j^{n+1} + \psi_{j-1}^{n+1} - \epsilon^2 V_j \psi_j^{n+1}] \tag{10}$$

This differencing scheme is implicit and we must obtain ψ_j^{n+1} before so as to get ψ_n^j. The way to do so will be outlined shortly. However, the operator $(1 + i\delta H)$ is not unitary. Unitarity is a characteristic of the original wave equation which ensures that the normalization of the wavefunction does not change with time.

So, an unitary approximation to the operator $\exp(-i\delta H)$ is used which is known as the Cayley form:

$$[1 - 1/2(i\delta H)]/[1 + 1/2(i\delta H)] \tag{11}$$

So, applying this in place of $\exp(-i\delta H)$, we get

$$[1 + 1/2(i\delta H)]\psi_j^{n+1} = [1 - 1/2(i\delta H)]\psi_j^n \tag{12}$$

Consequently, we arrive at the equation:

$$-(i\delta/2\epsilon^2)\psi_{j+1}^{n+1} - (i\delta/2\epsilon^2)[(2\epsilon^2/i\delta)\psi_j^{n+1} - 2\psi_j^{n+1} - \epsilon^2 V_j\psi_j^{n+1}] - (i\delta/2\epsilon^2)\psi_{j-1}^{n+1} =$$

$$(i\delta/2\epsilon^2)\psi_{j+1}^{n} + (i\delta/2\epsilon^2)[(2\epsilon^2/i\delta)\psi_j^{n} - 2\psi_j^{n} - \epsilon^2 V_j\psi_j^{n}] + (i\delta/2\epsilon^2)\psi_{j-1}^{n}$$

which simplifies to:

$$\psi_{j+1}^{n+1} + [i\lambda - \epsilon^2 V_j - 2]\psi_j^{n+1} + \psi_{j-1}^{n+1} = -\psi_{j+1}^{n} + [i\lambda + \epsilon^2 V_j + 2]\psi_j^{n} - \psi_{j-1}^{n} \quad (13)$$

where $\lambda = 2\epsilon^2/\delta$

2.2 Recurrence Relation

To solve eq.(13), we define : $\Omega_j^n = -\psi_{j+1}^n + [i\lambda + \epsilon^2 V_j + 2]\psi_j^n - \psi_{j-1}^n$

Then Eq.(13) becomes,

$$\psi_{j+1}^{n+1} + [i\lambda - \epsilon^2 V_j - 2]\psi_j^{n+1} + \psi_{j-1}^{n+1} = \Omega_j^n \quad (14)$$

Now, we make the assumption that,

$$\psi_{j+1}^{n+1} = e_j^n \psi_j^{n+1} + f_j^n \quad (15)$$

where e_j^n and f_j^n are two auxiliary functions defined by this equation. Substituting eq.(15) into eq.(14) we get,

$$\psi_j^{n+1} = \frac{\psi_{j-1}^{n+1}}{2 + \epsilon^2 V_j - e_j^n - i\lambda} + \frac{f_j^n - \Omega_j^n}{2 + \epsilon^2 V_j - e_j^n - i\lambda} \quad (16)$$

This equation is similar to eq.(15) except the index j replaced by $j-1$.Comparing eq.(15) and eq.(16) we get:

$e_{j-1}^n = \frac{1}{2+\epsilon^2 V_j - e_j^n - i\lambda}$ and $f_{j-1}^n = e_{j-1}^n(f_j^n - \Omega_j^n)$

Solving for e_j^n and f_j^n in terms of e_{j-1}^n and f_{j-1}^n we get:

$$e_j^n = 2 + \epsilon^2 V_j - i\lambda - \frac{1}{e_{j-1}^n} \quad (17)$$

$$f_j^n = \Omega_j^n + \frac{f_{j-1}^n}{e_{j-1}^n} \tag{18}$$

These equations are the recursion relations for f and e functions. Note that, e_j^n is not dependent on time , but f_j^n is time dependent as we have Ω_j^n on the R.H.S of eq.(18).

To solve these recursion relations, we have to define the boundary conditions of the system. The wavefunction must vanish at the end points of the box.So, $\psi(0,t) = \psi(L,t) = 0$ for all t. Now, $0 \le x \le L$ and if $L = J\epsilon$ (where J is a relatively large integer) , then the boundary conditions can be written as $\psi_J^n = \psi_0^n = 0$ for all n.

