Name: Gaurav
Roll no: 2020PHY1122

Date | Page

Q1 a) while solving Schrodinger equation using Numerov method, we don't get correct asymptotic behaviour at end points when integrated from x_{min} to x_{max} because the classical turning points, the solution should contain only that part of solution which suggest that $x \to \infty$, $\psi \to 0$.

General solution:

$$\psi(x,t) = Ae^{-ikx} + Be^{ikx}$$

Beyond classical turning point

$$\psi(x,t) \to 0$$

if $x > x_{classical}$,

then $B = 0$

when, $x \to 0$, $Be^{ikx} \to \infty$, which will not lead to $\psi \to 0$

and if $x < -x_{classical}$ then $A = 0$

but our numerically calculated solution contains both exponentially increasing and decreasing terms leading to wrong asymptotic behaviour

b) To get the correct asymptotic behaviour,

1. We need to choose a particular point in the classical allowed region, let it be x_p

2. Perform numerov integration from x_{min} to x_p and then backward integration from x_{max} to x_p using numerov

3. We will check whether the value at x_p is same or not in both the cases.

4 If it is not, then we multiply one of them by a constant so that now at x_p the values of ψ is same when calculated from both directions

$$\psi_L(x=x_p) = \psi_R(x=x_m) \quad \text{(i)}$$

5 Now; we ensure that the values of derivative at x_p, calculated from both sides should come out to be same.

$$\left.\frac{d\psi_L}{dx}\right|_{x=x_p} = \left.\frac{d\psi_R}{dx}\right|_{x=x_m} \quad \text{(ii)}$$

