Assignment 6 - Harmonic Oscillator - II

SGTB Khalsa College, University of Delhi

Ankur Kumar(2020PHY1113)(20068567010)

Unique Paper Code: 32221501

Paper Title: Quantum Mechanics

Submitted on: August 24, 2022

B.Sc(H) Physics Sem V

Submitted to: Dr. Mamta

Programming

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import simps
import pandas as pd
from scipy.stats import linregress


def f(x, e):

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


def numerov(x_i, x_f, N, e, n_parity, key1, key2):

    x = np.linspace(x_i, x_f, N)
    h = x[1] - x[0]

    U = np.zeros(len(x))
    C = np.ones(len(x)) + np.multiply((h**2)/12, f(x, e))

    if n_parity % 2 == 0:

        if key2 == True:
            U[0] = -1
        else:
            U[0] = 1

        U[1] = (6 - 5*C[0])/C[1]
    else:
        U[0] = 0
        U[1] = h


    for i in range(1, len(x)-1):

        U[i+1] = (1/C[i+1])*((12-10*C[i])*U[i]-C[i-1]*U[i-1])

    if key1 == 0:

        norm = simps(U**2, x)
        U_vals = U/(np.sqrt(norm))
        return U_vals, x

    else:
        extended_u = []

        if U[0] == 0:

            for i in range(1, len(x)):
                extended_u.append(-1*(U[-i]))

            for i in range(0, len(x)):
                extended_u.append(U[i])

        else:
            for i in range(1, len(x)):
                extended_u.append(1*(U[-i]))

            for i in range(0, len(x)):
                extended_u.append(U[i])

        extended_u = np.array(extended_u)

        extended_x_vals = np.linspace(-x_f, x_f, 2*N-1)

```

```python
        norm = (simps((extended_u)**2, extended_x_vals))
        u_list = (extended_u)/np.sqrt(norm)


        return u_list, extended_x_vals

def e_shooting(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)/2):

       E_max = E
    else:

       E_min = E

    return E_min, E_max


def eigen_vals(xi, xf, N, n_node, E_min, E_max, n):

    tol = 1
    while tol > 10e-6:

        U = numerov(xi, xf, N, (E_max+E_min)/2, n, key1 = 0, key2=None)[0]
        E_min_new, E_max_new = e_shooting(U, n_node, E_min, E_max)

        E_min = E_min_new
        E_max = E_max_new

        tol = abs(E_max - E_min)


    if n/2 == 1: E_min_new = E_min + 4

    return E_min_new, E_max_new


eigen_values = []
n = []
analytic_eigen_values = np.array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5])

for i in range(0, 6):
    E_min,E_max= eigen_vals(0, 5, 100, i, 0, 25/2, i)
    eigen_values.append((E_min+E_max)/2)
    n.append(i)

data = {

  'n': n,
  'Calc Eigen Vals': eigen_values,
  'Analytic Eigen Vals': analytic_eigen_values

}

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

```


plt.scatter(n,eigen_values, color = 'red', label = 'Eigen Values')
plt.plot(n, eigen_values)
plt.title('e(n) as a function of n')
plt.xlabel('n')
plt.ylabel('e(n)')
plt.grid()
plt.legend()
plt.show()

curve1 = linregress(n, eigen_values)

print("Slope for e vs n = ", curve1[0])
print("Intercept e vs n = ", curve1[1])

n_sq = (np.array(n))**2


curve2= linregress(n_sq, eigen_values)

print("Slope e vs n^2 = ", curve2[0])
print("Intercept vs n^2 = ", curve2[1])

fitted_eigen = curve2[0]*n_sq + np.ones(len(n_sq))*curve2[1]

plt.scatter(n_sq,eigen_values, color = 'red', label = 'Eigen Values^2')
plt.plot(n_sq, fitted_eigen, label = 'Fitted Curve')
plt.title('e(n^2) as a function of n^2')
plt.xlabel('n^2')
plt.ylabel('e(n^2)')
plt.grid()
plt.legend()
plt.show()


for i in range(0, 5):

    if i == 2:
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2=True)

    elif i == 3:
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2=None)
        result = result*-1
    else:
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2=None)

    plt.plot(x_vals, result, label = f'n ={i}')

    plt.title('First Five Functions')
    plt.xlabel(r'$\xi$')
    plt.ylabel(r'$U(\xi)$')
    plt.grid()
    plt.legend()
plt.show()


for i in range(0, 5):

    if i == 2:
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2 = True)
        result = result**2
    elif i == 3:
```

```
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2 =
 None)
        result = (result*-1)**2

    else:
        result, x_vals = numerov(0, 5, 100, eigen_values[i], n[i], key1 = 1, key2 =
 None)
        result = result**2

    plt.plot(x_vals, result, label = f'n ={i}')

    plt.title('First Five Probability Densities')
    plt.xlabel(r'$\xi$')
    plt.ylabel(r'$|U^2(\xi)|$')
    plt.grid()
    plt.legend()
plt.show()
```