Then, for j=1 , eq.(14) becomes : $\psi_2^{n+1} = (2 + \epsilon^2 V_1 - i\lambda)\psi_1^{n+1} + \Omega_1^n$

By comparing with eq.(15), we get: $e_1 = 2 + \epsilon^2 V_1 - i\lambda$ and $f_1^n = \Omega_1^n$

Now, using the recursion relations in eq.(17) and eq.(18), we can get the values of $e_2, e_3, e_4, e_5....$ and $f_2^n, f_3^n, f_4^n, f_5^n.....$ for all n.

Using the boundary condition, $\psi_J^n = 0$ for all n, and writing eq.(15) for $j = J-1$, we have:

$$\psi_{J-1}^{n+1} = -\frac{f_{J-1}^n}{e_{J-1}} \tag{19}$$

noting that $\psi_J^n = 0$ for all n. Eq.(15) can also be expressed as :

$$\psi_j^{n+1} = \frac{\psi_{j+1}^{n+1} - f_j^n}{e_j} \tag{20}$$

Now, using eq.(19) and eq.(20), we can determine $\psi_{J-2}^{n+1}, \psi_{J-3}^{n+1}......$ till ψ_1^{n+1}, while $\psi_0^{n+1} = 0$.

Thus, we can begin with the wavefunction ψ_j^0 $[\psi(x,0)]$ with $(j = 0, 1, 2, 3, 4, 5....J)$ for all n. This enables us to determine Ω_j^0 from which the rest follows.

2.3 Defining the wavepacket

We treat the reflection-transmission event by choosing the initial state of the particle as a Gaussian wave packet. Thus we have,

$$\psi(x,0) = exp(ik_0x)exp(-(x-x_0)^2/2\sigma_0^2)$$

The above wave packet is centered about $x = x_0$ with a spreading in x which is governed by σ_0. The factor in front of Gaussian function makes our initial wave function move towards potential barrier. Since we are quantizing in large box of length L, so at the edges wave function should vanish. So in order to make the wave function at edges zero we have to choose x_0 and σ_0 in such a way that, $\psi(0,0) = 0$ and $\psi(L,0) = 0$.

Choosing $x_0 = L/4$ we find that:

$$\psi(x,0) = exp(ik_0x)exp(-(x-1/4)^2/2\sigma_0^2) \tag{21}$$

Assuming the spreading in wave packet to be 5% of the box length that is $\sigma_0 = L/20$. Then, $\mid \psi(0,0) \mid = \exp(-12.5) \lesssim 4 \times 10^{-6}$. This can be approximated to zero in order to meet $\psi(0,0)$ equal to zero and $\psi(L,0)$ equal to zero.

Thus Gaussian wave packet is compatible to with the box normalization. In order to avoid the difficulties arises from box normalization we need to impose two restrictions on the wave packet. First the wave packet should not be allowed to travel so far that it hits the wall of the box. For this we allow the wave packet to move from $L/4$ to $3L/4$. This is done by taking the average velocity of wave packet to be $v_0 = 2k_0 \approx L/2T$, where $T = N\delta$ is total time require by wave packet to move from $L/4$ to $3L/4$.

Thus, $2k_0 \approx J\epsilon/2N\delta = J\lambda/4N\epsilon$, where $L = J\epsilon$ and $\lambda = 2\epsilon^2/\delta$.

Second restriction we need to impose on spreading of the wave packet in the course of time. Reflected and transmitted wave packet should be localized so that they are out of the region of the potential and still far from the walls of the box. If a Gaussian wave packet have initial spreading σ_0 than spreading in the wave packet with time is given by:

$$\sigma^2 = (\sigma_0^4 + 4T^2)^{1/2} = (\sigma_0^4 + 16N^2\epsilon^4/\lambda^2)^{1/2}$$

In this way we have constructed a reflection and transmission event of a Gaussian wave packet from a potential barrier.

3 Simulation Results

This section illustrates the results we obtained from running the python code. All the following sets of images are for Gaussian wave-packets with k = 0.596, spread = 1.5 and sigma = 15 whereas for plane wave, E =0.5

