TOPIC:Quantum Harmonic Oscillator

```
GROUP NUMBER:1

MEMBERS:
Deepak Kumar Keshari(21CY40015)
Subhadip Saha(21CY40046)
Kishor Das(21CY40022)
Ritamay Mondal(Motivation)
```

OUTLINE

- **The Harmonic Oscillator Eigenstates**
- **The Classical Harmonic Oscillator**
- **Comparing Classical vs. Quantum Harmonic Results**
- **Plot Eigenvalues and Eigen functions of harmonic oscillator**

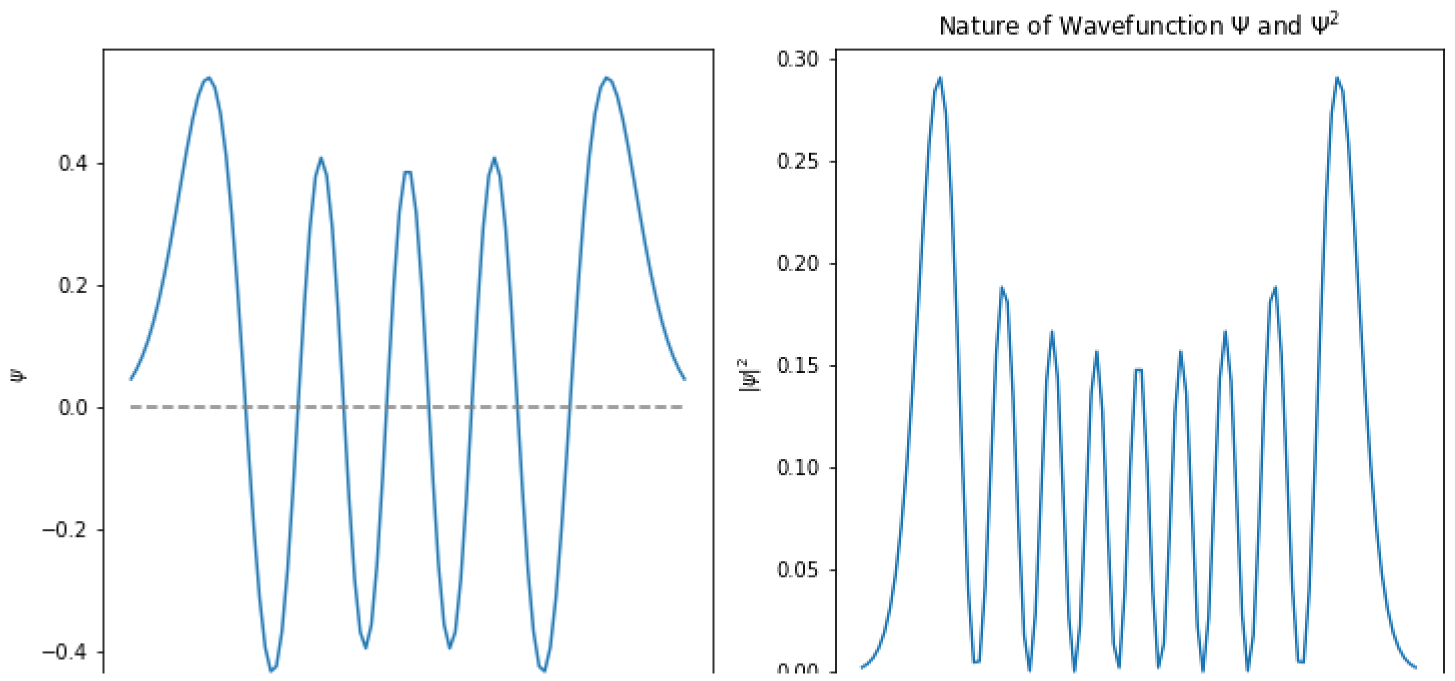
---

*First we tried to plot the wave equation for a particular eigen state w.r.t on
edimensional coordinate.
During solving the schrodinger wave equation for harmonic oscillator we found
that the wave equation consists two parts one is the exponential part and ano
ther is the Hermite polynomial part. It was easy to deal with the exponential
part but to solve the Hemrmite part we developed a analytical pathway so tha
t the hermite polynomial could be solved.In this way by combining both the re
sults we were able to plot the wave functions for various eigen states.*

*First we tried to plot the wave equation for a particular eigen state w.r.t one dimensional coordinate.
During solving the schrodinger wave equation for harmonic oscillator we found that the wave equation consists two parts one is the exponential part and another is the Hermite polynomial part. It was easy to deal with the exponential part but to solve the Hemrmite part we developed a analytical pathway so that the hermite polynomial could be solved. In this way by combining both the results we were able to plot the wave functions for various eigen states.*

The Harmonic Oscillator Eigenstates

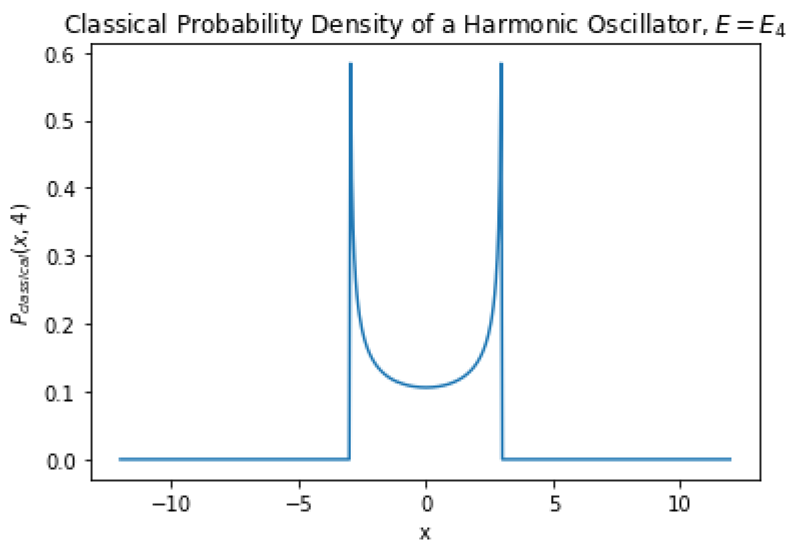
```python
import numpy as np
import matplotlib.pyplot as plt
import math
def fac(n):
    return math.factorial(n)
def evenHn(x,n):              #Compute the even Hermite polynomial
  n1 = int(n/2)
  sum = 0
  for l in range(0,n1+1):

    s_term = ((-1)**(n1-l))/(fac(2*l)*fac(n1-l))
    sum += s_term * (2*x)**(2*l)
  Hn = fac(n)*sum
  return Hn

def oddHn(x,n):              #Compute the odd Hermite polynomial
  n1 = int((n-1)/2)
  sum = 0
  for l in range(0,n1+1):

    s_term = ((-1)**(n1-l))/((fac(2*l))*fac(n1-l))
    sum += s_term * (2*x)**(2*l+1)

  Hn = fac(n)*sum
  return Hn

def gauss(Hn,x):            #The gaussian function returns the product of Hermite polynomial and exponential function of
  f = Hn*math.exp((-1*(x**2))/2)
  return f

# def simpson_rule(y,h):
#   sum1=y[0]+y[-1]
#   for i in range(1,len(y)-1):
#     if i % 2 == 0:
#       ai = 2
#     else:
#      ai = 4
#     sum1 += ai * y[i]
#   sum1 = sum1 * h/3
#   return sum1

def N(n):                   #normalization constant
    '''Normalization constant '''

    return 1./np.sqrt(np.sqrt(np.pi)*2**n*fac(n))


def main(n):                       #Main Function Carrying out the Plot of psi Vs x and also probability density.
  x =np.linspace(-5,5,100)
  h = x[1]-x[0]
  f = []
  f2 = []
  for _ in x:

    if n%2==0:                      #Check for Odd or Even Hermite polynomial
      Hn = evenHn(_,n)
    else:
      Hn = oddHn(_,n)
    f1 = gauss(Hn,_)
    f.append(f1)
    f2.append(f1**2)

  No =  N(n)


  plt.figure(figsize=(12,6))
  plt.subplot(1,2,1)
  plt.plot(x,No*np.array(f))
  plt.plot([-5,5],[0,0], color='gray', linestyle='--')
  plt.xlabel("$x$")
  plt.ylabel("$ψ$")
  plt.subplot(1,2,2)
  plt.plot(x,No**2*np.array(f2))
  plt.xlabel("$x$")
  plt.ylabel("$|ψ|^2$")
  plt.title("Nature of Wavefunction $Ψ$ and $Ψ^2 $")
n = 8
main(n)
```



The Classical Harmonic Oscillator

The classical probability of finding the particle at x is given by $x_{max} = \sqrt(2E_m/\omega^2)$ where xmax is the classical turning point, and $P_{classical}(x)$ is understood to be zero for $|x| > |xmax|$.

```
import matplotlib
import matplotlib.pyplot as plt
import numpy
import numpy.polynomial.hermite as Herm

#Choose simple units
m=1.
w=1.
hbar=1.
#Discretized space
dx = 0.05
x_lim = 12
x = numpy.arange(-x_lim,x_lim,dx)

def classical_P(x,n):
    E = hbar*w*(n+0.5)
    x_max = numpy.sqrt(2*E/(m*w**2))
    classical_prob = numpy.zeros(x.shape[0])
    x_inside = abs(x) < (x_max - 0.025)
    classical_prob[x_inside] = 1./numpy.pi/numpy.sqrt(x_max**2-x[x_inside]*x[x_inside])
    return classical_prob

plt.figure()
plt.plot(x, classical_P(x,4))
plt.xlabel(r"x")
plt.ylabel(r"$P_{classical}(x,4)$")
plt.title(r"Classical Probability Density of a Harmonic Oscillator, $E=E_4$")
plt.show()
```



Comparison of the Classical and Quantum Harmonic Results

We know that quantum physics often makes counter-intuitive predictions that contradict classical physics. But we also know that quantum effects do not often appear in the macroscopic world. For the example of the harmonic oscillator, at which energies do classical and quantum physics most agree, and at which energies do they most strongly diverge? Based on the above graphs, describe some of the differences predicted by quantum and classical physics for the harmonic oscillator.

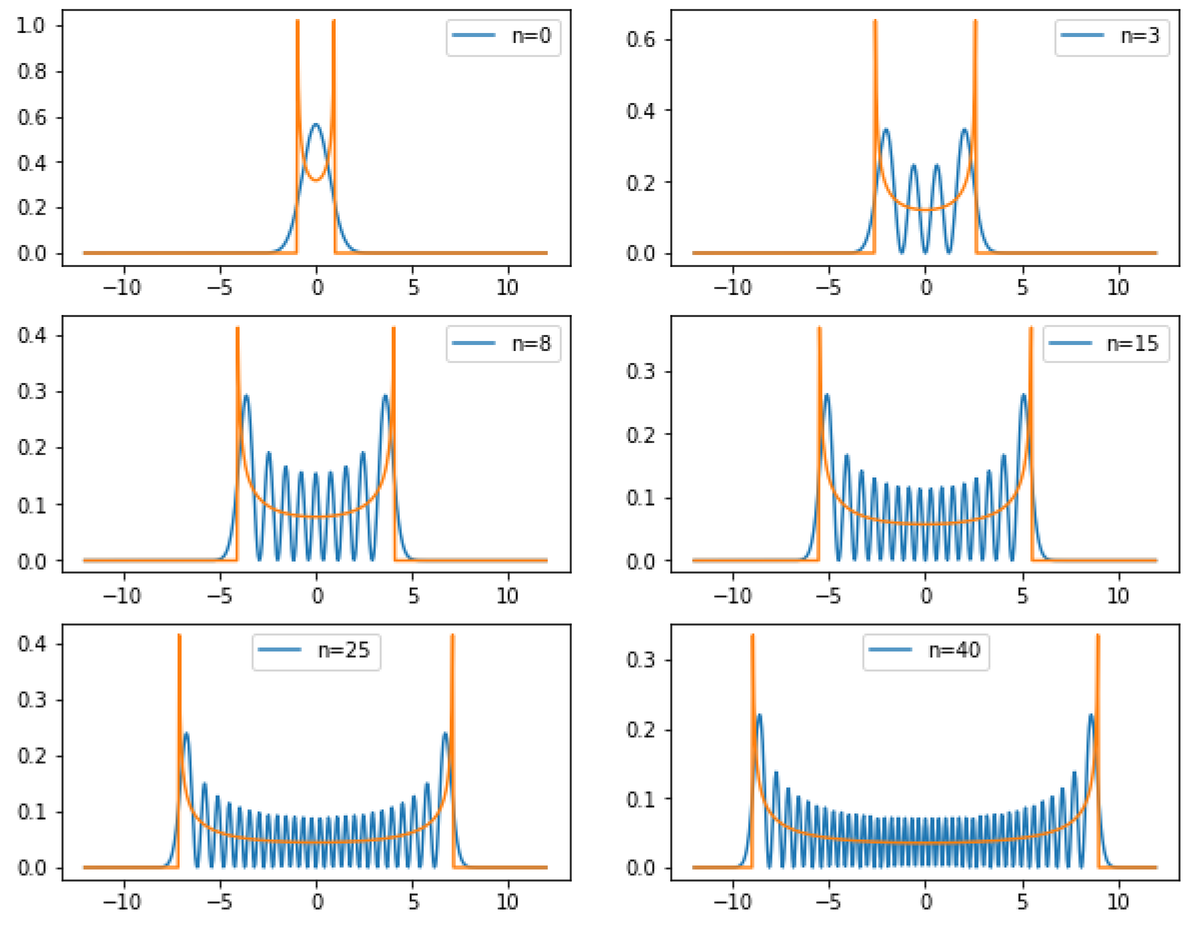
The difference between quantum and classical physics is most distinct for the lower energy levels, where quantum mechanics does not at all obey the classical intuition that particles spend most of their time near the classical turning point. At very high energies though, the general shape of the quantum probability distribution starts to look more like the classical result.

Here are three additional differences between the classical and quantum results.
Classically, there is zero probability for the particle to exist outside of the energetically allowed region (beyond the turning point). However, the quantum results indicate some small but finite probability of measuring the particle in the classically "forbidden" region.

The classical probability distribution says that there is some chance of finding the particle at any point within the classically allowed region. In contrast, the quantum probability distributions possess nodes of zero probability within this region.

This one's not obvious from the graphs, but it's clear enough that a classical harmonic oscillator can possess any energy. In particular, there is no minimum allowable energy, in stark contrast to the quantum harmonic oscillator, whose minimum energy (ground state energy, vacuum energy) is $E_0 = \hbar\omega/2$.

```python
import matplotlib
import matplotlib.pyplot as plt
import numpy
# import numpy.polynomial.hermite as hermite
from scipy.special import hermite
import math

#Choose simple units
m=1.
w=1.
hbar=1.
dx = 0.05
x_lim = 12
x = numpy.arange(-x_lim,x_lim,dx)

def N(n):
    '''Normalization constant '''

    return 1./np.sqrt(np.sqrt(np.pi)*2**n*factorial(n))

def psi(x, n ):
    """Harmonic oscillator wavefunction for level n computed on grid of points x"""

    Hr=hermite(n)

    ψx = N(n)*Hr(x)*np.exp(-0.5*x**2)

    return ψx

def classical_P(x,n):
    E = hbar*w*(n+0.5)
    x_max = numpy.sqrt(2*E/(m*w**2))
    classical_prob = numpy.zeros(x.shape[0])
    x_inside = abs(x) < (x_max - 0.025)
    classical_prob[x_inside] = 1./numpy.pi/numpy.sqrt(x_max**2-x[x_inside]*x[x_inside])
    return classical_prob

plt.figure(figsize=(10, 8))
plt.subplot(3,2,1)
plt.plot(x, numpy.conjugate(psi(x,0))*psi(x,0), label="n=0")
plt.plot(x, classical_P(x,0))
plt.legend()
plt.subplot(3,2,2)
plt.plot(x, numpy.conjugate(psi(x,3))*psi(x,3), label="n=3")
plt.plot(x, classical_P(x,3))
plt.legend()
plt.subplot(3,2,3)
plt.plot(x, numpy.conjugate(psi(x,8))*psi(x,8), label="n=8")
plt.plot(x, classical_P(x,8))
plt.legend()
plt.subplot(3,2,4)
plt.plot(x, numpy.conjugate(psi(x,15))*psi(x,15), label="n=15")
plt.plot(x, classical_P(x,15))
plt.legend()
plt.subplot(3,2,5)
plt.plot(x, numpy.conjugate(psi(x,25))*psi(x,25), label="n=25")
plt.plot(x, classical_P(x,25))
plt.legend()
plt.subplot(3,2,6)
plt.plot(x, numpy.conjugate(psi(x,40))*psi(x,40), label="n=40")
plt.plot(x, classical_P(x,40))
plt.legend()
plt.show()
```



