HARMONIC OSCILLATOR

Harmonic Oscillator in Quantum Mechanics:

- Schrodinger Equ. For 1-D harmonic oscillator:

$$\frac{-\hbar^2}{2m}\psi''(x) + \frac{1}{2}kx^2\psi(x) = E\psi(x)$$

$\boldsymbol{where\ k = Force\ const. = m\omega^2 \quad \omega = Angular\ frequency\ of\ oscillation}$

Radial part (l=0) of S-wave Schrodinger equ. is given by

$$\frac{d^2U}{dr^2} = \frac{2m}{\hbar^2}[V(r) - E]U(r)$$

$$\boldsymbol{Here, V(r) = \frac{1}{2}kx^2 = \frac{m\omega^2x^2}{2}}$$

Python Code To Solve SE :

```
import numpy as np
import matplotlib.pyplot as plt

h=1
m=1
e=3.795
w=1
r_min=0.01
r_max=10
n=1000
r=np.linspace(r_min,r_max,n)
d=r[2]-r[1]
V=np.zeros((n,n))
for i in range(n):
    V[i,i]=(m*w**2/2)*r[i]*r[i]
```

Python Code To Solve SE :

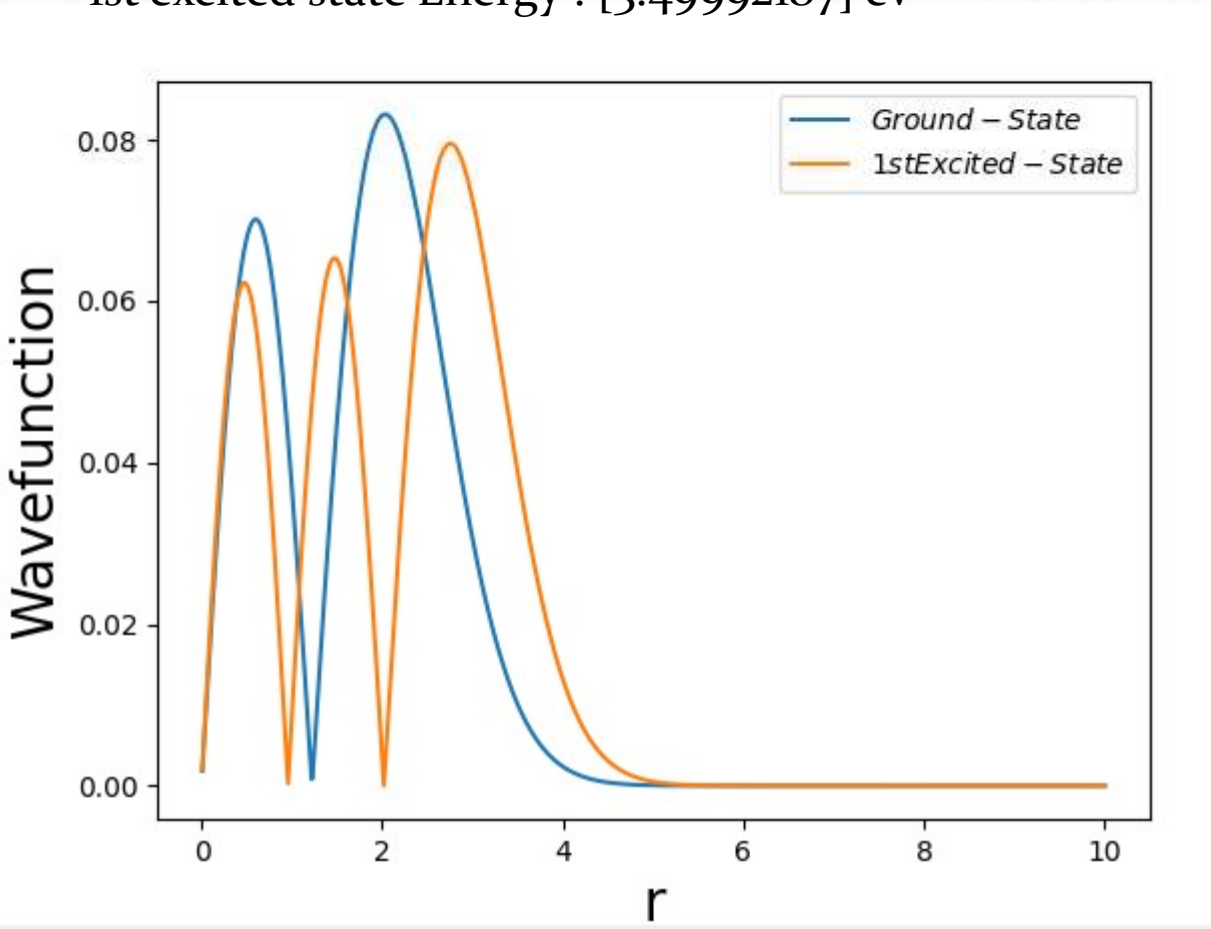
```
K=np.eye(n)*(-2)
for i in range(n-1):
    K[i,i+1]=1
    K[i+1,i]=1
H=((-h**2)/(2*m*d**2))*K+V
E,U=np.linalg.eigh(H)
print ("Energy eigen values are:\n",E)
print ("\nGround state Energy :",E[0:1000:1000],"ev")
print ("\n1st excited state Energy :",E[1:1000:999],"ev")

plt.xlabel('r',size=20)
plt.ylabel('Wavefunction',size=20)
plt.plot(r,abs(U[:,1]))
plt.plot(r,abs(U[:,2]))
plt.legend(['$Ground-State$','$1st Excited-State$'])
plt.show()
```