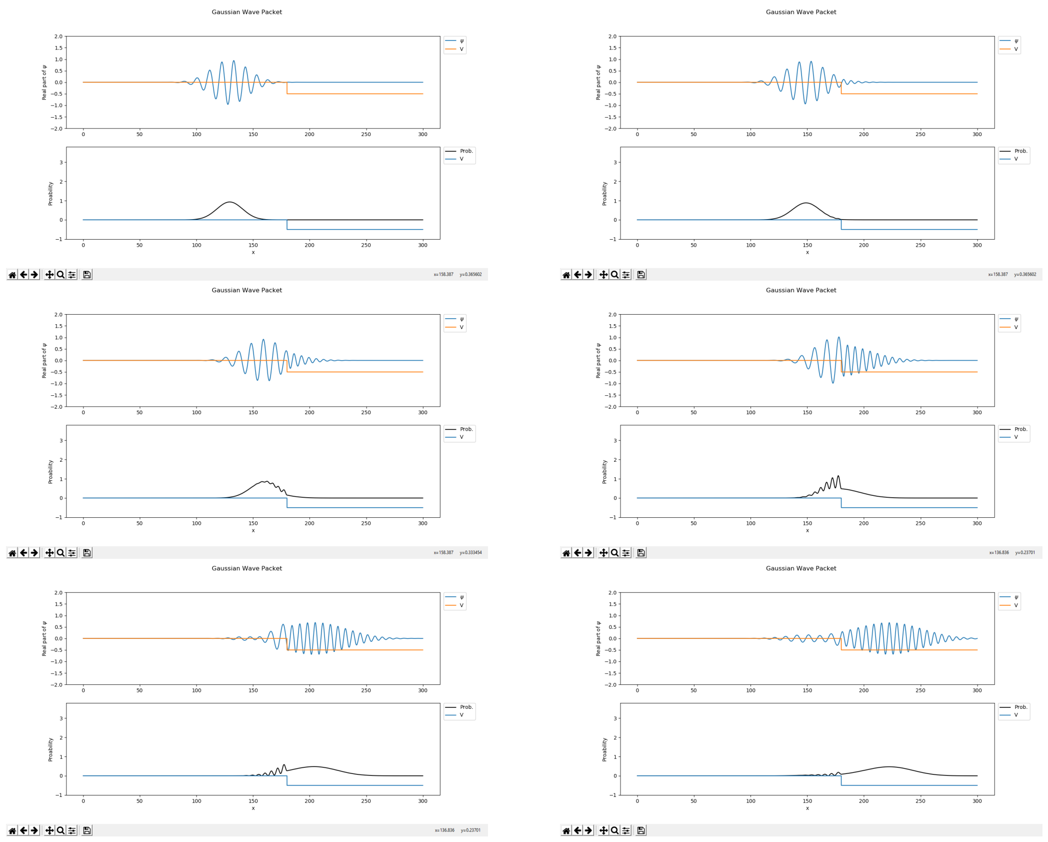

Figure 1: For Gaussian wave-packet with **k** = 0.596 and negative step potential V = - 0.5

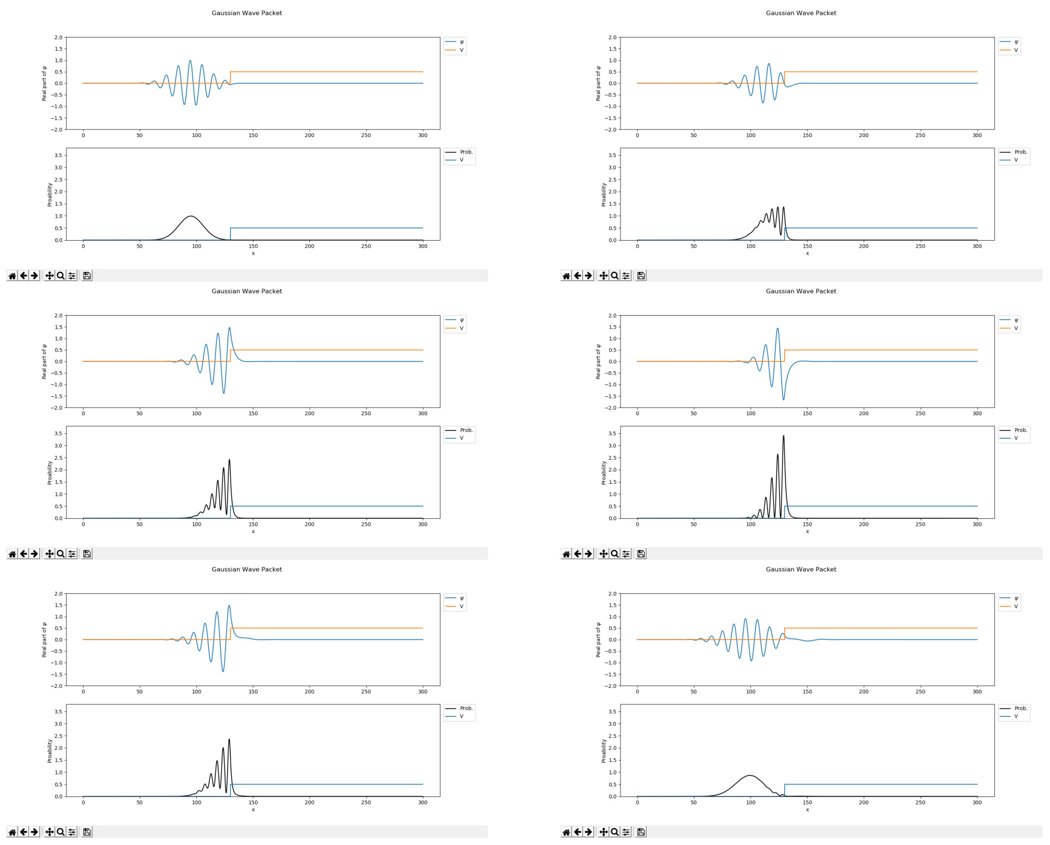

Figure 2: For Gaussian wave-packet with $\mathbf{k} = 0.596$ and step potential V = 0.5

