- CU ROLL NO. : 223223-21-0046
- CU REGISTRATION NO. : 223-1111-0325-22
- CC10 PRACTICAL FILE - QUANTUM MECHANICS

Contents

1 Particle in a Box -I

A particle of mass m is in a one-dimensional finite square well potential

$$V(x) = \begin{cases} V_0, & x < 0 \\ 0, & 0 \leq x \leq a \\ V_0, & x > a \end{cases}$$

Find the energy of the *first excited state* by graphically solving the given transcendental equation that appears as the eigenvalue condition and plot the corresponding normalized wave function numerically by solving the time independent Schrödinger equation. [Consider $\frac{m}{\hbar^2} = 1$, $V_0 = 50$, and $a = 1$]. Hence, find the probability density.

$$\sqrt{u_0^2 - \nu^2} = -\nu \cot(\nu), \quad \text{where} \quad u_0 = \sqrt{\frac{ma^2 V_0}{2\hbar^2}}, \quad \nu = \frac{a\sqrt{2mE}}{\hbar}$$

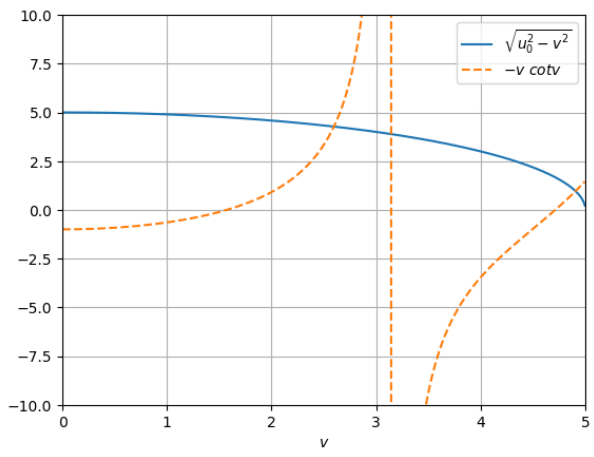
(Symbols have usual meanings).

1.1 Code

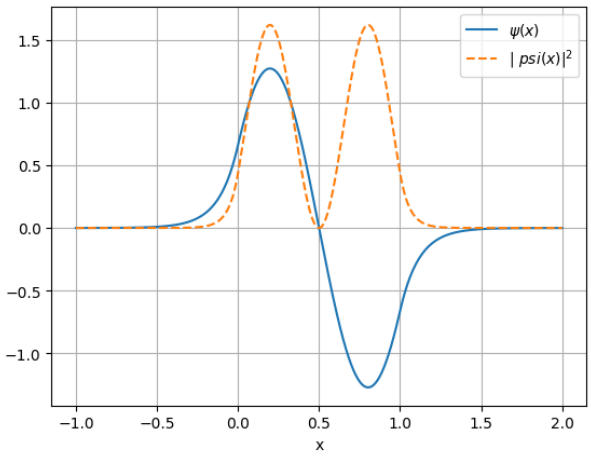
```

import numpy as np
import matplotlib . pyplot as plt
from scipy . optimize import newton
from scipy . integrate import odeint , simps

# Transendental Functions
f1= lambda v: np. sqrt (( u0)**2 -v **2)
f2= lambda v: -v/np.tan (v)
# Parameters
a,V0 ,m_by_hbar2 = 1.0 ,50.0 ,1.0
u0 = np. sqrt ( m_by_hbar2 *a **2* V0 /2)

# Scaling
v = np. linspace (1e-6,u0 ,1000 , endpoint = False )

# Plotting
plt . plot (v,f1(v),ls="-",label ="$\sqrt{u_0 ^2-v^2}$")
plt . plot (v,f2(v),ls="--",label ="$-v\ cot{v}$")
plt . xlim ([0 , u0 ])
plt . ylim ([ -10 ,10])
plt.grid ()
plt.xlabel ("$v$ ")
plt.legend ()
plt.savefig (" Transendental_2 .png")
plt.show ()
Guess = eval ( input (" Enter a guess root from the graph = "))
# Transendental solution function
f = lambda v: np. sqrt (u0 **2 -v **2) +v/np.tan(v)

root = newton (f, Guess )

E =(2* root /a) **2/(2* m_by_hbar2 )
print ("Eigen - energy = ",E)

