Assignment 7 - Harmonic Oscillator - III(Matching)

SGTB Khalsa College, University of Delhi

Preetpal Singh(2020PHY1140)(20068567043)

Unique Paper Code: 32221501

Paper Title: Quantum Mechanics and Applications

Submitted on: August 24, 2022

B.Sc(H) Physics Sem V

Submitted to: Dr. Mamta

Preetpal Singh
Date=> Sep 09, 2022

Roll No.=> 2020PHY1140

Assignment- 7 (Harmonic Oscillator)(Matching)

Date ..../..../.........

(a) Explain why correct asymptotic solutions are not obtained when you integrate the schrodinger from x_{min} to x_{max}.

Ans Since, the solution of schrodinger wave equation of Harmonic oscillator is of the form

$$\psi = Ae^{ikx} + Be^{-ikx}$$

So, when x approaches +∞ or -∞ then ψ approaches ∞, which is physically not feasible. We can ignore this dominance of approaching ∞ theoretically by removing Ae^{ikx} or Be^{-ikx} when required.

However, when we solve for same Equation numerically, It's not possible to do so. Therefore, the wave function approaches ~~+∞~~ infinity while taking bigger values of x.

(b) Discuss the how the instability discussed above maybe controlled by matching the solution at a point in the allowed region.

Ans. To remove the instability, we divide the integration region (x_{min} to x_{max}) into two parts by classical turning point (the point at which solution becomes instable).
Therefore, we perform integration from 0 to x_{cl} with forward integration.
→ Perform integration from x_{max} to x_{cl} with backward integration.

Then, we have to do matching at classical turning point to ensure that we get same value at classical turning point with forward and backward integration.
To do matching
→ match the value of integration at x_{cl} by using scaling factor
→ match the derivative at x_{cl}

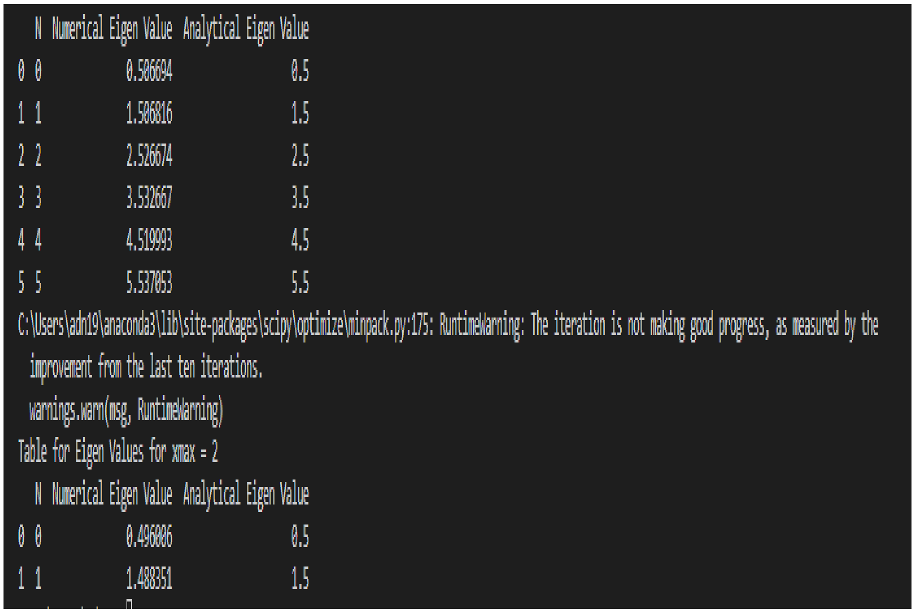
Spiral

Programming

```
import numpy as np
import matplotlib.pyplot as plt
import math
import scipy.integrate as integrate
from scipy import optimize,stats
from scipy.optimize import fsolve
import pandas as pd
from scipy.integrate import solve_ivp

def numerov(x,E_min, E_max):
    c_i =[];u=[]
    h = x[1]-x[0]
    Alpha = 2*(((-x**2)/2)+(E_min+E_max)/2)
    ddx_12 = (h**2)/12
    for i in range(0,len(x)):
        c_i_ = 1 + np.multiply(ddx_12,Alpha[i])
        c_i.append(c_i_)
    if (E_max-0.5)%2==0:
        u_0 = 1
        u_1=(6-5*c_i[0])/c_i[1]
    else:
        u_0 = 0
        u_1=h
    u.append(u_0);u.append(u_1)
    for i in range(2,len(x)):
        u_ = (1/c_i[i])*(((12-10*c_i[i-1])*u[i-1])-c_i[i-2]*u[i-2])
        u.append(u_)
    return u,  x
def numerov_(x,E_min, E_max):
    c_i =[];u=[]
    h = -(x[1]-x[0])
    Alpha = 2*(((-x**2)/2)+(E_min+E_max)/2)
    ddx_12 = (h**2)/12

    for i in range(0,len(x)):
        c_i_ = 1 + np.multiply(ddx_12,Alpha[i])
        c_i.append(c_i_)
    u_0 = 0
    u_1=h
    u.append(u_0);u.append(u_1)
    for i in range(2,len(x)):
        u_ = (1/c_i[i])*(((12-10*c_i[i-1])*u[i-1])-c_i[i-2]*u[i-2])
        u.append(u_)
    return u,  x

def e_range(u,n_node,E_min,E_max) :
    I = []
    E = (E_min+E_max)/2
    for i in range(len(u)):
        if (u[i-1]*u[i]) < 0:
           I.append(i)
    N_node = len(I)
    if N_node > int(n_node):
       E_max = E
    else:
       E_min = E
    return len(I),E_min,E_max

def E(n_node,E_min,E_max,tol):
    for i in range(1000):
        p=numerov(x,E_min, E_max  )
        p1=numerov_(x1,E_min, E_max)
        U_, U_back_ = matching(p,p1,x,x1)
        U = U_[:-1]
        U_back = U_back_[::-1]
        for i in U_back:
            U.append(i)
        u_norm=U/np.sqrt(integrate.simps(np.power(U,2),np.linspace(xi_1,xf_2,len(U))))
        I ,E_min_new,E_max_new = e_range(u_norm,n_node,E_min,E_max)
        if abs(E_max_new - E_min_new)<tol:
            break
        else:
           E_min = E_min_new
           E_max = E_max_new
    return E_min_new,E_max_new,U

def combine(list1, list2):
    list1_ = np.delete(list1, len(list1)-1)
    result_list = []
    result_list = list(list1_)
    for item in list2:
        result_list.append(item)
    return result_list


def parity(n_node,E_min,E_max,tol):
    p = E(n_node,E_min,E_max,tol)
    t = p[2][::-1]
    # t_1 = p[2][::-1]
    array = [];array_1=[]

    if n_node%2 != 0:
        # array_1 = combine((np.multiply(-1,t_1)),p[2])
        array = combine((np.multiply(-1,t)),p[2])
        array_1 = combine((np.multiply(-1,t)),p[2])
    elif n_node%2 == 0:
        # array_1 = combine(t_1, p[2])
        array = combine(t, p[2])
    return array, array
def cl_trn_pts(xf_1,N,n_node):
    x = np.linspace(0,xf_1,N+1)
    index=0;xf_=0
    for i in x:
        index+=1
        if round(i,2) == round(np.sqrt(2*n_node+1),2):
            xf_ = i

            break
        x_for,x_back=x[:index+1],x[index:]
    return xf_,index,x_for,x_back[::-1]

def matching(p,p1,x,x1):
    rescale_fac = p[0][-1]/p1[0][-1]
    p2=[]
    for i in p1[0]:
        p2.append(i*rescale_fac)