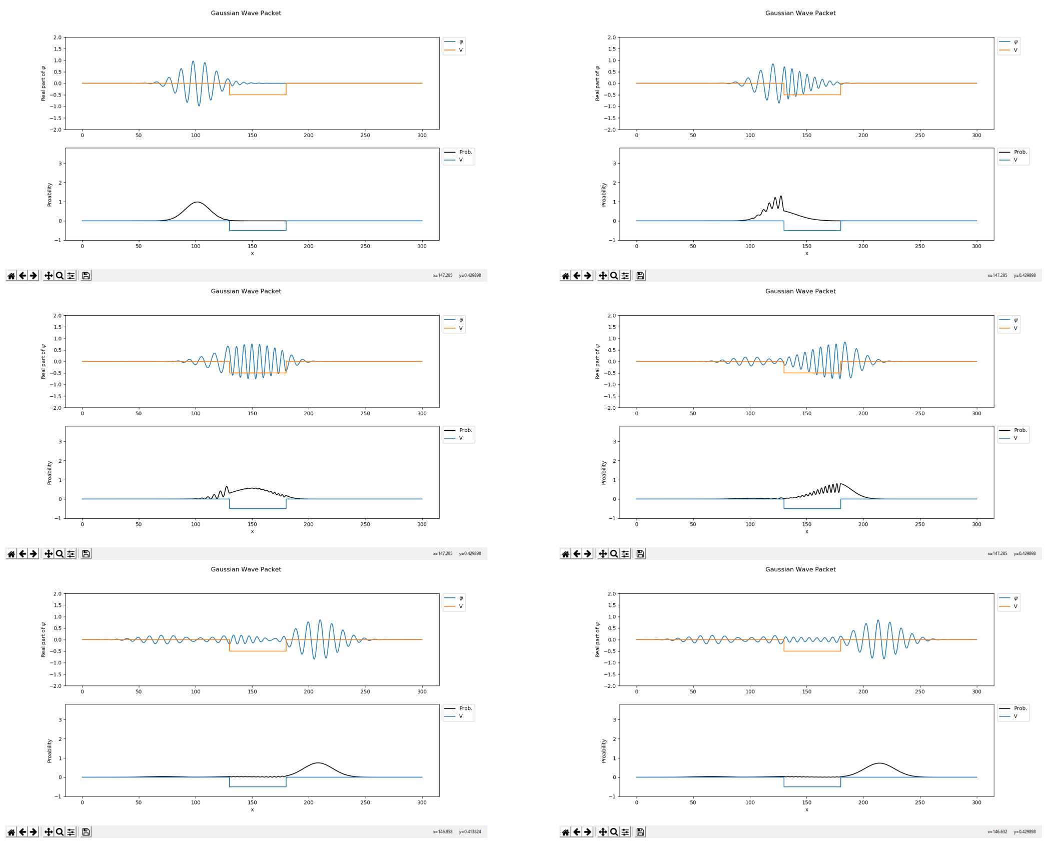

Figure 3: For Gaussian wave-packet with **k** = 0.596 and well potential V = - 0.5