```

```
 # Potential
V= lambda x: V0 if (x <0 or x>a) else 0.0
# Schrodinger solution
def SE(ini ,x,E):
  psi,psidot = ini
  dpsi2_dx2 =(2.0* m_by_hbar2 )*(V(x)-E)*psi
  return [psidot,dpsi2_dx2]

b=a
x=np.linspace(-b,a+b ,1000)
psi = odeint (SE ,[0 ,1] ,x, args =(E ,))[: ,0]

# Normalize
psi =psi/np. sqrt ( simps (psi **2 ,x))
prob =psi **2
plt.plot (x,psi ,'-',label ="$\psi(x)$")
plt.plot (x,prob ,'--',label ="$|\ psi(x) |^2 $")
plt.grid ()
plt.xlabel ("x")
plt.legend (loc='best')
plt.savefig (" Transdendental .png")
plt.show ()



#output
Enter a guess root from the graph = 2.6
Eigen - energy =  13.475722739242373
```

1.2 Output



(a) Transcendental Equation



(b) Wave function and Probability density

2 Particle in a Box -II

A particle of mass m is in potential

$$V(x) = \begin{cases} 0, & x < 0 \\ -V_0, & 0 \leq x \leq a \\ 0, & x > a \end{cases}$$

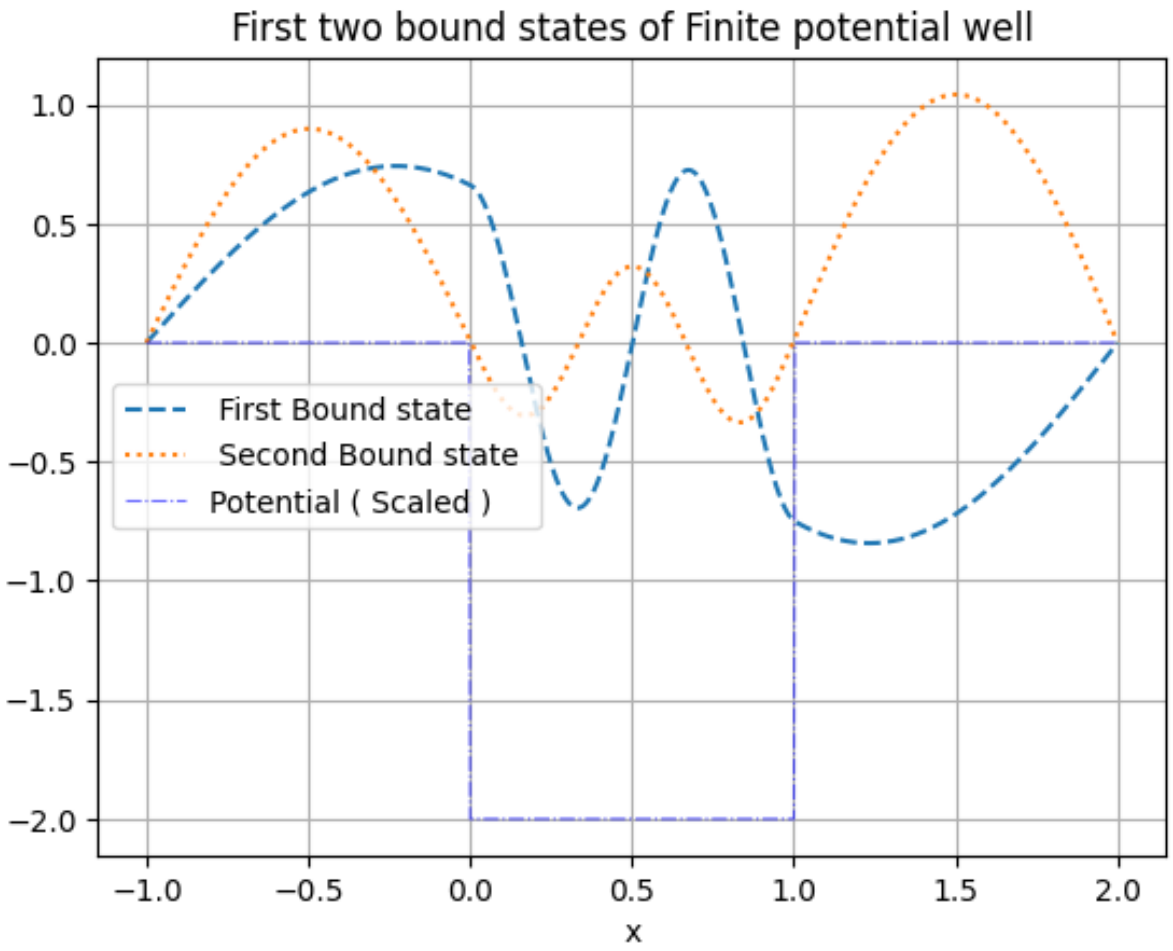
Solve the time independent Schrödinger equation numerically to find first two bound states. Use Shooting algorithm to find Eigenenergies and Euler or Numerov algorithm for solving ODE and find the normalized wave function ψ

2.1 Code

```

from scipy import optimize as opt
import numpy as np
import matplotlib . pyplot as plt
from scipy . integrate import simps

# Parameters
a,V0 ,m_by_hbar2 = 1 ,40 ,1
# Euler method
def Solver (f,z0 ,x,E,dx):
 zs =[ z0]
 z=z0
 for i in range (len(x) -1):
  z=z+dx*np. array (f(z,x[i],E))
  zs. append (z)
 return np. array (zs)

# Schrodinger Equation
def SE( psi_psidot ,x,E):
 psi , psidot = psi_psidot
 psiddot =2.0* m_by_hbar2 *(V(x)-E)*psi
 return [ psidot , psiddot ]

# Potential
V= lambda x: np. where ((x >=0) & (x <=a) ,-V0 , 0.0)

# Shooting algorithm

def shoot (E):
 psi = Solver (SE ,psi0 ,x,E,dx)[: ,0]
 return psi [ -1]
b=a
x,dx=np. linspace (-b,a+b ,1000 , retstep = True )
psi0 =np.array ([0 ,1e-6])

Emin ,Emax ,dE =0 ,100 ,0.1
energies =np. arange (Emin ,Emax ,dE)
shoots =[ shoot (E) for E in energies ]
Ev_guess = energies [np. where (np. diff (np. signbit ( shoots )))]
Ev= [ opt. newton (shoot ,i) for i in Ev_guess [0:2]]

```

```
print (" First two bound state energies = ",Ev)

def wavefunction (E):
 psi = Solver (SE ,psi0 ,x,E,dx)[: ,0]
 psi =psi /np.sqrt ( simps(psi*psi,x))
 return psi

psi1 = wavefunction (Ev [0])
psi2 = wavefunction (Ev [1])

plt.plot (x,psi1 ,ls="--",label =" First Bound state ")
plt.plot (x,psi2 ,ls="dotted",label =" Second Bound state ")
plt.plot (x,V(x)/20 , ls="dashdot",lw =0.8 , color ="b",alpha =0.6 ,
    label ="Potential ( Scaled )")
plt.title (" First two bound states of Finite potential well ")
plt.legend (loc="best")
plt.xlabel ("x")
plt.grid ()
plt.savefig ("finite_well .png")
plt.show ()

#output
First two bound state energies =  [2.0872570796673386,
    4.909178400460912]
```

2.2 Output


First two bound states of Finite potential well
First Bound state
Second Bound state
Potential (Scaled)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

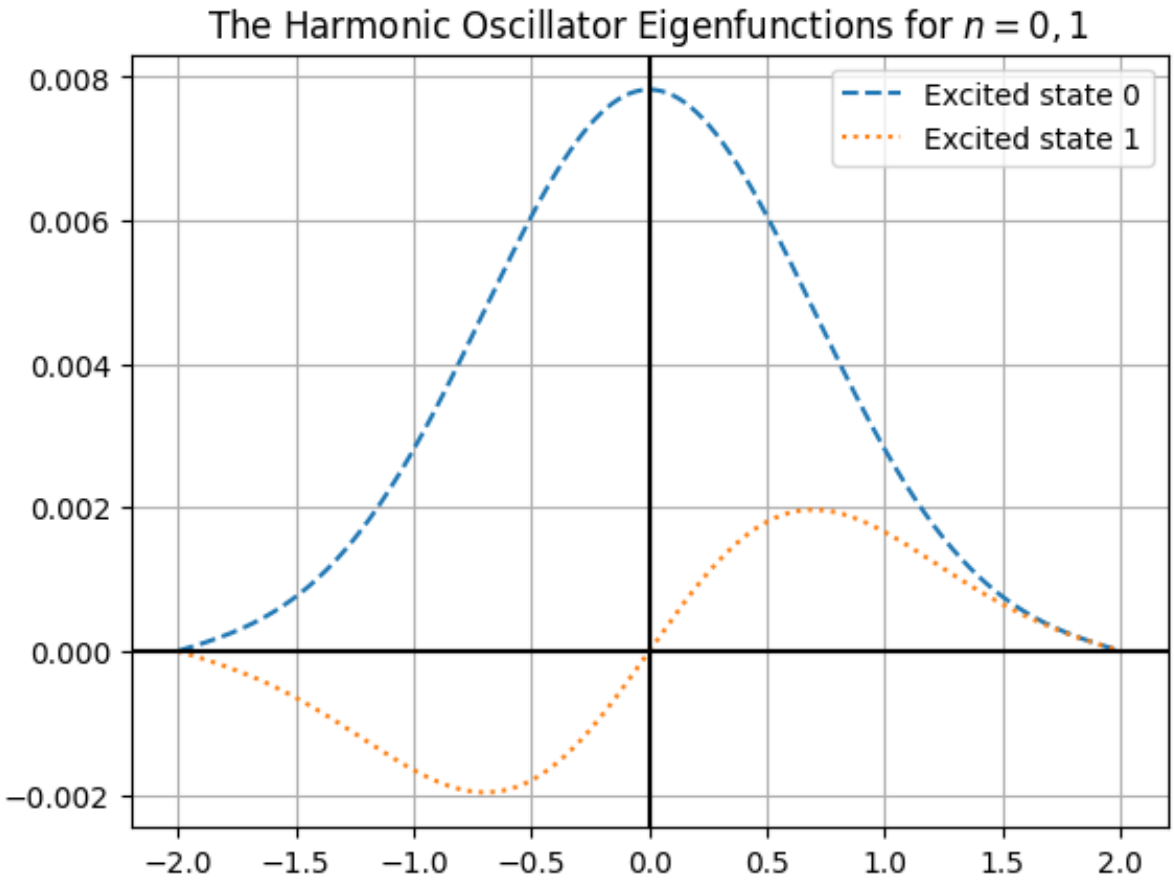
3 Particle in a Harmonic Potential

A particle of mass m is in potential V(x) = $\frac{1}{2}kx^2$ (Harmonic Oscillator). Solve the time independent Schrödinger equation numerically to find first two bound states. Use Shooting algorithm to find Eigenenergies and Euler or Numerov algorithm for solving ODE and find the normalized wave function ψ

3.1 Code

```
1
2  import numpy as np
3  import scipy as sp
4  import matplotlib.pyplot as plt
5
6  hb, m = 0.1, 1
7  def Solver(V, x, psi0, dpsi, E, dx):
8      mh = 2 * m / hb ** 2
9
10     # Numerov
11     psi1 = psi0 + dpsi * dx
12     psi = np.array([psi0, psi1])
13
14     for i in range(2, len(x)):
15         a = 2 * (1 + mh * 5 / 12 * dx ** 2 * (V(x[i - 1]) - E))
16         b = -(1 + mh * 1 / 12 * dx ** 2 * (V(x[i - 2]) - E))
17         d = (1 + mh * 1 / 12 * dx ** 2 * (V(x[i]) - E))
18
19         ppsi = a / d * psi[i - 1] + b / d * psi[i - 2]
20         psi = np.append(psi, ppsi)
21
22     return psi
23
24 # Bisection
25 def bisect_TISE(V, Emin, Emax, x, dx, nodes, tol=1e-6):
26     mxItr = 2000  # Maximum number of iterations
27     N = len(x)
28     psi0, psiN = 0, 0
29     dpsi = (-1) ** nodes * 1e-3  # 2nd initial value is set accordingly
30
31     i = 0  # Iteration
32     while abs(Emax - Emin) > tol or i < mxItr:
33         E = (Emax + Emin) / 2
34         psi = Solver(V, x, psi0, dpsi, E, dx)
35         C_nodes = 0
36
37         for k in range(1, N - 2):
38             if psi[k] * psi[k + 1] < 0:
39                 C_nodes += 1
40
41         if C_nodes > nodes:  # Counting nodes
42             Emax = E
43         elif C_nodes < nodes:
44             Emin = E
45         else:
46             if psi[N - 1] > psiN:  # Matching RHS boundary condition
47                 Emin = E
```

```
            else:
                Emax = E

        i += 1

    return E, psi

# Potential
def V(xx):
    w = 0.25
    k = m * w ** 2
    Vv = 0.5 * k * xx ** 2
    return Vv

# Main
x0, xN = -2.0, 2.0
x, dx = np.linspace(x0, xN, 1000, retstep=True)
Vx = V(x)

Emin, Emax = min(Vx), max(Vx)
node0 = 0  # Number of nodes
node1 = 1

# Bisection Calling
E0, psi_0 = bisect_TISE(V, Emin, Emax, x, dx, node0)
E1, psi_1 = bisect_TISE(V, Emin, Emax, x, dx, node1)

# Normalization
from scipy.integrate import simpson
def norm(x, psi):
    psimod2 = psi ** 2
    inte = simpson(psimod2, x = x)
    Psi = psi / inte ** 0.5
    return Psi

psi0 = norm(x, psi_0)  # Normalized psi_0
psi1 = norm(x, psi_1)  # Normalized psi_1

# Output and plotting
print(f"Eigenvalues of Ground and 1st excited state = {round(E0, 5)}, {
    round(E1, 5)}")
plt.title("The Harmonic Oscillator Eigenfunctions for $n=0,1$")
plt.plot(x, psi_0, ls = "dashed", label = f"Excited state {node0}")
plt.plot(x, psi_1, ls = "dotted", label = f"Excited state {node1}")
plt.grid()
plt.axhline(y = 0, color = "k")
plt.axvline(x = 0, color = "k")
plt.legend(loc = "best")
plt.savefig("Harmonic_oscillator.png")
plt.show()

#output
Eigenvalues of Ground and 1st excited state = 0.01534, 0.04629
```



3.2 Output

4 Hydrogen Atom

Radial part of time independent Schrödinger equation for hydrogen atom is given by

$$\frac{d^2u(r)}{dr^2} = \left[\frac{l(l+1)}{r^2} - \frac{2}{r} - E_n\right] u(r)$$

- $R(r) = \frac{u(r)}{r}$ = Radial wave function
- r = Dimensionless radial distance
- $E_n = -\frac{1}{n^2}$ = Dimensionless Energy for principal quantum number n
- l = Angular momentum quantum number

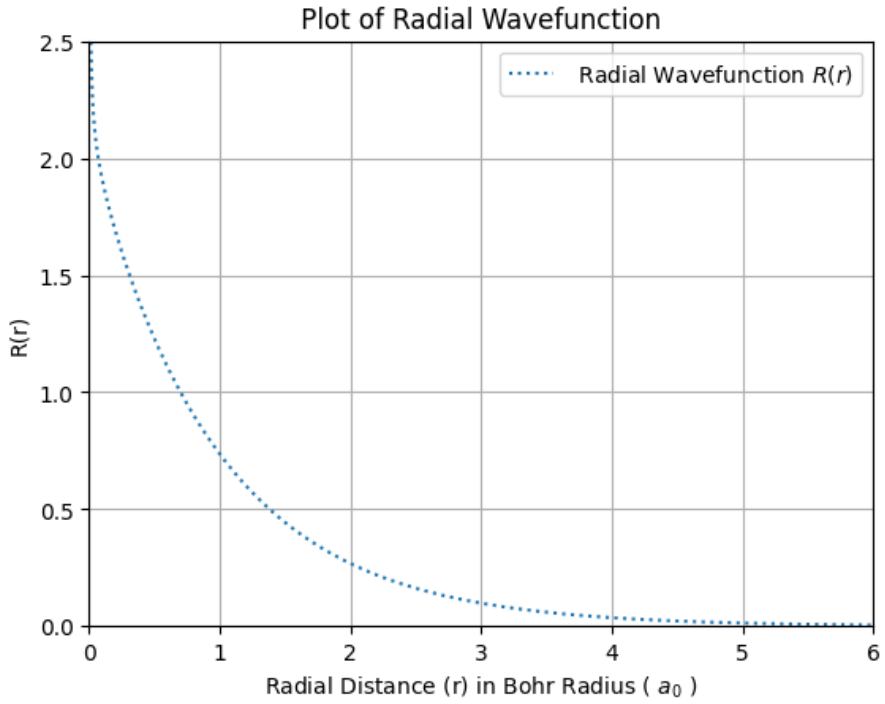
Numerically solve the given equation to find the radial part of wave function for 1s electron using Euler or Numerov algorithm and hence find the electron probability density. Also evaluate the average and most probable radial distance of the 1s electron from the nucleus and compare with their theoretical values.

