Quantum Harmonic Oscillator

Jacob Jones

Physics

BOISE STATE UNIVERSITY

I. Introduction

The quantum harmonic oscillator is the quantum-mechanical analog of the classical harmonic oscillator where on the microscopic level vibrations of, for example, atoms in a crystalline structure or diatomic molecules vibrate about their equilibrium positions where the potential is at a local minimum.

II. Theory

As with many quantum-mechanical analogs certain values are quantized; in this case the energy levels for the quantum oscillator are quantized which means that they are only allowed to occupy discrete energy states which in turn limits the positions that the body can occupy.

This program will endeavor to solve the Schrodinger Wave Equation for the allowed energy states.

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi = E\psi$$

For analytical purposes and verification the program will also calculate energy eigenvalues given by the equation:

$$E_n = (n + \frac{1}{2})\omega(\frac{h}{2\pi})$$

Solving the Schrodinger equation for the allowed energy states will enable the program to determine the wave function, ψ(x). When ψ(x) is multiplied by its complex conjugate (the harmonic oscillator is not complex so I am able to simply square ψ(x)) it yields the probability density function, $|\psi(x)^2|$. The probability density function describes the likelihood that the body occupies a position in a region of space. This program plots $|\psi(x)^2|$ for various eigenstates and the final quantum state will reflect The Large-N limit, which is where $n = 10^{26}$ in the equation

$$E_n = (n + \frac{1}{2})\omega(\frac{h}{2\pi})$$

III. Algorithm

I have normalized all constants for the sake of time and convenience:

```
N = 1000
b = 2                    # point outside of HO to check if the function diverges
k = 100                  # spring constant
m = 1                    # mass of object
w = np.sqrt(k/m)         # HO frequency
h = 1                    # normalized Planck constant
L = 1                    # size of the HO
x = np.linspace(-b, b, N) # x-axis
psi = np.zeros([N,2])    # Wave function values and its derivative (psi and phi)
psi_init = np.array([.001,0]) # Wave function initial states
E = 0.0                  # global variable Energy  needed for Sch.Eq, changed in function "WF"
```

The program begins by defining the potential function where $V(x) = \frac{1}{2}kx^2$. The user defined function checks to ensure that x is in the region of the harmonic oscillator, if it is, the user defined function will return $V(x) = \frac{1}{2}kx^2$ if it isn't then then it will return a constant value.

```
def V(x):
    if abs(x)< L:
        return 0.5*k*x**2
    else:
        return 0.5*k*L**2
```

The Schrodinger equation is a second order differential equation in order to solve such an equation in Python I had to transform it into two first order differential equations and specify initial conditions.

```
def S(psi, x):
    dpsi = psi[1]
    dphi = (2.0*m/h**2)*(V(x) - E)*psi[0]
    return np.array([dpsi, dphi])
```

The program is now ready to utilize *odeint()* which I have imported from the *scipy.integrate* package. *Odeint()* will solve the two differential equations and store the values in a two by 'x' array.

```
def WF(ener):
    global psi
    global E
    E = ener
    psi = odeint(S, psi_init, x)
    return psi[-1,0]
```

The program then calculates ψ(x) for energies ranging from zero to a maximum energy, which is partially defined in the known energy function.

```
def KE(en):
    Emax = max(en)
    print ('Allowed Energies for 1D Harmonic Oscillator:')
    n = 0
    while((n+0.5)*h*w < Emax):
        print (((n+0.5)*h*w))
        n+=1
```

The energies that return values for ψ(x) equal to zero are valid states of the harmonic oscillator. The energies where the value of ψ(x) is outside of the harmonic oscillator are stored in a list that is then tested to see if ψ(x) converges within the boundaries. The program tests for convergence, using the Brent method, in order to ensure localization. The Brent method checks for when the sign of ψ(x) changes and then returns the position.

```
def Z(x,y):
    zeroes = []
    s = np.sign(y)
    for i in range(len(y)-1):
        if s[i]+s[i+1] == 0:
            zero = brentq(WF, x[i], x[i+1])
            zeroes.append(zero)
    return zeroes
```

The program is now, for the most part, ready to output data..

IV. Results

Plot A: $|\psi(x)^2|$ for first five energy eigenvalues.

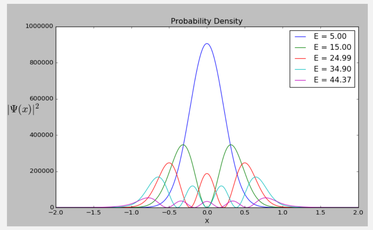
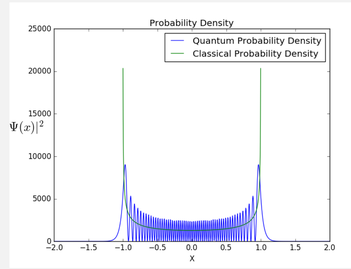


Table A:

```
Calculated Energies for 1D Harmonic Oscillator:
4.9999573488938897
14.998993318527705
24.987220722688636
34.89958659462934
44.37425486787529
Allowed Energies for 1D Harmonic Oscillator:
5.0
15.0
25.0
35.0
45.0
```



Plot B: $|\psi(x)^2|$ for last allowable energy compared to classical probability density function where mass, m, is set equal to 130. Replicating Large-N limit.

V. Analysis

This program is able to accurately calculate the energy eigenvalues for the wave equation and generates a plot of the probability density function which can be valuable information in determining properties of materials.

Plot A shows that as energy increases so do the number of nodes of the wave function and when the energy gets high enough, as in Plot B, the quantum state oscillates so rapidly that only its average value for position can be detected. In order to avoid significant run time my program increases the mass of the object, not n, 130 times in order to replicate the Large-N limit. The program then generates a plot that compares it to the classical probability. This offers key insights into how the quantum realm translates into our macro world.

With regards to computational techniques I found the Brent method to be an interesting and effective tool in calculations and I am confident that I will use it in future programs.