    return p[0], p2

def E_range(p1,p2,x,x1,E_min,E_max) :
    E = (E_min+E_max)/2
    if phi(E) > 0:
        E_max = E
    else:
        E_min = E
    return E_min,E_max

def phi(E):
    array, array1 = parity(n_node,E,E_max,tol)
    u_norm=array/np.sqrt(integrate.simps(np.power(array,2),np.linspace(-xf_2,xf_2,len(array))))
    c_i=[]
```

```python
    h = x[1]-x[0]
    Alpha = 2*(((-x**2)/2)+E)
    ddx_12 = (h**2)/12
    for i in range(0,len(x)):
        c_i_ = 1 + np.multiply(ddx_12,Alpha[i])
        c_i.append(c_i_)
    t= int(len(u_norm)/2)
    p=len(x)
    G = (1/h)*(u_norm[t+p+1]+u_norm[t+p-1]-((12*c_i[-1])-10)*u_norm[t+p])
    return abs(G)

def match_der(p1,p2,x,x1,E_min,E_max,tol):
    E_min_new,E_max_new,U=E(n_node,E_min,E_max,tol)
    root = fsolve(phi,E_min_new,xtol=1e-10)
    return root[0]


''' ------------------------------------------------------------------------------------------ '''
num_eig_val=[];n=[];anal_eig_val=[]
for i in range(0,6):
    n_node=i
    xf_1_=10;N=1000;E_max=n_node+0.5;E_min=0;tol=0.4
    if n_node %2 ==0:
        u_0 = 1
        u_prime=0
    else:
        u_0 = 0
        u_prime = 1

    xf_,index,x,x1 = cl_trn_pts(xf_1_,N,n_node)
    xi_1=0;xf_1=xf_;xi_2=xf_1_;xf_2=xf_
    p=numerov(x,E_min, E_max  )
    p1=numerov_(x1,E_min, E_max)
    p1,p2=matching(p,p1,x,x1)
    num_eig_val_ = match_der(p1,p2,x,x1,E_min,E_max,tol)
    num_eig_val.append(num_eig_val_)
    n.append(i)
    anal_eig_val.append(i+0.5)
print("Table for Eigen Values for xmax = 10")
data = {
    "N":n,
    "Numerical Eigen Value":num_eig_val,
    "Analytical Eigen Value":anal_eig_val,
}
print(pd.DataFrame(data))

'''------------------------------------------------------------'''
num_eig_val=[];n=[];anal_eig_val=[]
for i in range(0,2):
    n_node=i
    xf_1_=2;N=1000;E_max=n_node+0.5;E_min=0;tol=0.4
    if n_node %2 ==0:
        u_0 = 1
        u_prime=0
    else:
        u_0 = 0
        u_prime = 1

    xf_,index,x,x1 = cl_trn_pts(xf_1_,N,n_node)
    xi_1=0;xf_1=xf_;xi_2=xf_1_;xf_2=xf_
    p=numerov(x,E_min, E_max  )
    p1=numerov_(x1,E_min, E_max)
    p1,p2=matching(p,p1,x,x1)
    num_eig_val_ = match_der(p1,p2,x,x1,E_min,E_max,tol)
    num_eig_val.append(num_eig_val_)
    n.append(i)
    anal_eig_val.append(i+0.5)
print("Table for Eigen Values for xmax = 2")
data = {
    "N":n,
    "Numerical Eigen Value":num_eig_val,
    "Analytical Eigen Value":anal_eig_val,
}
print(pd.DataFrame(data))
```

Result and Discussion



```
   N  Numerical Eigen Value  Analytical Eigen Value
0  0               0.506694                     0.5
1  1               1.506816                     1.5
2  2               2.526674                     2.5
3  3               3.532667                     3.5
4  4               4.519993                     4.5
5  5               5.537053                     5.5
C:\Users\adn19\anaconda3\lib\site-packages\scipy\optimize\minpack.py:175: RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
Table for Eigen Values for xmax = 2
   N  Numerical Eigen Value  Analytical Eigen Value
0  0               0.496006                     0.5
1  1               1.488351                     1.5
```