4.1 Code

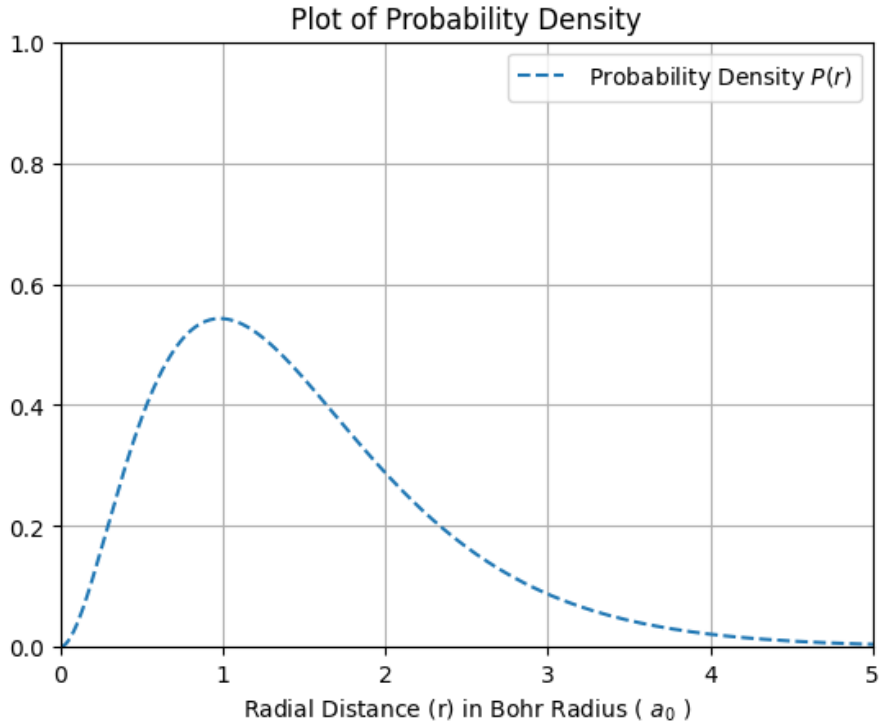
```
1
2 from scipy import optimize as opt
3 import numpy as np
4 from scipy . integrate import simps
5 import matplotlib . pyplot as plt
6
7 # Euler method
8 def Solver (f,z0 ,s,En):
9  zs =[ z0]
10  z=z0
11  for i in range (len(s) -1):
12   z=z+(s[i+1] -s[i])*np. array (f(z,s[i],En))
13   zs. append (z)
14  return np. array (zs)
15 def SEr (y,r,En):
16   (u,up)=y
17   upp = (l*(l+1)/r**2 -2/r-En)*u
18   return np. array ([up ,upp ])
19 # Parameters
20 n,l=1 ,0 # 1s state
21
22 En = -1.0/n **2
23
24 def SolveSEr (En):
25  rb=r [:: -1]
26  u0=np. array ([0.0 , -1e-3])
27  ub= Solver (SEr ,u0 ,rb ,En)
28  u=ub [: ,0][:: -1]
29  u=u/np. sqrt ( simps (u**2 ,r))
30  return u
31 r=np.logspace( -4.0,2 ,1000)
32 u= SolveSEr(En)
33
34 #Radial Solution
35 R=u/r
36
37 plt . plot (r,R,":",label =" Radial Wavefunction $R(r)$")
38 plt . xlabel ('Radial Distance (r) in Bohr Radius ( $a_0$ )')
39 plt . title (" Plot of Radial Wavefunction ")
40 plt . ylabel ("R(r)")
41 plt . grid ()
42 plt . xlim (0 ,6)
43 plt . ylim (0 ,2.5)
44 plt . legend ()
45 plt . savefig (" radial_wf .png")
46 plt . show ()
47
48 # Probability density
49 PD=u **2
50
51 plt . plot (r,PD ,"--",label =" Probability Density $P(r)$")
52 plt . xlabel ('Radial Distance (r) in Bohr Radius ( $a_0$ )')
53 plt . title (" Plot of Probability Density ")
54 plt . xlim (0 ,5)
55 plt . ylim (0 ,1)
56 plt . grid ()
57 plt . legend ()
58 plt . savefig (" prob_density .png")
```

```
plt . show ()


 # Computational Average Value
r_avg_comp = simps (u*r*u,r)

# Computational Most Probable value
r_mp_comp = r[np. argmax (PD)]


print (" COMPUTATIONAL :")
print (f" Average distance = { r_avg_comp } a_0")
print (f" Most probable distance = { r_mp_comp } a_0\n\n")

# Theoritical Values
r_avg_th = 1.0 #unit of bohr radius
r_mp_th = 1.5

print (" THEORETICAL :")
print (f" Theoretical Average distance = { r_avg_th } a_0")
print (f" Theoretical Most probable distance = { r_mp_th } a_0\n\n")

print (" ERRORS :")
print (f"Average distance = { round (abs( r_avg_th - r_avg_comp ) ,4)}
    a_0 ")
print (f" Most probable distance = { round (abs( r_mp_th - r_mp_comp )
    ,4)} a_0 \n")

print (" Where a_0 is Bohr Radius ")


#Output
COMPUTATIONAL :
 Average distance = 1.4897000356516839 a_0
 Most probable distance = 0.9862658461312821 a_0


 THEORETICAL :
 Theoretical Average distance = 1.0 a_0
 Theoretical Most probable distance = 1.5 a_0


 ERRORS :
Average distance = 0.4897a_0
 Most probable distance = 0.5137 a_0

 Where a_0 is Bohr Radius
```