Result and Discussion

	n	Calc Eigen Vals	Analytic Eigen Vals
0	0	0.500003	0.5
1	1	1.500002	1.5
2	2	2.500003	2.5
3	3	3.500000	3.5
4	4	4.500011	4.5
5	5	5.500093	5.5

Figure 1: Eigen Values

The calculated eigen values are very close to the analytical values. The tolerance was set to 10e-6.

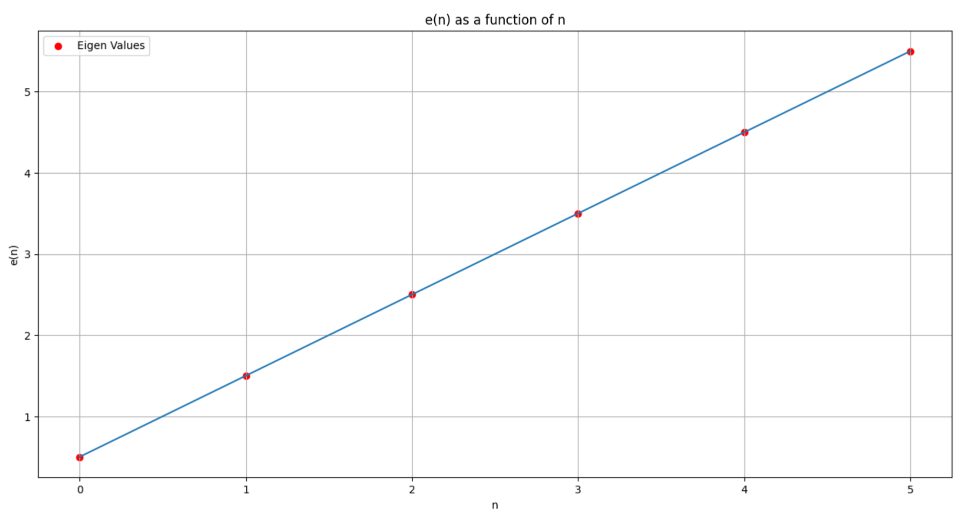

Figure 2: e(n) Vs n

The graph of e(n) as a function of n comes out to be a straight line as expected. The equation is given as: $E = (n + 1/2)$. It is the equation of a straight line.

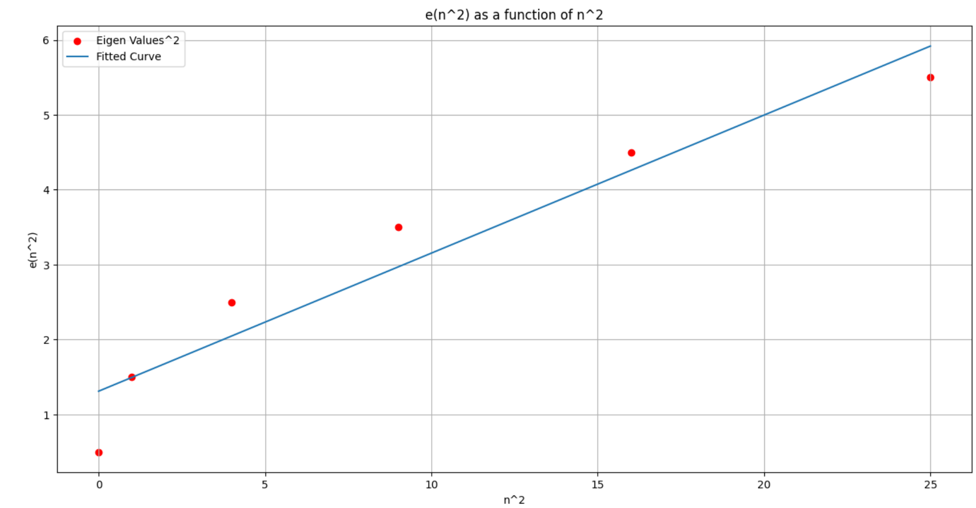

Figure 3: e Vs n^2

The graph between $e(n^2)$ and n^2 is given above.

```
Slope for e vs n =  1.0000136375427247
Intercept e vs n =  0.4999842246373496
Slope e vs n^2 =  0.18427833936724086
Intercept vs n^2 =  1.3108002076277867
```

Figure 4: Slopes and Intercepts

The equation $E = (n + 1/2)$ reveals that the intercept should be 1/2 and the slope should be 1. It can be seen that the calculated values are accurate.

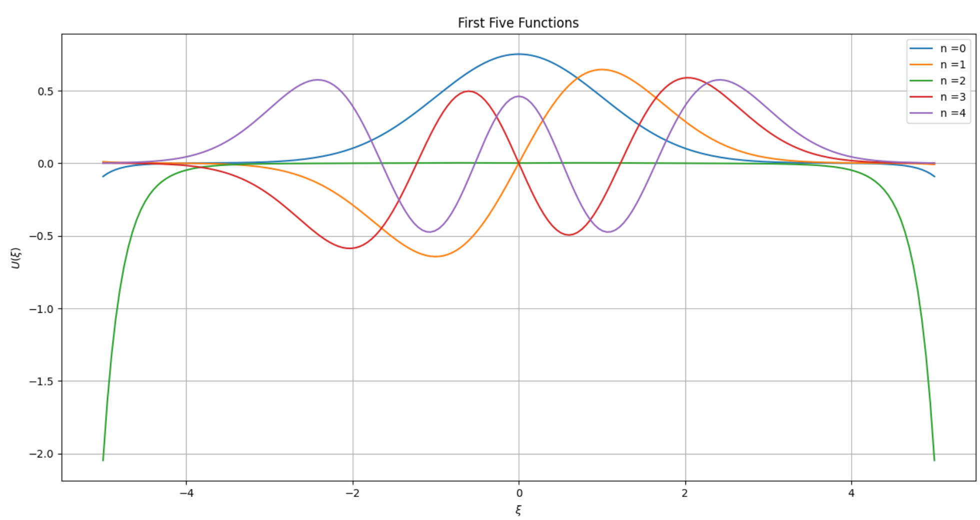

Figure 5: First 5 States

These are the first five states.

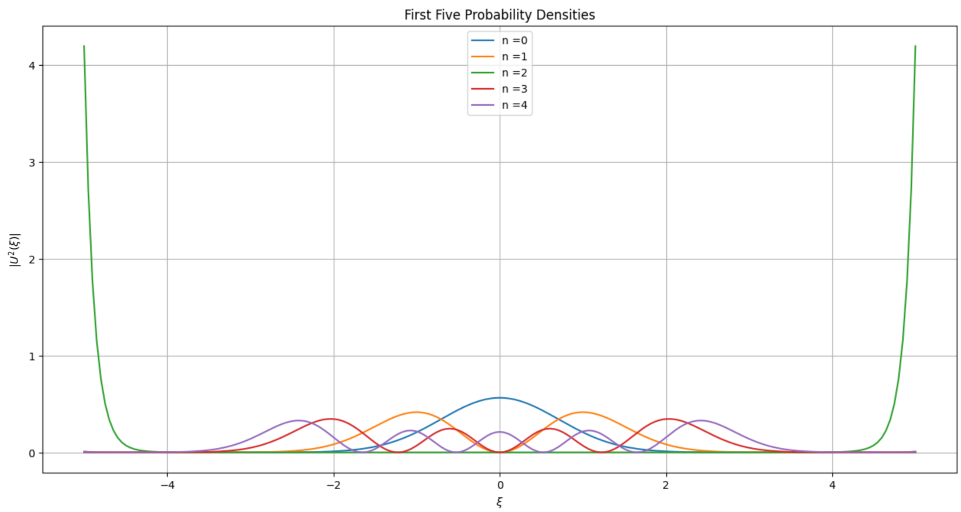

Figure 6: First 5 Probability Densities

These are the first five probability densities.

The graphs for $x_{max} = 10$, 50 did not reveal any important information as both sides extended to infinity.