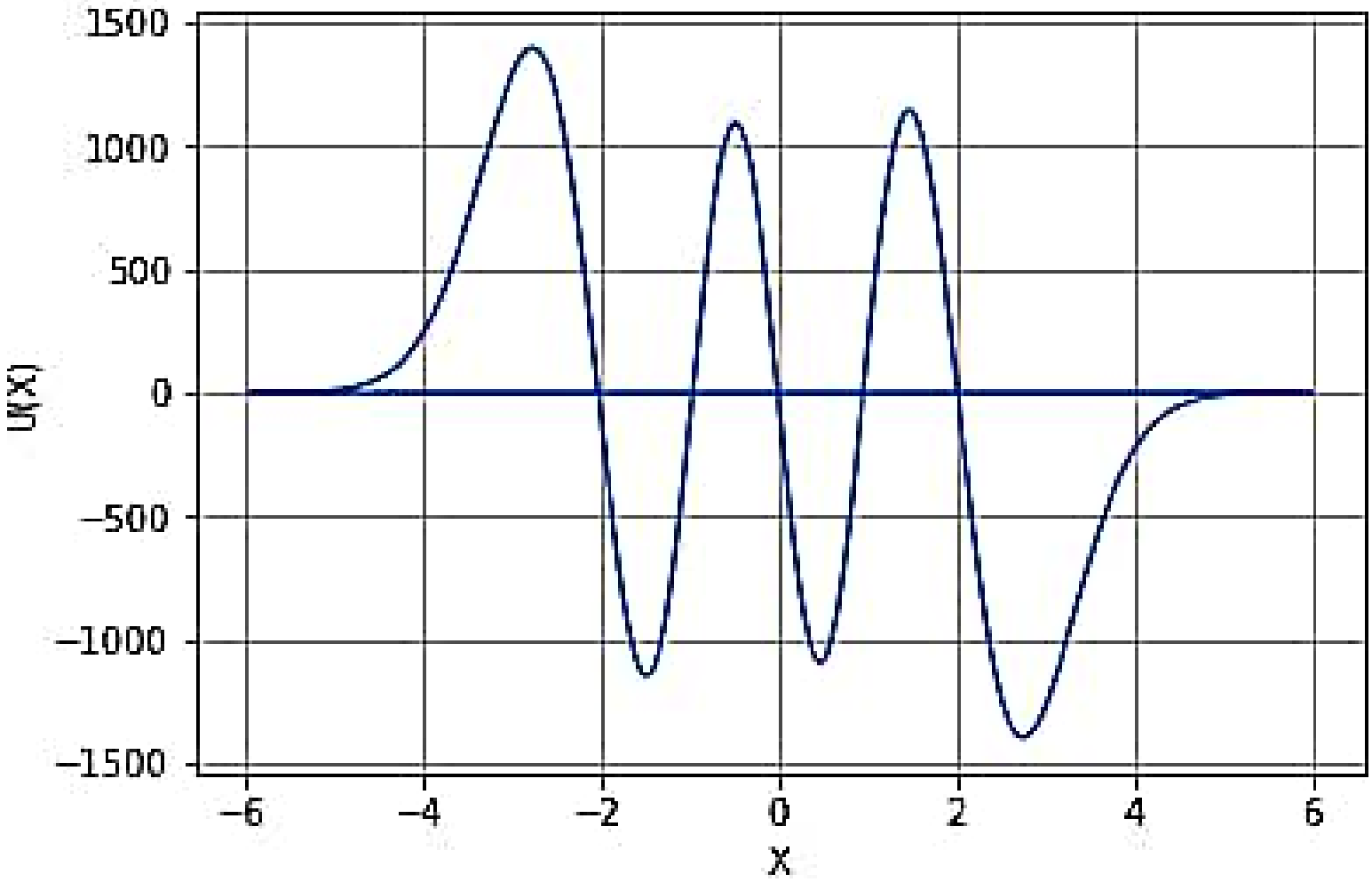
by satisfying both eqn (i) & (ii), it will give us correct asymptotic behaviours

```python
#name : Gaurav
#rollno : 2020PHY1122

import numpy as np
import matplotlib.pyplot as plt

# POTENTIAL FUNCTION

def f(e,z):
    return 2*(e - (1/2)*(z**2))

# NUMEROV

def Numerov(a,b,n,N,f): #numerov method

    h = (b-a)/N     #step size
    x = np.arange(a,b+h,h)
    e = n+0.5
    u = np.zeros([N+1])

    C = 1 + ((h**2)/12)*f(e,x)

    if n%2!=0: #odd states
        u[0] = 0
        u[1]= h
    else :      #even states
        u[0] = 1
        u[1] = (6-5*C[0])*(u[0]/C[1])

    for i in range(2,N+1):
        u[i] = ((12 - 10*C[i-1])*u[i-1] - C[i-2]*u[i-2])/C[i]

    return x,u
```

```
35  #PARITY
36  def parity(x,u,n):
37      x_minus = -x[1:]
38
39      X = np.append(x_minus[-1::-1], x)
40
41      if n % 2 != 0:  # odd states
42
43          u_minus = -u[1:]
44          U = np.append(u_minus[-1::-1], u)
45
46      else:  # even states
47          u_minus = u[1:]
48          U = np.append(u[-1::-1], u_minus)
49
50      return X,U
51
52  # MULTIPLYING BY FACTOR
53
54  def factor(a,b,n,N,f):
55
56      if n%2==0:
57          x,u=Numerov(a,b,n,N,f)
58      else:
59          x, u = Numerov(a, b, n, N, f)
60
61      xcl1 = round(np.sqrt(2 * n + 1), 2)
62      p = 0
63      for i in x:
64          if (round(i, 2)) == xcl1:
65              break
66          else:
67              p += 1
68      c=x[p-1]
69
70      h = (b - a) / N
```

```
69
70      h = (b - a) / N
71      N2= int((c+h - a)/h)
72
73      if n%2==0:
74
75          x1,u1=Numerov(a,c+h,n,N2,f)
76      else:
77
78          x1, u1 = Numerov(a, c+h, n, N2, f)
79      N3 = int((b - (c-h)) / h)
80      x2,u2=Numerov(b,c-h,n,N3,f)
81
82      h=x2[1]-x2[0]
83      k= u1[-1]/u2[-1]
84      u2_new= k*u2
85      u2=u2_new
86
87      return u2[-3], u2[-2], u1[-3], c,u1[-2], u2[-1], u1[-1], h
88
89  print(factor(0,6,2,500,f))
90
91  def derivative(factor):
92      num= factor[2] + factor[0] - (12*factor[3] - 10)*factor[1]
93      d_der= num/factor[4]
94      return d_der
95
96
97  def bisection(a,b,nmax,N,f,derivative,tol,factor):
98      nmin=0
99      d_der=derivative(factor(a,b,nmax,N,f))
100     while abs(d_der)>=tol:
101         if d_der<=0:
102             nmin=(nmin+nmax)/2
103         else:
104             nmax=(nmin+nmax)/2
```

```
103             else:
104                 nmax=(nmin+nmax)/2
105             n_new= (nmin+nmax)/2
106 
107             d_der = derivative(factor(a, b, n_new, N, f))
108 
109         return (nmax+nmin/2 + 0.5), d_der
110 
111 
112     #2ND APPROACH
113     def alternate(factor):
114         num1=factor[6]-factor[2] - factor[0] + factor[5]
115 
116         return num1
117 
118 
119     # EXAMPLE:
120 
121     x,u=Numerov(6,0,5, 100, f)
122     X,U = parity(x, u, 5)
123     plt.plot(X,U)
124     plt.grid()
125     plt.xlabel('X')
126     plt.ylabel('U(X)')
127     plt.show()
128 
129     n = [0,1,2,3,4,5]
130     e_cal = []
131     for i in n:
132 
133         e_cal.append(bisection(0,8,n[i],10000,f,alternate,10**(-5),factor)[0])
134 
135     print('the first 6 energy values are : ',e_cal)
```


1500
1000
500
0
−500
−1000
−1500
U(X)
−6
−4
−2
0
2
4
6
X

```
In [1]: runfile('D:/python work/prog sem
5/1122_Gaurav_qmLab-A7.py', wdir='D:/python
work/prog sem 5')
(-0.7101062753316718, -0.7207969539920858,
-0.7535088746730806, 2.232,
-0.7424982822320886, -0.7314857348273436,
-0.7314857348273436, -0.011999999999999567)
```

Warning

Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.

```
 the first 6 energy values are :  [0.5, 1.5,
2.5, 3.5, 4.5, 5.5]
```