4.2 Output



(a) Wave function



(b) Probability density

5 Triangular Potential Well

A particle of mass m is in potential

$$V(x) = \begin{cases} \infty, & x < 0 \\ x, & x \geq 0 \end{cases}$$

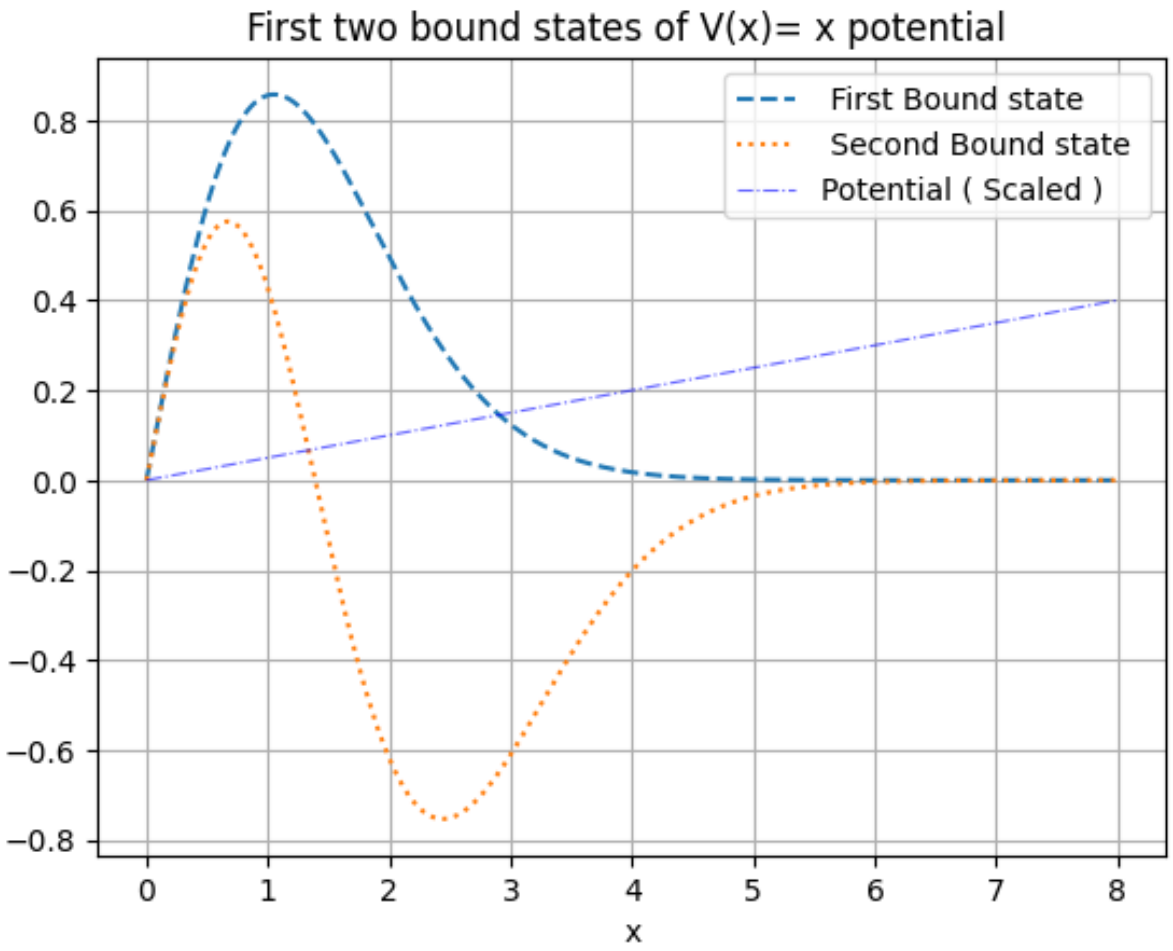
Solve the time independent Schrödinger equation numerically to find first two bound states. Use Shooting algorithm to find Eigenenergies and Euler or Numerov algorithm for solving ODE and find the normalized wave function ψ
Given $\frac{2m}{\hbar^2} = 1$