Plot of Probability densities of harmonic oscillator

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy.special import hermite
from math import factorial
def N(n):
    '''Normalization constant '''

    return 1./np.sqrt(np.sqrt(np.pi)*2**n*factorial(n))

def psi(n, x):
    """Harmonic oscillator wavefunction for level n computed on grid of points x"""

    Hr=hermite(n)

    ψx = N(n)*Hr(x)*np.exp(-0.5*x**2)

    return ψx


def E(n): #Eigen values

    '''Eigenvalues in units of h'''

    return (n + 0.5)

def V(x):  #potential energy
    """Potential energy function"""

    return 0.5*x**2

```

```
'''
Installing celluloid module for animated plot of energy and eigen values.
'''

!pip install  celluloid

import numpy as np
import matplotlib.pyplot as plt

from celluloid import Camera
from IPython.display import HTML

fig, bx = plt.subplots(figsize = (8,10))
camera = Camera(fig)
plt.close()
# fig, bx = plt.subplots(figsize=(10,10))
n_mbx = int(input("Enter the desired level: "))+1 #mbximum level as per desired
h = 6.626e-34  #J-s
ω = 4138.5     #cm-1
xmin, xmbx = -np.sqrt(2*E(n_mbx)), np.sqrt(2*E(n_mbx))
x = np.linspace(xmin, xmbx, 1000)
for i in range(n_mbx+1):
   for n in range(i):

          # plot potential V(x)
          bx.plot(x,V(x),color='black')

          # plot psi squared which we shift up by values of energy
          bx.plot(x,psi(n,x)**2 + E(n), lw=2)

          # add lines and labels

          bx.axhline(E(n), color='gray', linestyle='--',lw=1)
          bx.text(xmbx, 1.*E(n), f"n = {n}")


   Energy = h*ω*(2*i+1)
   print(f"Energy at eigen value {E(i)} is: {Energy} Joules")
   camera.snap()


bx.set_xlabel('x')
bx.set_ylabel('$\psi^2_n(x)$')
animation = camera.animate(interval=1000)
HTML(animation.to_html5_video())
```

```
Requirement already satisfied: celluloid in /usr/local/lib/python3.7/dist-packages (0.2.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from celluloid) (3.2.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->celluloid) (0.1
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->celluloc
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->celluloid)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib->celluloid) (1.19
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib->celluloid)
Enter the desired level: 7
Energy at eigen value 0.5 is: 2.7421700999999998e-30 Joules
Energy at eigen value 1.5 is: 8.226510299999999e-30 Joules
Energy at eigen value 2.5 is: 1.37108505e-29 Joules
Energy at eigen value 3.5 is: 1.9195190699999998e-29 Joules
Energy at eigen value 4.5 is: 2.46795309e-29 Joules
Energy at eigen value 5.5 is: 3.01638711e-29 Joules
Energy at eigen value 6.5 is: 3.5648211299999995e-29 Joules
Energy at eigen value 7.5 is: 4.1132551499999996e-29 Joules
Energy at eigen value 8.5 is: 4.66168917e-29 Joules
```

Future prospects and Applications

The harmonic oscillator is a model which has several important applications in both classical and quantum mechanics.

It serves as a prototype in the mathematical treatment of such diverse phenomena as

elasticity, acoustics, AC circuits,
molecular and crystal vibrations,
electromagnetic fields and
optical properties of matter.

References

Thank You

- *https://helentronica.com/2014/12/28/qm-with-python-swing-on-the-quantum-harmonic-oscillator/*
- *https://chem.libretexts.org/Ancillary_Materials/Interactive_Applications*
- *https://opentextbc.ca/universityphysicsv3openstax/chapter/the-quantum-harmonic-oscillator/*
- https://github.com/agarret7/QHO-Visualizer/tree/master/src
- *https://en.wikipedia.org/wiki/Quantum_harmonic_oscillator*