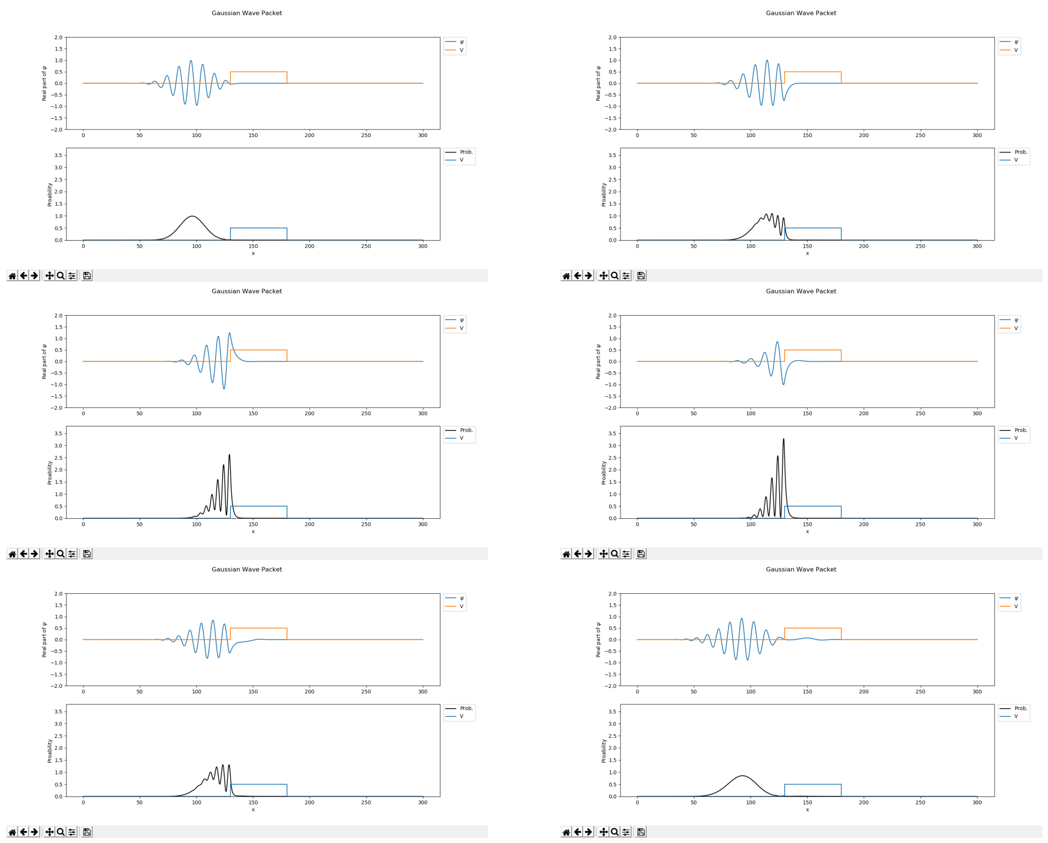

Figure 4: For Gaussian wave-packet with **k** = 0.596 and barrier potential V = 0.5

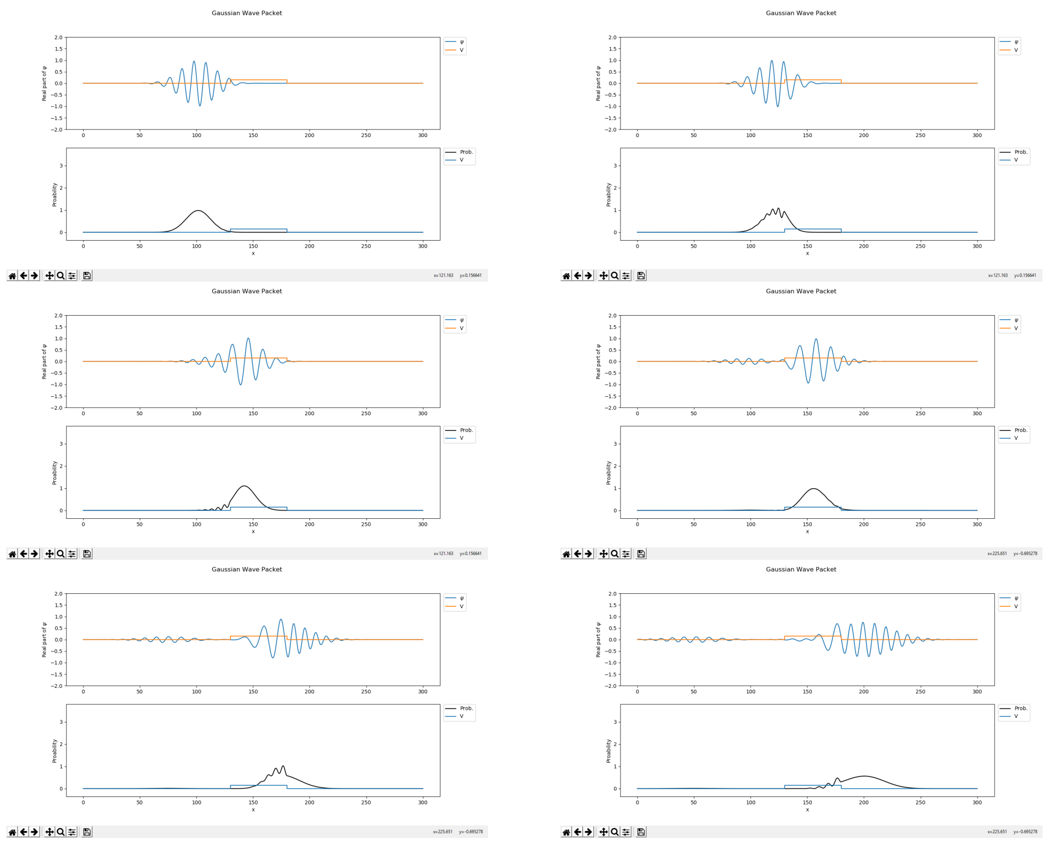

Figure 5: For Gaussian wave-packet with $\mathbf{k}$ = 0.596 and negative step potential V = 0.15

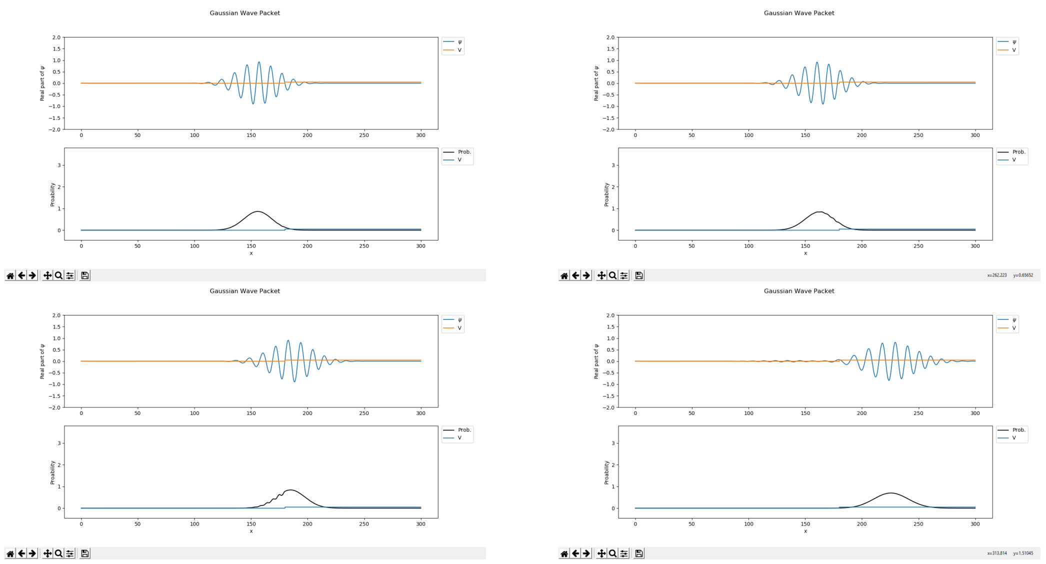

Figure 6: For Gaussian wave-packet with $\mathbf{k} = 0.596$ and step potential V = 0.05

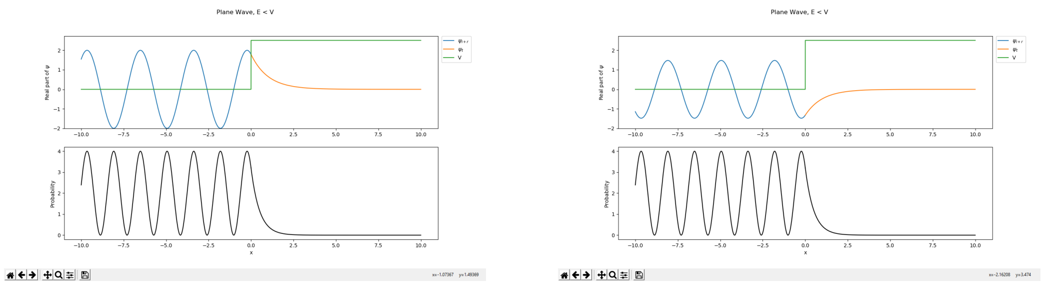

Figure 7: A plane wave with kinetic energy in free space, E = 1.5 and step potential V = 2.5

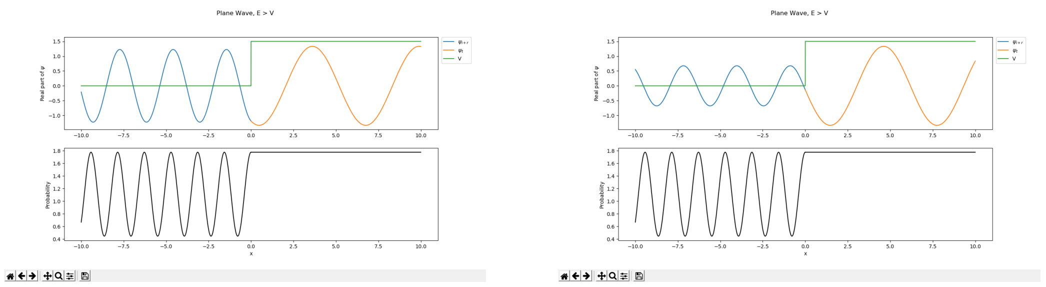

Figure 8: A plane wave with kinetic energy in free space, E = 2.0 and step potential V = 1.5

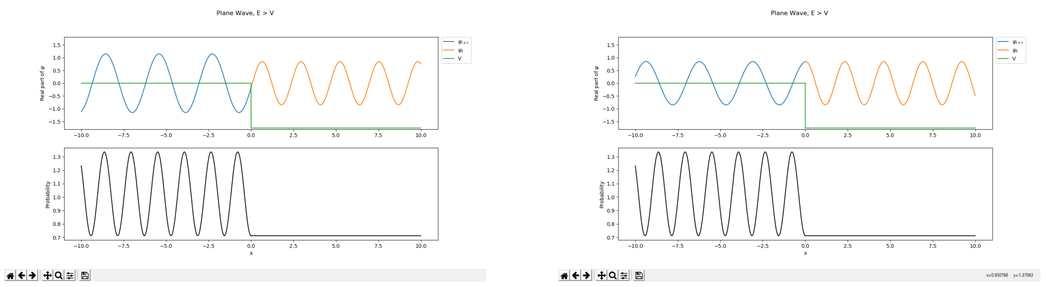

Figure 9: A plane wave with kinetic energy in free space, E = 2.0 and step potential V = -1.75

4 Reference

Abraham Goldberg, Harry M. Schey, Judah L. Schwartz. , Computer-Generated Motion Pictures of One-Dimensional Quantum-Mechanical Transmission and Reflection Phenomena,March 1967, *American Journal of Physics*, Volume 35, Number 3

5 Python Codes

5.1 Code for square potential barrier/well with incident Gaussian wavefunction

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import *
import matplotlib.animation as animation
import cmath

pi = np.pi
h_x = 0.01
h_t = 0.5

x0 = 0.0
xe = 300.0

L = abs(xe-x0)              # Length of the box

N = int(abs(xe-x0)/h_x)

sigma = L/20.0              # Spreading in the Gaussian wave packet

V0 = 1.50                   # Height of the  potential barrier
lamda = 2.0*h_x**2/h_t

spread = 1.5
total_time = lamda*(sigma**2)*((spread**2 - 1.0)**0.5)/(4.0*h_x**2)
```

```
k = (N+1)*lamda/(8*h_x*total_time)

xv = 130.0

x = np.arange(x0,xe,h_x)

#        Values of the psi at the boundry of box
psi = np.exp(1j*k*x)*np.exp(-(x-L/4)**2/(2.0*sigma**2))
psi[0] = complex(0.0,0.0)
psi[N-1] = complex(0.0,0.0)

#        Potential barrier
V = []
for i in range(N):
        if      (x0 + i*h_x) >= 180.0:
                V.append(V0)

        else:
                V.append(0.0)

#        Omega
def om(psi):
        global V
        oma = [-psi[1]]
        for i in range(1,N):
                if i != N-1:
                        oma1 = psi[i+1]
                else:
                        oma1 = 0.0
                oma.append(-oma1 + (2.0 + h_x**2 * V[i] + 1j*lamda)*
                    psi[i] - psi[i-1])
        z=np.arange(N)
        return oma
omega = om(psi)

def v1v2(psi,omega):
        global V
        v1 = [complex(0.0,0.0)]
        v2 = [complex(0.0,0.0)]
        v1.append(2.0 + h_x**2 * V[1] - 1j*lamda)
        v2.append(omega[1])
        v1[0] = 0.0
        v2[0] = v1[0]*(v2[1] - omega[1])
        for i in range(2,N):
                v1.append(2.0 + h_x**2 * V[i] - 1j*lamda - 1/v1[i-1])
                v2.append(omega[i] + v2[i-1]/v1[i-1])
        return v1,v2
v1,v2 = v1v2(psi,omega)

fig= figure()
fig.suptitle("Gaussian_Wave_Packet")

ax01 = subplot2grid((2,1), (0, 0))
ax03 = subplot2grid((2,1), (1, 0))



ax01.set_ylabel("Real_part_of_$\psi$")
ax03.set_xlabel("x")
ax03.set_ylabel("Proability")

wave1, pot1 = ax01.plot(x,psi.real,x,V)
wave1.set_label('$\psi$')
pot1.set_label('V')
ax01.legend(bbox_to_anchor=(1.01, 1), loc=2, borderaxespad=0.)
ax01.set_ylim([-2,2])

wave2, pot2 = ax03.plot(x,abs(psi)**2,'xkcd:black',x,V)
wave2.set_label('Prob.')
pot2.set_label('V')
ax03.legend(bbox_to_anchor=(1.01, 1), loc=2, borderaxespad=0.)
ax03.set_ylim([V0 - 0.5,3.8])


def animate(t):
        if t == 0:
            return wave1,wave2
        global psi,v1,v2,omega,x

        psi[0] = complex(0.0)
        psi[N-1] = complex(0.0)
        psi[N-2] = -v2[N-2]/v1[N-2]
        for i in range(N-3,0,-1):
            psi[i] = (psi[i+1] - v2[i])/v1[i]


        omega = om(psi)

        v1,v2 = v1v2(psi,omega)

        wave1.set_data(x,psi.real)
        wave2.set_data(x,abs(psi)**2)

        return wave1, wave2

ani = animation.FuncAnimation(fig,animate,None, interval = 1)

plt.show()
```

5.2 Code for Step potential with incident plane wave-function

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pylab import *
import matplotlib.animation as animation

E = 2.0           # Energy of the incidence particle
Vc = 1.75         # Higth of the potential barrier
V_x = 0
m = 1.0           # Mass of incident particle
h_bar = 1.0
x_le = -10.0
x_re = 10.0

k1 = (2.0*m*E)**0.5/h_bar
if Vc > E:
        k2 = (2.0*m*(Vc-E))**0.5/h_bar
        w_exp = True
else:
        k2 = (2.0*m*(E-Vc))**0.5/h_bar
        w_exp = False

w1 = E
w2 = E

A = 1.0
if w_exp == False:
        B = A*(k1-k2)*np.exp(2.0*1j*k1*V_x)/(k1+k2)
        C = A*2.0*k1*np.exp(1j*(k1-k2)*V_x)/(k1 + k2)
else:
        B = A*(k1 - 1j * k2)*np.exp(2.0*1j*k1*V_x)/(k1 + 1j * k2)
        C = A*2.0*k1*np.exp((1j*k1+k2)*V_x)/(k1 + 1j * k2)

h = 0.001
```

```
x_l = np.arange(x_le ,V_x,h)
x_r = np.arange(V_x,x_re ,h)
x = np.arange(x_le ,x_re ,h)

n = len(x)


psi_i = A*np.exp(1j*k1*x_l)

psi_r = B*np.exp(-1j*k1*x_l)

if w_exp == False:
        psi_t = C*np.exp(1j*k2*x_r)
else:
        psi_t = C*np.exp(-k2*x_r)

psi_c = psi_i + psi_r

def potential(x):
        if x  >= V_x:
                return Vc
        else:
                return 0.0

V = []
for i in range(n):
        V.append(potential(x_le + i*h))

fig= figure()
if E > Vc:
        fig.suptitle("Plane␣Wave,␣E␣>␣V", fontsize=12)
else:
        fig.suptitle("Plane␣Wave,␣E␣<␣V", fontsize=12)

ax01 = subplot2grid((2,1), (0, 0))
ax03 = subplot2grid((2,1), (1, 0))

ax01.set_ylabel("Real␣part␣of␣$\psi$")
```

```
ax03.set_xlabel("x")
ax03.set_ylabel("Probability")


wave_c, wave_t, pot = ax01.plot(x_l,psi_c.real, x_r,psi_t.real, x,V)
wave_c.set_label('$\psi_{i+r}$')
wave_t.set_label('$\psi_t$')
pot.set_label('V')
ax01.legend(bbox_to_anchor=(1.01, 1), loc=2, borderaxespad=0.)
ax01.set_ylim([-1.8,1.8])

wave_a1, wave_a2 = ax03.plot(x_l,abs(psi_c)**2,'xkcd:black',x_r,
                              abs(psi_t)**2,'xkcd:black')
plt.xlabel('x')
plt.ylabel('Probability')

def animate(t):
        global psi_c
        global psi_t
        global x,x_l,x_r

        psi_c_t = psi_c*np.exp(-1j*w1*t/100)
        wave_c.set_data(x_l,psi_c_t.real)
        wave_a1.set_data(x_l,abs(psi_c_t)**2)

        psi_t_t = psi_t*np.exp(-1j*w1*t/100)
        wave_t.set_data(x_r,psi_t_t.real)
        wave_a2.set_data(x_r,abs(psi_t_t)**2)

        return wave_c,wave_t,wave_a1,wave_a2



ani = animation.FuncAnimation(fig, animate,None,interval=50)

plt.show()
```