5.1 Code

```

from scipy import optimize as opt
import numpy as np
import matplotlib . pyplot as plt
from scipy . integrate import simps

# Parameters
a,m_by_hbar2 = 10 ,1
# Euler method
def Solver (f,z0 ,x,E,dx):
 zs =[ z0]
 z=z0
 for i in range (len(x) -1):
  z=z+dx*np. array (f(z,x[i],E))
  zs. append (z)
 return np. array (zs)

# Schrodinger Equation
def SE( psi_psidot ,x,E):
 psi , psidot = psi_psidot
 psiddot =2.0* m_by_hbar2 *(V(x)-E)*psi
 return [ psidot , psiddot ]

# Potential
V= lambda x: x

# Shooting algorithm

def shoot (E):
 psi = Solver (SE ,psi0 ,x,E,dx)[: ,0]
 return psi [ -1]
b=a
x,dx=np. linspace (0,a ,1000 , retstep = True )
psi0 =np.array ([0 ,1e-6])

Emin ,Emax ,dE =0 ,100 ,0.1
energies =np. arange (Emin ,Emax ,dE)
shoots =[ shoot (E) for E in energies ]
Ev_guess = energies [np. where (np. diff (np. signbit ( shoots )))]
Ev= [ opt. newton (shoot ,i) for i in Ev_guess [0:2]]

```

```
print (" First two bound state energies = ",Ev)

def wavefunction (E):
 psi = Solver (SE ,psi0 ,x,E,dx)[: ,0]
 psi =psi /np.sqrt ( simps(psi*psi,x))
 return psi

psi1 = wavefunction (Ev [0])
psi2 = wavefunction (Ev [1])

plt.plot (x,psi1 ,ls="--",label =" First Bound state ")
plt.plot (x,psi2 ,ls="dotted",label =" Second Bound state ")
plt.plot (x,V(x)/20 , ls="dashdot",lw =0.8 , color ="b",alpha =0.6 ,
    label ="Potential ( Scaled )")
plt.title (" First two bound states of V(x)= x potential  ")
plt.legend (loc="best")
plt.xlabel ("x")
plt.grid ()
plt.savefig ("finite_well .png")
plt.show ()

#output
 First two bound state energies =  [1.850844091291765,
    3.239883890236296]
```

5.2 Output


First two bound states of V(x)= x potential
First Bound state
Second Bound state
Potential (Scaled)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
1
2
3
4
5
6
7
8
x

6 Time Dependent Schrödinger Equation

Write Python code for solving time dependent Schrödinger equation in one dimension to study time evolution of wave packet moving in free space for a Gaussian wave packet using Crank-Nicolson Algorithm.

The Gaussian wave packet will be of following form :

$$\psi(x,t) = \exp\left(-\frac{(x-x_0)^2}{a} + ikx\right),$$

where x_0, a and k are constants and $i = \sqrt{-1}$.

Print the output wave function at appropriately chosen three different instances of time to exhibit time evolution.

6.1 Code

```
1
2  import numpy as np
3  import scipy as sp
4  import matplotlib . pyplot as plt
5
6  l=40
7  dx =0.01
8  ti,tf,dt =0,10,0.1
9  N =1000
10 x,dx=np. linspace (-l/2,l/2,N, retstep = True )
11
12 t=np. arange (ti ,tf+dt ,dt)
13
14 x0 ,sig ,k= -10.0 ,2.0 , np.pi /10
15
16 gwp = 1/((2* np.pi) **0.5* sig) **0.5* np.exp (-(x-x0) **2/(4* sig**2)
       +1j*k*x)
17
18 line, = plt. plot (x,np.abs (gwp **2) )
19
20 # Potential
21 V=0*x[1: -1]
22
23 alpha = 1j*dt /(2* dx **2)
24 beta = 1+2* alpha +(1j*dt*V) *0.5
25 gamma = 1 -2* alpha -(1j*dt*V) *0.5
26
27
28
29 U1 = -alpha *np. eye (N -2,k= -1)+np. diag ( beta )-alpha *np.eye (N
       -2,k=1)
30 U2 = alpha *np. eye(N -2,k= -1)+np. diag ( gamma )+ alpha *np.eye (N
       -2,k=1)
31
32 U1_inv =np. linalg .inv (U1)
33
34  # Transformation matrix P
35 P=np.dot(U1_inv ,U2)
36
```

```
plt . xlabel ("x (a.u.)")
plt . ylabel ("$|\ psi(x) |^{2} $")
plt . title (" Time evolution of a Gaussian wave - packet ")
tt =0
T_inst =[0 ,2 ,4,6,8]
k = ['-','dashdot','--','dotted','solid']
for i,T in enumerate ( T_inst ):
  while tt <T:
   gwp [1: -1]= np.dot(P,gwp [1: -1])
   tt += dt
  plt.plot (x,np.abs(gwp)**2 , ls = k[i],label =f't={ T_inst [i]} ')
  plt.legend (loc='best' )
plt.grid ()
plt.savefig (" Gaussian_wavepacket .png")
plt.show ()
```

6.2 Output