Assignment 7 - Harmonic Oscillator - III (matching)

SGTB Khalsa College, University of Delhi

Ankur Kumar(2020PHY1113)(20068567010)

Unique Paper Code: 32221501

Paper Title: Quantum Mechanics

Submitted on: September 9, 2022

B.Sc(H) Physics Sem V

Submitted to: Dr. Mamta

Page No. ______
Date | | |

Theory :

(a) The approximate solution is of the form:

$$\psi(x) = \underbrace{Ae^{-x^2/2}}_{\text{Physical solution}} + \underbrace{Be^{x^2/2}}_{\text{unphysical solution}}$$

While solving the equation analytically, we make $B = 0$, hence ignoring the unphysical solution.

But numerically, in the large values of x, the unphysical solution dominates over the physical solution.

The physical solution decays rapidly to zero and the unphysical solution which grows exponentially to $\pm\infty$.

Even if we use a guess value of E to be exactly equal to an energy eigen value, numerical errors will cause the solution to grow into the unphysical one.

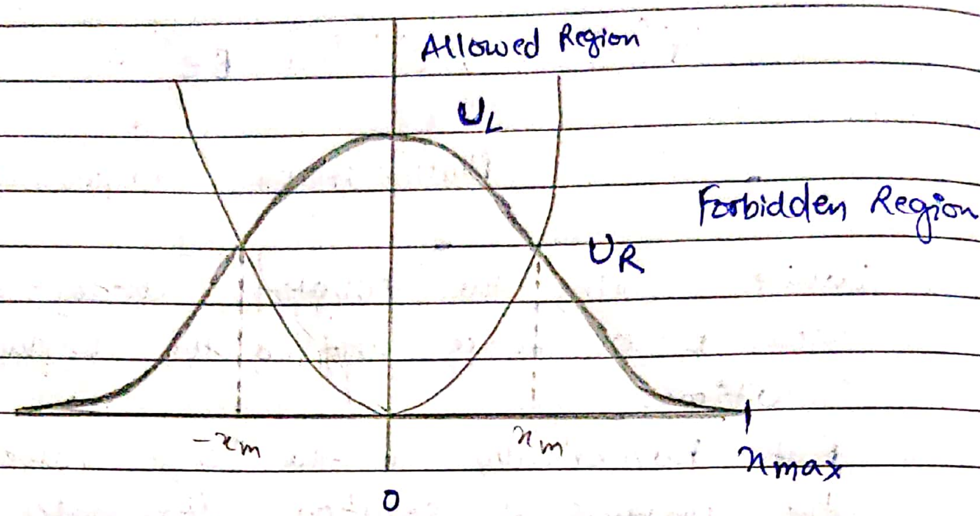
When solving the system numerically, however, numerical error causes the non physical solution,

→ $e^{x^2/2}$ on right boundary and

→ $e^{-x^2/2}$ on the left boundary

to influence the result.

(b) The instability may be controlled by matching the solution at a point in the allowed region.



we can perform two integrations and generate two solutions:

1. Start from $x=0$ and integrate forward $\rightarrow U_L$
2. Start from $x=x_{max}$ and integrate backward $\rightarrow U_R$

we choose a matching point x_m (usually the classical turning point) somewhere in the allowed region and match the solutions at $x=x_m$.

By multiplying one of the soln by a constant we can ensure that,

$$U_L(x_m) = U_R(x_m) = U(x_m).$$

The test for a true match is that

$$\left.\frac{dU_L}{dx}\right|_{x=x_m} = \left.\frac{dU_R}{dx}\right|_{x=x_m}$$

Page No.
Date

we Start with a guess value of ε, and with a simple root finding method, we find that value of ε for which the points at x_m and the derivative at x_m both match.

This produces a continous solution of the eq. which will always goes to 0 as x approaches x_{max} because we set the initial condition as such.

This removes the instability at the large values of x and wave function goes to 0 at x_{max}.

Asymptotic solutions are hence obtained.

Programming

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import simps
import pandas as pd
from scipy.optimize import newton, fsolve


def alpha(x, e):

    return 2*(e - (1/2)*(x**2))

def numerov(f, x_range, u0, e, key):

    h = x_range[1] - x_range[0]
    if key == 1:
        h = -h

    U_i = np.zeros(len(x_range))
    C_i = np.ones(len(x_range)) + np.multiply((h**2)/12, f(x_range, e))

    U_i[0] = u0

    if u0 == 0:
        U_i[1] = u0 + h
    else:
        U_i[1] = ((6 - 5*C_i[0])/C_i[1])*u0

    for i in range(1, len(x_range)-1):
        U_i[i+1] = (1/C_i[i+1])*((12-10*C_i[i])*U_i[i] - C_i[i-1]*U_i[i-1])

    return x_range, U_i, C_i

def turning_points(x_range, e, alpha):

    q = alpha(x_range, e)
    for i in range(1, len(x_range)):

        if q[i-1]*q[i] < 0:
            return x_range[i], i


def e_shooting(n_node, E_min, E_max, x_range, initial_cond):

    N_node = 100

    while N_node != n_node:

        I = []
        E = (E_min+E_max)/2

        u = numerov(alpha, x_range, initial_cond, E, key = 0)[1]


        for i in range(1, len(u)):

            if (u[i-1]*u[i]) < 0:

                I.append(i)

        N_node = len(I)

        if N_node > int((n_node)/2):

            E_max = E

```

```
        elif N_node < int((n_node)/2):

            E_min = E
        else:

            return E, E+0.1



def phi(e):

    turning_pt = turning_points(x_range, e, alpha)

    xL, uL, cL = numerov(alpha, np.linspace(0,turning_pt[0],len(x_range)), 1, e, key = 0)

    xR, uR, cR = numerov(alpha, np.linspace(x_range[-1], turning_pt[0], len(x_range)), 0, e, key = 1)

    uR_new = (uR/uR[-1])*uL[-1]

    return (uL[-2]+uR_new[-2]-((12*cL[-1]-10)*uL[-1]))/(x_range[1]-x_range[0])



def extending_func(x_range, n_node, initial_cond):

    guess1, guess2= e_shooting(n_node, 0, 5, x_range, initial_cond)
    zero = newton(phi, x0 = guess1, x1 = guess2, fprime = None)

    turning_pt = turning_points(x_range, zero, alpha)


    uR = numerov(alpha, np.linspace(0,turning_pt[0],len(x_range)), 1, zero, key = 0)

    uL = numerov(alpha, np.linspace(x_range[-1], turning_pt[0], len(x_range)), 0, zero, key = 1)


    uL_new = (uL[1]/uL[1][-1])*uR[1][-1]

    xLL = np.flip(uL[0])
    uLL = np.flip(uL_new)

    u_final = np.concatenate((uR[1], uLL))
    x_final = np.concatenate((uR[0], xLL))

    if u_final[0] == 0:

        flipped_u = -np.flip(u_final)
        extended_u = np.concatenate((flipped_u[:-1], u_final))

        flipped_x  = -np.flip(x_final)
        extended_x = np.concatenate((flipped_x[:-1], x_final))

    else:
        flipped_u  = np.flip(u_final)
        extended_u = np.concatenate((flipped_u[:-1], u_final))

        flipped_x  = -np.flip(x_final)
        extended_x = np.concatenate((flipped_x[:-1], x_final))

    return extended_x, extended_u, zero


x_range = np.linspace(0,2,100)
```

```python

x1, u1, z1= extending_func(x_range, 0, 1)

n = [1]
e = [0.5]

data = {

  'n': n,
  'Calculated eigen value': z1,
  'Analytical eigen value': e,

}

df = pd.DataFrame(data)
print(df)

plt.scatter(x1, u1, label = 'N=1', s = 5)
plt.title('Wave Function for n = 1')
plt.xlabel('x')
plt.ylabel('u(x)')
plt.grid()
plt.legend()
plt.show()

plt.scatter(x1, u1**2, label = 'N=1', s = 5)
plt.title('Probability Density for n = 1')
plt.xlabel('x')
plt.ylabel('|u(x)^2|')
plt.grid()
plt.legend()
plt.show()
```

Result and Discussion

	n	Calculated eigen value	Analytical eigen value
0	1	0.524034	0.5

Figure 1: Eigen Values

The calculated eigen value comes out to be the above value for the ground state.

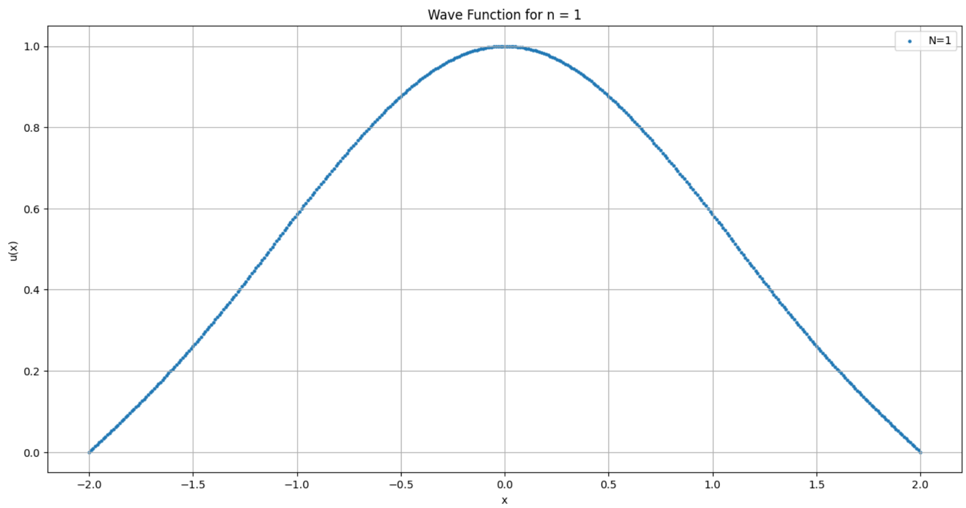


Figure 2: Ground State Wave Function

Wave Function for ground state.

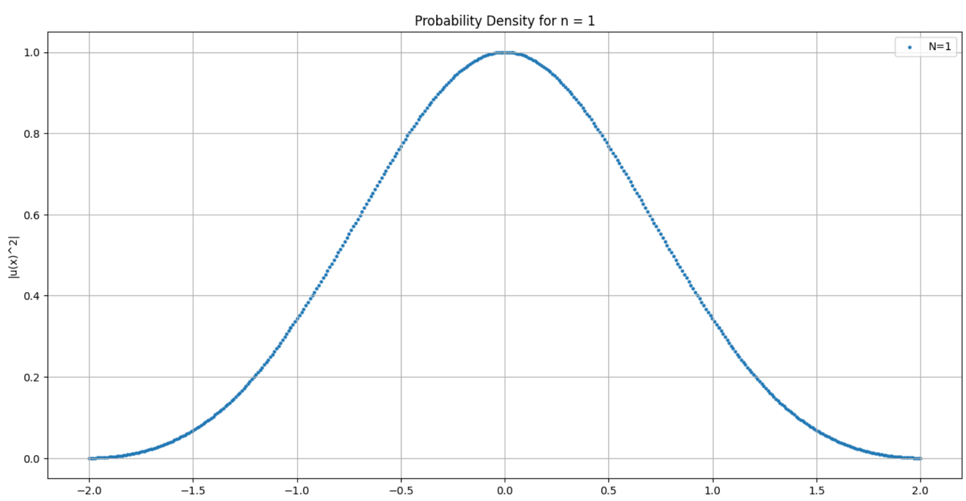


Figure 3: Ground State Probability Density

Probability Density for ground state.