Selected Topics In Computational Physics
Final Project

Umur Can Kaya
090140107
(Dated: March 30, 2018)

The question is; for a system given in FIG.1(*above*), when will the particle escape from region 1 by tunneling? In other words, when will the particle comepelitely radiate away from region 1? But since there are hard to deal unbound states in region 3, the original problem is approximated as shown in FIG.1(*below*) where L is finite but still $\gg a$. After this approximation since the particle can not completely escape from region 1, relaxation time of the system is calculated numerically with python and interpreted as escaping time.

INTRODUCTION

Consider a quantum mechanical particle confined in the following piecewise continuous potential $V(x)$:

$$V(x) = \begin{cases} 0, & 0 < x < a \\ V_0, & a \leq x \leq a + w \\ 0, & a + w < x < L \\ \infty, & \text{elsewhere} \end{cases}$$

Initially the particle is in the state

$$\psi(t = 0, x) = \sqrt{\frac{2}{a}} \sin\Big(\frac{\pi x}{a}\Big) \tag{1}$$

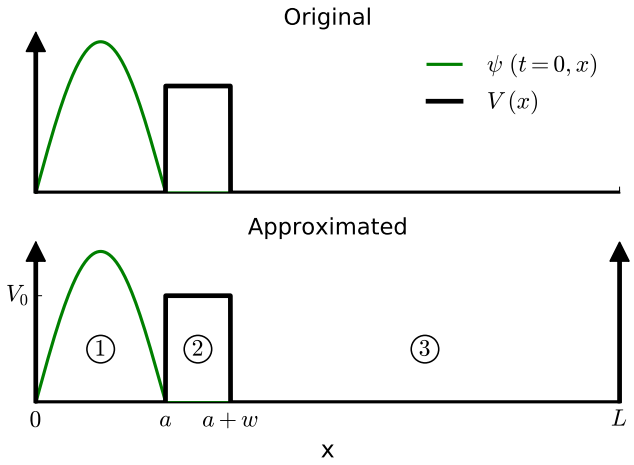

FIG. 1. Plot of $V(x)$ and $\psi(t = 0, x)$ versus position x. ①, ② and ③ are indicating the regions 1, 2 and 3 respectively. *Above* is the original problem and *below* is the approximated form as mentioned in the abstract.

Time evaluation of this system is described by time dependent Schrödinger equation:

$$i\hbar \frac{\partial}{\partial t'}\psi = H\psi, \qquad H = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x'^2} + V(x') \tag{2}$$

It is useful to define a unit system in order to nondimensionalize Eq (2) as $x = x'/a$ and $t = t'/\tau$ where $a = \sqrt{2mV_0}/\hbar$ and $\tau = \hbar/V_0$. After this translation new form of time dependent Schrödinger equation becomes:

$$i\frac{\partial}{\partial t}\psi = H\psi, \qquad H = -\frac{\partial^2}{\partial x^2} + 1 \tag{3}$$

Functions $u_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right)$ is choosen as basis for constructing the hamiltonian since they form a complete orthonormal set. Therefore the matrix element of the hamiltonian is written as:

$$H_{mn} = \int_0^L u_m H u_n \, dx \tag{4}$$

Functions ϕ_n satisfies the time independent part of the Schrödinger equation.

$$H\phi_n = E_n\phi_n$$

In position basis, these ϕ_n functions are written as:

$$\phi_n(x) = \sum_i v_{in} u_i \tag{5}$$

Where v_{in} is the n^{th} element of the i^{th} energy eigenvector of H. Therefore the solution of time dependent Schrödinger equation can be written in terms of $\phi_n(x)$ as given below.

$$\psi(t, x) = \sum_n c_n \phi_n(x) e^{-iE_n t} \tag{6}$$

Performing a Fourier trick on Eq. (6) at time $t = 0$ will yield the c_n coefficients.

$$c_n = \int_0^L \psi(t = 0, x)\phi_n(x)dx \tag{7}$$

RESULTS

For a system with properties $w = 0.1a$ and $L = 20.1a$, system is simulated as explained above and the results are shown in FIG. 2.

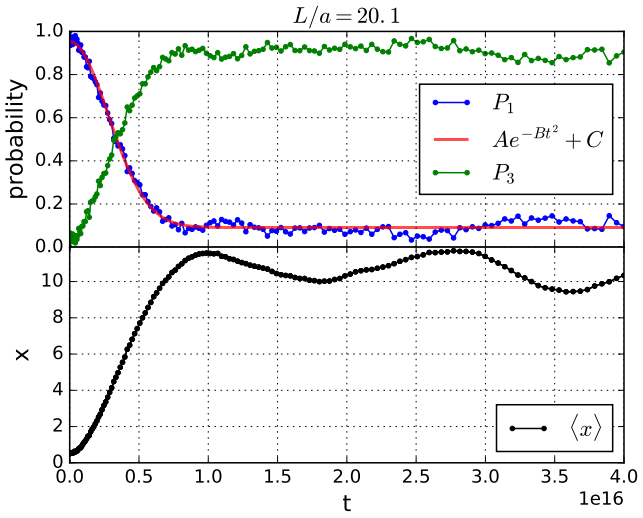

FIG. 2. *Above:* Probability of finding the particle in region 1 (blue) and region 3 (green) versus time. P_1 fits to a gaussian function (red). *Below:* Expected value of x versus time.

In this case fit function has the form $Ae^{-Bt^2} + C$ and the fit parameters are calculated as $A = 0.877504762$, $B = 6.52195870 \times 10^{-32}$, $C = 7.69176722 \times 10^{-2}$. By using these values half life t_0 of the radiation can be calculated as:

$$t_0 = 3.46907575 \times 10^{15} \tag{8}$$

This value is equal to the full width at half maximum of the gaussian function.

CALCULATIONS

Functions $u_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right)$ is choosen as basis for constructing the hamiltonian since they form a complete orthonormal set. Its matrix element can be analytically calculated as:

$$\begin{aligned} H_{mn} &= \int_0^L u_m H u_n dx \\ &= \frac{2V_0}{L} \int_a^{a+w} \sin\left(\frac{m\pi x}{L}\right) \sin\left(\frac{n\pi x}{L}\right) dx \\ &= \frac{2V_0}{L} \begin{cases} 0 & \text{for } m = n = 0 \\ -\frac{L}{2\pi n} \sin\left(\frac{\pi n}{L}x\right) \cos\left(\frac{\pi n}{L}x\right) + \frac{x}{2} \sin^2\left(\frac{\pi n}{L}x\right) + \frac{x}{2} \cos^2\left(\frac{\pi n}{L}x\right) & \text{for } m = n \neq 0 \\ -\frac{Lm \cos\left(\frac{\pi m}{L}x\right)}{\pi m^2 - \pi n^2} \sin\left(\frac{\pi n}{L}x\right) + \frac{Ln \cos\left(\frac{\pi n}{L}x\right)}{\pi m^2 - \pi n^2} \sin\left(\frac{\pi m}{L}x\right) & \text{for } m \neq n \end{cases} \end{aligned}$$

Therefore N dimensional hamiltonian matrix can be constructed in python as:

```
def matrix\_mn(N, a, w, L, V0, x):
    H = np.zeros([N,N])
    equal = lambda m,n,x : -L*sin(pi*n*x/L)*cos(pi*n*x/L)/(2*pi*n) + x*sin(pi*n*x/L)**2/2 + x*cos(pi*n*x↩
        /L)**2/2
    otherwise = lambda m,n,x : -L*m*sin(pi*n*x/L)*cos(pi*m*x/L)/(pi*m**2 - pi*n**2) + L*n*sin(pi*m*x/L)*↩
        cos(pi*n*x/L)/(pi*m**2 - pi*n**2)
    for m in range(N):
        for n in range(N):
            if m == 0 and n == 0:
                H[m,n] = 0
            elif m == n:
                H[m,n] = equal(m,n,a+w) - equal(m,n,a)
            elif m != n:
                H[m,n] = otherwise(m,n,a+w) - otherwise(m,n,a)

    return H*2*V0/L
```

After constructing the hamiltonian, energy eigenvalues `E` and eigenvectors `v` can be calculated in python as given below:

```
E, v = numpy.linalg.eigh(H)
```

Since the basis functions Ψ_n and the eigenvectors `v` of hamiltonian is known, solutions of time independent Schrödinger equation $\phi_n(x)$ becomes:

$$\phi_n(x) = \sum_i^N \boxed{\text{v[i,n]}}\, u_i(x)$$

In python, it is convinient to store all $\phi_n(x)$'s in a matrix called `phi_list`. Where $\phi_n(x)$ = `phi_list[n,:]` and it is constructed as:

`phi_list = numpy.dot(vecs.T,M)`

Where `M` is the matrix that contains the basis vectors such that $u_n(x)$ = `M[n,:]`. c_n coefficients can be easly calculated as shown in Eq.(7)

```
c_list = np.zeros([N,])
for n in range(N):
    c_list[n] = scipy.integrate.simps(psi0*phi_n,↩
        x)
```

Finally $\psi(x,t)$ can be calculated as:

```
psi = numpy.zeros(phi[0,:].shape)
for i in range(N):
    psi += c_list[n] * phi_list[n,:] * numpy.exp↩
        (-1j*E[n]*t)
```

The final code is:

```
spy# -*- coding: utf-8 -*-
"""
Created on Sun Nov 19 14:08:59 2017

@author: umurcankaya
"""
import numpy as np
from numpy import pi, sin, cos, sqrt, exp
import matplotlib.pyplot as plt
from scipy.integrate import simps
from matplotlib import cm
from scipy.optimize import curve_fit

def matrix_mn(N, a, W, L, V0, x):

    M = np.zeros([N,len(x)])
    H = np.zeros([N,N])

    equal = lambda m,n,x : -L*sin(pi*n*x/L)*cos(pi*n*x/L)/(2*pi*n) + x*sin(pi*n*x/L)**2/2 + x*cos(pi*n*x↩
        /L)**2/2
    otherwise = lambda m,n,x : -L*m*sin(pi*n*x/L)*cos(pi*m*x/L)/(pi*m**2 - pi*n**2) + L*n*sin(pi*m*x/L)*↩
        cos(pi*n*x/L)/(pi*m**2 - pi*n**2)

    for m in range(N):
        M[m,:] = sin((m)*pi*x/L)
        for n in range(N):
            if m == 0 and n == 0:
                H[m,n] = 0
            elif m == n:
                H[m,n] = equal(m,n,a+W) - equal(m,n,a)
            elif m != n:
                H[m,n] = otherwise(m,n,a+W) - otherwise(m,n,a)

    return H*2*V0/L, M*sqrt(2/L)

def c_n(phi_n,psi0,x):
    return simps(psi0*phi_n,x)

def Psi(t,c_list,phi_list,E,x):
    return np.dot((c_list * exp(-1j*E*t)).T,phi_list)

def x_expected(psi,x):
    return simps(x*psi*np.conj(psi),x)

def probability(psi,x,where,a,L,W):
    if where == a:
        idx = x<=a
    elif where == L:
        idx = x>=a+W
    x = np.array(x[idx])
    psi = np.array(psi[idx])
    return simps(psi*np.conj(psi),x)

# ------------- main ------------- #

a = 1
W = 0.1*a
L = 20*a+W
V0 = 1

N = 97*2
x = np.linspace(0,L,1e5)
```

```
V = x*0 + V0
V[x < a] = 0
V[x > a+W] = 0

psi0 = sqrt(2/a) * sin(pi*x/a)
psi0[x>a] = 0

H, M = matrix_mn(N,a,W,L,V0,x)
E, vecs = np.linalg.eigh(H)

phi_list = np.dot(vecs.T,M)

c_list = np.zeros([N,])
for n in range(N):
    c_list[n] = c_n(phi_list[n,:],psi0,x)

time = np.zeros((150,1))
time[:,0] = np.linspace(0,20,150)**2
time *= 1e14
expected_all = np.zeros([len(time),1])
prob_a = np.zeros([len(time),1])
prob_L = np.zeros([len(time),1])

for i in range(len(time)):

    t = time[i]

    psi = Psi(t,c_list,phi_list,E,x)
    expected_all[i] = np.real(x_expected(psi,x))
    prob_a[i] = np.real(probability(psi,x,a,a,L,W))
    prob_L[i] = np.real(probability(psi,x,L,a,L,W))
    """
    fig=plt.figure(1)
    plt.plot(x,V,color="black",linewidth=2, label="$V$")
    plt.plot(x*0,x,color="black",linewidth=2)
    plt.annotate(("t = %.3e"%t),xy=(L/2-3*a,1))
    plt.annotate(("<x> = %.3f"%expected_all[i]),xy=(L/2-3*a,0.8))
    plt.xlim([-0.02,L/2])
    plt.ylim([0,1.5])
    plt.grid()
    plt.plot(x,psi*np.conj(psi),label="$|\psi(t,x)|^2$")
    plt.legend(loc='upper right')
    plt.xlabel("x")
    plt.show()
    print()
    """
prob_a = np.real(prob_a)
prob_L = np.real(prob_L)

idx = time > 1.5e16
t_last = time[idx]
expected_all_last = expected_all[idx]
prob_a_last = prob_a[idx]
prob_L_last = prob_L[idx]
ave_all = sum(expected_all_last)/len(expected_all_last)
ave_a = sum(prob_a_last)/len(prob_a_last)
ave_L = sum(prob_L_last)/len(prob_L_last)

print("a/(L-(a+w)) = ", a/(L-(a+W)))
print("ave_a/ave_L = ", ave_a/ave_L)

def gauss(x,A,B,C):
    return A * np.exp(-B*x**2) + C

popt, pcov = curve_fit(gauss, time.ravel(), prob_a.ravel(), p0=[0.878,6.522e-32,0.077])
fit_gauss = gauss(time, *popt)

half_life = np.sqrt( -np.log((0.5*fit_gauss[0]-popt[2])/popt[0])/popt[1] )

print("half-life = ", half_life)

f, (ax1, ax2) = plt.subplots(2, sharex=True)
ax1.set_title("$L/a = %.1f$"%(L/a),fontsize=15)
ax1.plot(time,np.array(prob_a), ".-", label="$P_1$")
ax1.plot(time,fit_gauss,"r-",lw=2, label="$Ae^{-Bt^2}+C$", alpha=0.7)
#ax1.plot(time,fit,"--grey",alpha=0.7)
ax1.plot(time,np.array(prob_L), ".-", label="$P_3$")
#plt.title("probability of finding the particle between")
ax1.legend(loc="best",fontsize=15)
ax1.grid()
ax1.set_ylabel("probability",fontsize=15)
ax2.plot(time,np.array(expected_all),"k.-",label=r"$\langle x \rangle$")
ax2.set_xlabel("t",fontsize=15)
```

```
ax2.set_ylabel("x",fontsize=15)
ax2.legend(loc="lower right",fontsize=15)
ax2.grid()
ax2.plot(time,np.array(expected_all),"k.-")
ax2.set_xticks([0,2,4,6,8,10],["0","2","4","6","8","10"])
ax2.set_ylim([0,11.9])
f.subplots_adjust(hspace=0)
plt.setp([a.get_xticklabels() for a in f.axes[:-1]], visible=False)
plt.show()
```