OUTPUT:

Ground state Energy : [1.49998437] ev

1st excited state Energy : [3.49992187] ev



Comparison of Classical & Quantum Result:

- Classically Equ of motion of a Harmonic Oscillator :

$$\frac{d^2x}{dt^2} + \omega^2 x = 0; \qquad where\ \omega^2 = \frac{k(Force\ Const.)}{m}$$

Classical probability :

$$\rho = \frac{1}{\pi}\sqrt{\left(\frac{k}{2E - kx^2}\right)}$$

Normalized Wave Function:

$$\psi_n(x) = \left(\frac{\sqrt{\alpha}}{2^n n!\sqrt{\pi}}\right)^{1/2} H_n(\sqrt{\alpha}x)e^{\alpha x^2/2}$$

Comparison of Classical & Quantum Result using Computation:

```
import matplotlib
import matplotlib.pyplot as plt
import numpy
import numpy.polynomial.hermite as Herm
Import math
#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)
```

Comparison of Classical & Quantum Result using Computation:

```
def hermite(x, n):
    xi = numpy.sqrt(m*w/hbar)*x
    herm_coeffs = numpy.zeros(n+1)
    herm_coeffs[n] = 1
    return Herm.hermval(xi, herm_coeffs)


def stationary_state(x,n):
    xi = numpy.sqrt(m*w/hbar)*x
    prefactor = 1./math.sqrt(2.**n * math.factorial(n)) *
    (m*w/(numpy.pi*hbar))**(0.25)
    psi = prefactor * numpy.exp(- xi**2 / 2) * hermite(x,n)
    return psi
```

Comparison of Classical & Quantum Result using Computation:

```
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
```

Comparison of Classical & Quantum Result using Computation:

```
plt.figure(figsize=(10, 8))
plt.subplot(3,2,1)
plt.plot(x,
   numpy.conjugate(stationary_state(x,0))*stationary_state(x,0),
   label="n=0")
plt.plot(x, classical_P(x,0))
plt.legend()
plt.subplot(3,2,2)
plt.plot(x,
   numpy.conjugate(stationary_state(x,3))*stationary_state(x,3),
   label="n=3")
plt.plot(x, classical_P(x,3))
plt.legend()
```

Comparison of Classical & Quantum Result using Computation:

```
plt.subplot(3,2,3)
plt.plot(x,
   numpy.conjugate(stationary_state(x,8))*stationary_state(x,8),
   label="n=8")
plt.plot(x, classical_P(x,8))
plt.legend()
plt.subplot(3,2,4)
plt.plot(x,
   numpy.conjugate(stationary_state(x,15))*stationary_state(x,15),
   label="n=15")
plt.plot(x, classical_P(x,15))
plt.legend()
```

Comparison of Classical & Quantum Result using Computation:

```
plt.subplot(3,2,5)
plt.plot(x,
    numpy.conjugate(stationary_state(x,25))*stationary_state(x,25),
    label="n=25")
plt.plot(x, classical_P(x,25))
plt.legend()
plt.subplot(3,2,6)
plt.plot(x,
    numpy.conjugate(stationary_state(x,40))*stationary_state(x,40),
    label="n=40")
plt.plot(x, classical_P(x,40))
plt.legend()
plt.show()
```

OUTPUT: