3.2 Finite Difference Method

Solving boundary value problems with shooting method is vulnerable towards instability. The finite difference method has better stability characteristics, but it generally needs more computation to obtain a specified accuracy.

In this approach, the differential equation is replaced with the finite difference terms of derivatives. The number of finite difference terms corresponding to a derivative depends on the accuracy and order of the derivative. Also, the step size dx is also tuned by the required accuracy.

3.2.1 Linear finite difference method

1 The general form of a linear second order boundary value problem is given by

$$y'' = p(x)y' + q(x)y + r(x), \text{ for } x_0 \le x \le x_N,$$

$$\text{with } y(x_0) = y_0 \text{ and } y(x_N) = y_N \qquad \dots (3.15)$$

Where, $p(x)$, $q(x)$ and $r(x)$ are continuous and differentiable functions of x within the interval $x_0 \le x \le x_N$. The derivatives y' and y'' have to be approximated by finite difference scheme. This approximation scheme is called **difference-quotient approximation.** The scheme is described below.

1. First of all space discretization has to be done with step size h.

$$N = \frac{x_N - x_0}{h}$$

$$x_i = x_0 + ih \text{ for } i = 0, 1, \dots, N \qquad \dots (3.16)$$

2. The discretization of y' and y'' have to be calculated at x_i by finite difference formulas.

$$y'(x_i) = \frac{1}{2h}[y(x_{i+1}) - y(x_{i-1})] - \frac{h^2}{6}y'''(\eta_i), \quad \text{where } x_{i-1} < \eta_i < x_{i+1}$$

$$y''(x_i) = \frac{1}{h^2}[y(x_{i+1}) - 2y(x_i) + y(x_{i-1})] - \frac{h^2}{12}y^4(\xi_i), \quad \text{where } x_{i-1} < \xi_i < x_i+1$$

3. Using the above two formulas, the discrete form of equation-3.15 is given by

$$\frac{1}{h^2}\left[y(x_{i+1}) - 2y(x_i) + y(x_{i-1})\right] = \frac{p(x_i)}{2h}\left[y(x_{i+1}) - y(x_{i-1})\right] + q(x_i)y(x_i) + r(x_i) - \frac{h^2}{12}\left[2p(x_i)y'''(\eta_i) - y^4(\xi_i)\right]$$

$$\Rightarrow -\left[1 + \frac{h}{2}p(x_i)\right]y(x_{i-1}) + \left[2 + h^2q(x_i)\right]y(x_i) - \left[1 - \frac{h}{2}p(x_i)\right]y(x_{i+1}) = -h^2r(x_i) + O(h^2)$$

$$\text{for } 1 \le i \le N - 1 \qquad \dots (3.17)$$

This is the finite difference approximation of equation-3.15 for $1 \le i \le N - 1$ with truncation error $O(h^2)$. The solution at boundaries are given by $y(x_0) = y_0$, $y(x_N) = y_N$.

The matrix representation of equation-3.17 is

$$Ay = b \qquad \cdots (3.18)$$

2

where

$$\mathbf{A} = \begin{bmatrix} 2+h^2q(x_1) & -1+\frac{h}{2}p(x_1) & 0 & 0 & \cdots & 0 \\ -1-\frac{h}{2}p(x_2) & 2+h^2q(x_2) & -1+\frac{h}{2}p(x_2) & 0 & \cdots & \vdots \\ 0 & -1-\frac{h}{2}p(x_3) & 2+h^2q(x_3) & -1+\frac{h}{2}p(x_3) & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & -1+\frac{h}{2}p(x_{N-2}) \\ 0 & \cdots & \cdots & 0 & -1-\frac{h}{2}p(x_{N-2}) & 2+h^2q(x_{N-1}) \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} y(x_1) \\ y(x_2) \\ \vdots \\ \vdots \\ y(x_{N-2}) \\ y(x_{N-1}) \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} -h^2r(x_1)+\left(1+\frac{h}{2}p(x_1)\right)y_0 \\ -h^2r(x_2) \\ \vdots \\ \vdots \\ -h^2r(x_{N-2}) \\ -h^2r(x_{N-1})+\left(1-\frac{h}{2}p(x_{N-1})\right)y_N \end{bmatrix} \qquad \cdots (3.19)$$

Solution of matrix equation-3.18 provides the values $y(x_1)$, $y(x_2)$, ..., $y(x_{N-1})$. These values along with the boundary values $y(x_0)$ and $y(x_N)$ represent the complete numerical solution of equation-3.15. Numerical solution of matrix equation-3.18 will be obtained by solving that tridiagonal linear matrix system. The matrix equation can easily be solved by Gauss Elimination or Gauss-Seidel method. We do not attempt solving this matrix equation by Gauss-Seidel method, because the matrix may not satisfy diagonally dominant condition. So, this set of equations can definitely be solved with Gauss Elimination method. The overall steps for solving boundary value problems by finite difference method are described below.

... difference method with Gauss Elimination

Figure 3.28

3.3 Eigen value Problems

An eigen value problem is the differential equation which satisfy the the boundary conditions for some specific values of the unknown **internal parameter** present in that equation. Those specific values are the eigen values. The solution of that equation corresponding to a eigen value, is the eigen function. A linear eigen value problem with Dirichlet boundary conditions can be put forward by the class of **Strum-Liouville eigen value problem.**

$$\frac{d}{dx}\left(P(x)\frac{dy}{dx}\right) = Q(\lambda, x)y + R(x) \quad \text{with } y(x_0) = y_0 \text{ and } y(x_N) = y_N \qquad \text{... (3.27)}$$

With some simple transformations* above equation can be reduced to the following linear eigen value problem.

* Readers can consult any standard mathematical physics book. A simple compact introduction about Strum-Liouville eigen value problems can be found at http://people.uncw.edu/hermanr/mat463/ODEBook/Book/SL.pdf

$$\frac{d^2y}{dx^2} = p(x)\frac{dy}{dx} + q(\lambda, x)y + r(x) \quad \text{with} \quad y(x_0) = y_0 \quad \text{and} \quad y(x_N) = y_N \qquad \text{... (3.28)}$$

Here, the unknown parameter λ, has to be tuned to satisfy the the boundary conditions $y(x_0) = y_0$ and $y(x_N) = y_N$. The values of λ for which the boundary conditions are satisfied are called the **eigen values** and the corresponding solutions are the **eigen functions**. The class of Strum-Liouville eigen value problems has some properties those which are important to deal with these class of problems.

(a) The eigen values are real, countable, ordered and there is a smallest eigen value. Thus eigen values can be written as $\lambda_1 < \lambda_2 < ... < \lambda_n < ...$. However, there is no largest eigen value and $n \to \infty$, $\lambda_n \to \infty$.

(b) **For each eigen value λ_n, there exists an eigen function ϕ_n with n-1 nodes** (x_0, x_N).

(c) Eigen functions corresponding to different eigen values are orthogonal.

$$< \phi_n, \phi_m >= \delta_{nm}, \quad n, m = 1, 2, ...$$

We have to solve the differential equation-3.28 numerically. This is not an easy task. Here problem is that, we have to solve the differential equation satisfying both the boundary conditions for an accurate value of the unknown parameter(λ). For an inaccurate value of the unknown parameter, the boundary conditions couldn't be satisfied. Let us develop a numerical scheme based on finite-difference method to solve equation-3.28.

(a) First, the derivative is taken fixed* (say α) at $x = x_0$. Thus, according to finite difference ($O(h)$), the next value of solution after the boundary value, y_1, can be given by the following equation.

$$\frac{y_1 - y_0}{h} = \alpha$$

$$y_1 = y_0 + \alpha h \qquad \text{... (3.29)}$$

Here, the value of the α is chosen arbitrarily. Now, what value may be the chosen for α. The magnitude of α ($|\alpha|$) does not affect the solution, but its sign is crucial. For any arbitrary magnitude of α, the solution will be obtained unnormalized. Later normalization can be done. But, the sign of α governs the initial direction of the solution. So, if sign is not correct, the wrong answer can be obtained. The following figure will make this point clear.

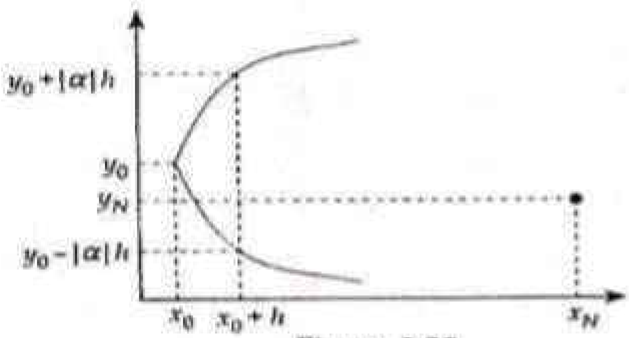

Figure 3.29

* It is a major difference from the technique for solving boundary value problem with the shooting method. In shooting method, this derivative actually tuned to satisfy the boundary conditions. Here it is kept fixed.

For any arbitrary value of α, the initial direction of the solution is completely governed by the sign of α. The other boundary $x = x_N$, may not lie on the same horizontal line parallel to x-axis.

(b) A trial value of the parameter λ (say λ_1) will be taken. If tuning of λ is performed by bisection method, then this trial value is in the middle of upper and lower bounds of λ provided by user.

(c) Next, the solution will be propagated by a finite-difference method till the other boundary $x = x_N$.

(d) Several solutions corresponding the different accurate values of λ can be obtained. Each of the solution is the eigenfunction corresponding to the eigenvalue (the accurate value of λ).

According to the property of Strum-Liouville eigenvalue problem, nth order eigenfunction contains $n - 1$ nodes. Thus, to obtain a eigenfunction of particular order, we may apply the **node counting method**. Node counting method simply counts the nodes and compares this counts with the number of nodes corresponding to a particular eigenfunction. In general, the lowest order eigenfunction does not has a node. The higher order eigenfunctions has number of nodes equal to the order number. In successive iterations, different values of λ are shooted by bisection method* to obtain the desired number of nodes. Let the lower and upper bounds of λ are given as λ^0_{mn} and λ^0_{mx}. First, the range of λ is bisected as $\lambda_1 = 0.5\ (\lambda^0_{mn} + \lambda^0_{mx})$. Number of nodes is counted for λ_1. If count is less than the required number of nodes then, we have to shift towards higher λ. So for next iteration $\lambda^1_{mn} \leftarrow \lambda_1$. If count is higher then the upper bound will be modified as $\lambda^1_{mx} \leftarrow \lambda_1$. Next iteration again starts with $\lambda_2 \leftarrow 0.5(\lambda^1_{mn} + \lambda^0_{mx})$ or $\lambda_2 = 0.5(\lambda^0_{mn} + \lambda^1_{mx})$. This process continues till the number of nodes does not match with the desired value. The idea of node counting method will be cleared from the below figure.

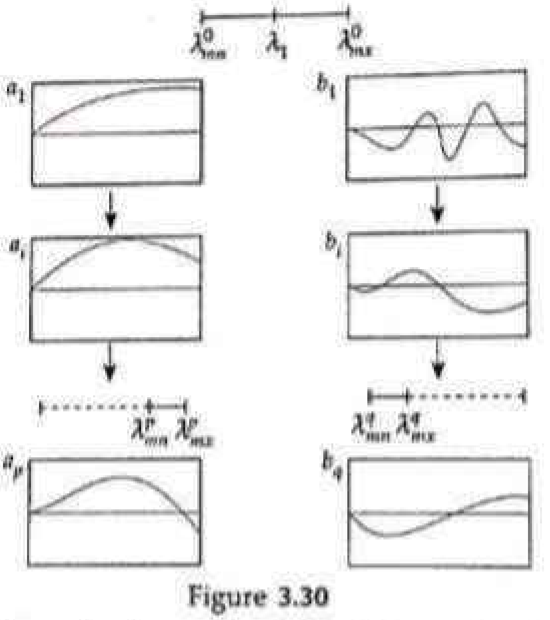

Figure 3.30

* Newton-Raphson method can also be employed. But Newton-Raphson method needs derivative which is not simple to calculate. Whereas, bisection method does not need any derivative calculation. So, it is simple to implement. But, bisection method can only be applied with upper and lower bounds. So, a gross idea about the bounds of eigenvalue is required.

The figure describes the process of searching the value of λ for single node eigenfunction. Here, the eigenfunction with single node is being searched. First, the λ interval is bisected at λ_1, and the solution for λ_1 is obtained by any finite-difference method. Any one of the two hypothetical situations (a_1 and b_1) is thought to be happened. The case a_1 with zero number of nodes and b_1 with four number of nodes. Next, the lower and upper bounds are adjusted according to the scheme said above. After, ith iteration, let the situations resembles with the cases a_i and b_i respectively. Let after pth and qth iterations, the solution obtained with single node. Here the node counting algorithm ends. We, found out the range of λ for which the solution has single node. But still the eigenfunction does not obtained. Now the second search for exact λ will starts for which the boundary condition $y(x_N) = y_N$ are exactly satisfied by the solution.

(e) Node counting method shrinks the overall λ regime into a small zone within which the required number of nodes of the solution is satisfied. Now from here, the matching of boundary condition starts. Since, around the boundary, the values of solution $y(x)$ will be almost same. Thus, we have to tune λ such that $y_{N-1} \approx y_N$. This tuning of λ is also be done by bisection method. The following figure describes this process.

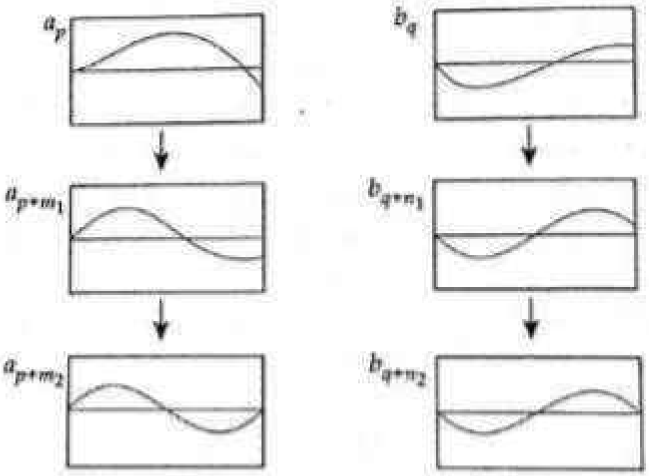

Figure 3.31

(f) Since, we have started with an arbitrary value of derivative (α), the eigenfunction obtained in this process is not normalized. Before we proceed further, we have to clear this point. An eigenvalue problem provides the eigenfunction with a undetermined pre-factor. It means eigenfunction is of the form

$$\psi(x) = A\phi(x) \qquad \text{... (3.30)}$$

Where, the constant A is not still been calculated. This can be determined with a completely separate condition. This is called the normalization condition. One common normalization condition is

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1 \qquad \text{... (3.31)}$$

Irrespective of the choice of α, this normalization condition puts the eigenfunction in the same scaling. Still if some doubt is there, then the following example will certainly clear it. Let, for some arbitrary choice of α, Mr. X obtained the solution of the following boundary value problem

$$\frac{d^2y}{dx^2} = -\lambda y \quad \text{with } y(0) = 0,\ y(2\pi) = 0$$

as

$$\lambda_n = \frac{n^2}{4},\ y_n(x) = 10\sin\left(\frac{nx}{2}\right) \qquad n = 1, 2, 3, \ldots$$

For other choice of α, Mr. Y obtained the solution as

$$\lambda_n = \frac{n^2}{4},\ y_n(x) = 100\sin\left(\frac{nx}{2}\right) \qquad n = 1, 2, 3, \ldots$$

The pre-factor of eigenfunctions $y_n(x)$ are different in two cases. But both the eigenfunctions satisfy the differential equation along with the boundary conditions. Then which one is correct? Actually both of them are correct with two different scalings. These solutions are said to be unnormalized. Now, if we apply a common normalization condition-3.31, then all these solutions can be represented with same scaling. That is normalization, that all. According to the choice of finite-difference method (mentioned in step (c)) to solve the differential equation-3.28, we can develop two schemes to solve eigenvalue problems numerically. Two finite-difference methods are implemented.

3.3.1 Central difference method

The linear eigenvalue problem-3.28 is taken. The mathematical steps of solution by central difference method is presented in section-3.2.1. Equation-3.17 provides the following propagation of solution.

$$\left[1-\frac{h}{2}p(x_i)\right]y(x_{i+1}) = [2 + h^2q(\lambda, x_i)]y(x_i) - \left[1+\frac{h}{2}p(x_i)\right]y(x_{i-1}) + h^2r(x_i)$$

$$\Rightarrow y(x_{i+1}) = \frac{a}{d}\,y(x_i) + \frac{b}{d}\,y(x_{i-1}) + \frac{c}{d} \qquad \text{for } 1 \le i \le N-1$$

Where $a = [2 + h^2q(\lambda, x_i)]$, $b = -\left[1+\frac{h}{2}p(x_i)\right]$, $c = h^2r(x_i)$ and $d = \left[1-\frac{h}{2}p(x_i)\right]$ ... (3.32)

Determination of propagator of eigenvalue equation-3.28 according to central difference method (equation-3.32), can be calculated by the following steps.

Algorithm 3.3.1.1 – procedure propCntDiff

Define the mathematical functions $p(x)$, $q(\lambda, x)$, $r(x)$

Get the values x_0, y_0, x_N, y_N, y_1

Get the value of h

for $i = 0, 1, 2, \ldots, N$

  $x_i \leftarrow x_0 + ih$

for $i = 2, 3, \ldots, N-1$

  $a \leftarrow [2 + h^2q(\lambda, x_i)]$

  $b \leftarrow -[1 + \frac{h}{2}p(x_i)]$

  $c \leftarrow h^2r(x_i)$

  $d \leftarrow [1 - \frac{h}{2}p(x_i)]$

  $y_i \leftarrow \frac{a}{d}y_{i-1} + \frac{b}{d}y_{i-2} + \frac{c}{d}$

The determination of eigen values and eigen functions (solution of equation-3.2...) in steps (d) and (e) are presented in the following algorithm.

Algorithm 3.3.1.2–Solving eigen value problem by central difference method

Define procedure propCntDiff ( )
Define the procedure for mathematical functions $p(x)$, $q(\lambda, x)$, $r(x)$
Get the values x_0, y_0, x_N, y_N, y_1
Get the value of h, λ_{mn}, λ_{mx}
Get the value of n, k_{mx}
for $i = 0, 1, 2, \ldots, N$
    $x_i \leftarrow x_0 + ih$
$k \leftarrow 0$
while $|\lambda_{mx} - \lambda_{mn}| > 10^{-6}$ and $k < k_{mx}$
    $\lambda \leftarrow \frac{1}{2}(\lambda_{mn} + \lambda_{mx})$
    $Y_i \leftarrow$ propCntDiff ( ) for $i = 0, 1, 2, \ldots, N$
    $c \leftarrow 0$
    for $i = 1, 2, \ldots, N - 2$
        if $Y_i Y_{i+1} < 0$
            $c \leftarrow c + 1$
    if $c > n$
        $\lambda_{mx} \leftarrow \lambda$
    else if $c < n$
        $\lambda_{mn} \leftarrow \lambda$
    else
    if $Y_{N-1} > y_N$
    $\lambda_{mn} \leftarrow \lambda$
    else if $Y_{N-1} < y_N$
    $\lambda_{mx} \leftarrow \lambda$
    $t \leftarrow t + 1$
if $t < t_{mx}$
λ, x and Y are obtained

Two Python functions **propCntDiff( )** and **CntDiffEigVal( )** are developed accordi... above two algorithms. Both of the Python functions are kept in the script CntDiffEig...

6

Program 3.3.1.1 – CntDiffEigVal.py

```
# Propagator with central difference calculation
def propCntDiff(pr, p, q, r, x, y, dx):
    # Arguments:
    #   p ==> User defined function with parameter 'lb', in the form p(lb, x)
    #   q ==> User defined function
    #   r ==> User defined function
    #   x ==> x-array
    #   y ==> y-array of the propagator at previous iteration
    #  dx ==> increment along x-axis
    #
    # Returns:
    #   yy ==> y-array of the propagator at current iteration
    #
    N = len(x)
    yy = [y[i] for i in range(N)]
    for i in range(2, N):
        a = 2 + dx**2 * q(pr, x[i-1])
        b = -(1+ dx/2 * p(x[i-1]))
        c = dx**2 * r(x[i-1])
        d = 1- dx/2 * p(x[i-1])
        yy[i] = a/d * yy[i-1] + b/d * yy[i-2] + c/d
    return yy

# Determination of eigenvalue and eigenfunction
def CntDiffEigVal(prMn, prMx, p, q, r, x0, y0, xN, yN, y1, dx, nodes, mxItr):
    # Arguments:
    #   prMn, prMx ==> lower and upper bounds of eigenvalue
    #   p ==> User defined function with parameter 'lb', in the form p(lb, x)
    #   q ==> User defined function
    #   r ==> User defined function
    #   x0, y0 ==> left boundary condition
    #   xN, yN ==> right boundary condition
    #   y1 ==> estimate of solution at the next point after left boundary
    #   dx ==> increment along x-axis
    #   nodes ==> Number of nodes of the eigenvalue
    #   mxItr ==> Number maximum allowed iterations
    #
    # Returns:
    #   pr ==> eigenvalue
```

```
    #  x ==> x-array of solution
    #  yy ==> y-array of solution
    #
    N = int( (xN - x0)/ dx )
    dx = (xN - x0)/N
    x = [x0+i*dx for i in range(N+1)]
    y = [0 for i in range(N+1)]
    y[0], y[1], y[N] = y0, y1, yN
    itr = 0
    while abs(prMx - prMn) > 1e-6 and itr < mxItr:
        pr = 0.5*(prMn + prMx)
        yy = propCntDiff(pr, p, q, r, x, y, dx)
        cnt = 0
        for i in range(1, N-2):
            if yy[i]*yy[i+1] < 0:
               cnt += 1
        if cnt > nodes:
           prMx = pr
        elif cnt < nodes:
           prMn = pr
        else:
           if yy[N-1] > yN:
           prMn = pr
           elif yy[N-1] < yN:
           prMx = pr
           itr += 1
  if itr < mxItr:
      return pr, x, yy
  else:
      return None, None, None
```

Several applications of the Python functions **propCntDiff( )** and **CntDiffEig...** solving eigen value problems are listed below.

(a) First problem is to determine first two eigen values and eigen functions of the ... equation.

CntDiffEigVal()

$$\frac{d^2y}{dx^2} = \frac{-3}{x}\frac{dy}{dx} - \frac{\lambda}{x^2} \quad \text{with } y(1) = 0,\ y(2) = 0$$

The lower and upper bounds of the eigen values is taken as [20, 90].

Example 3.3.1.1 – CntDiffEigValEx01.py

```
from math import *
import matplotlib.pyplot as plt
from CntDiffEigVal import CntDiffEigVal

def p(x):
    return -3/x

def q(lb, x):
    return -lb/x**2

def r(x):
    return 0
stln = ['k', 'k--'] # array line styles
dx = 0.001 # increment along x-axis
mxItr = 100 # maximum allowed iterations
lbMn, lbMx = 20, 90 # lower and upper bounds of eigenvalue
x0, y0, xN, yN = 1, 0, 2, 0 # boundary conditions
for nodes in range(2):
    y1 = y0 + (-1)**nodes * 1e-4 # point after left boundary
    lb, x, y = CntDiffEigVal(lbMn, lbMx, p, q, r, x0, y0, xN, yN, y1,
                                    dx, nodes, mxItr)

    plt.plot(x, y, stln[nodes],
            label = r'Eigenfunction for $\lambda$ = %.3f' %lb)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc='best', prop={'size':12})
plt.show()
```

The eigen values and eigen functions those are determined from the above code are illustrated below.

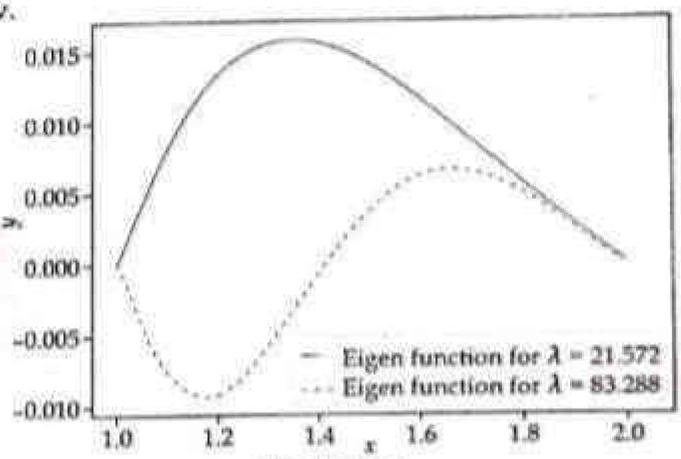

Figure 3.32

Figure 3.33

3.3.2 Numerov method

Generally, Numerov method is employed* for the linear eigen value problem given by the following equation.

$$\frac{d^2y}{dx^2} = q(\lambda, x) + r(x) \qquad \text{... (3.33)}$$

Numerov method discretizes the differential equation with higher accuracy. The $(N + 1)$ point space discretization is given by x_i for $0 \le i \le N$. Similarly, the discretized solution is y_i for $0 \le i \le N$. Where $x_i = x_0 + ih$, with h as the increment along x-axis.

The Taylor series expansion of continuous function $y(x)$ about a point x_i is

$$y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2!}y''(x_i) + \frac{h^3}{3!}y^{(3)}(x_i) + \frac{h^4}{4!}y^{(4)}(x_i) + \frac{h^5}{5!}y^{(5)}(x_i) + O(h^6)$$

Thus, the following relations can be written.

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{h^2}{2!}y''(x_i) + \frac{h^3}{3!}y^{(3)}(x_i) + \frac{h^4}{4!}y^{(4)}(x_i) + \frac{h^5}{5!}y^{(5)}(x_i) + O(h^6)$$

$$y(x_{i-1}) = y(x_i) - hy'(x_i) + \frac{h^2}{2!}y''(x_i) - \frac{h^3}{3!}y^{(3)}(x_i) + \frac{h^4}{4!}y^{(4)}(x_i) - \frac{h^5}{5!}y^{(5)}(x_i) + O(h^6)$$

The addition of the above two equations produces

$$y(x_{i+1}) - 2y(x_i) + y(x_{i-1}) = h^2y_i'' + \frac{h^4}{4!}y_i^{(4)} + O(h^6) \qquad \text{... (3.34)}$$

The fourth order derivative can be simplified as follows.

$$y^{(4)}(x_i) = \frac{d^2}{dx^2}\left(\frac{d^2y}{dx^2}\right)_{x=x_i} = \frac{d^2}{dx^2}\left(q(\lambda, x)y(x) + r(x)\right)_{x=x_i} \quad \text{from equation-3.33}$$

$$h^2y^{(4)}(x_i) = q(\lambda, x_{i+1})y(x_{i+1}) - 2q(\lambda, x_i)y(x_i) + q(\lambda, x_{i-1})y(x_{i-1}) + r(x_{i+1}) - 2r(x_i) + r(x_{i-1}) + O(h^4)$$

Replacing this in equation-3.34 one can simplify

---

* Numerical calculation of eigen values and eigen functions of equation-3.28 by Numerov method, is much complicated.

$$y(x_{i+1}) - 2y(x_i) + y(x_{i-1}) = h^2[q(\lambda, x_i)y(x_i) + r(x_i)] + \frac{h^2}{12}[q(\lambda, x_{i+1})y(x_{i+1}) + r(x_{i+1}) - 2q(\lambda, x_i)y(x_i) - 2r(x_i) + q(\lambda, x_{i-1})y(x_{i-1}) + r(x_{i-1})] + O(h^6)$$

$$y(x_{i+1})\left(1 - \frac{h^2}{12}q(\lambda, x_{i+1})\right) - 2y(x_i)\left(1 + \frac{5h^2}{12}q(\lambda, x_{i+1})\right) + y(x_{i-1})\left(1 - \frac{h^2}{12}q(\lambda, x_{i-1})\right)$$
$$= \frac{h^2}{12}\left(r(x_{i+1}) + 10r(x_i) + r(x_{i-1})\right) + O(h^6)$$

$$y(x_i) = \frac{a}{d}y(x_{i-1}) + \frac{b}{d}y(x_{i-2}) + \frac{c}{d} \text{ with transformation } i \leftarrow i - 1$$

Where $a = 2\left(1 + \frac{5h^2}{12}q(\lambda, x_{i-1})\right)$, $b = -\left(1 - \frac{h^2}{12}q(\lambda, x_{i-2})\right)$

$$c = \frac{h^2}{12}\left(r(x_i) + 10r(x_{i-1}) + r(x_{i-2})\right) \text{ and } d = \left(1 - \frac{h^2}{12}q(\lambda, x_i)\right) \quad \dots (3.35)$$

This is the formulation of solving a eigenvalue equation by Numerov method. Nu... method holds accuracy $O(h^6)$ which is even better than fourth order Runge-Kutta me... The propagator of a eigenvalue equation according to Numerov method can be obtain... executing the following steps.

Algorithm 3.3.2.1—Propagator according to Numerov method

Define the mathematical functions $p(x)$, $q(\lambda, x)$
Get the values x_0, y_0, x_N, y_N, y_1
Get the value of h
for $i = 0, 1, 2, \dots, N$
    $x_i \leftarrow x_0 + ih$
for $i = 2, 3, \dots, N-1$
    $a \leftarrow 2\left[1 + \frac{5h^2}{12}q(\lambda, x_{i-1})\right]$
    $b \leftarrow -\left[1 - \frac{h^2}{12}q(\lambda, x_{i-2})\right]$
    $c \leftarrow \frac{h^2}{12}[r(x_i) + 10r(x_{i-1}) + r(x_{i-2})]$
    $d \leftarrow \left[1 - \frac{h^2}{12}q(\lambda, x_i)\right]$
    $y_i \leftarrow \frac{a}{d}y_{i-1} + \frac{b}{d}y_{i-2} + \frac{c}{d}$

The rest part, i.e, node counting and satisfying boundary conditions according to bise... method, remains same as that of the algorithm-3.3.1.2. Combining the algorithms-3.3.1.2 &... 3.3.2.1, the Python functions **propNumerov( )** and **numerovEigVal( )** could be developed

Program 3.3.2.1—numerovEigVal.py

```
# Propagator with Numerov method
def propNumerov(pr, q, r, x, y, dx):
    # Arguments:
    #   p ==> User defined function with parameter 'lb', in the form p(lb, x)
    #   q ==> User defined function
    #   r ==> User defined function
    #   x ==> x-array
    #   y ==> y-array of the propagator at previous iteration
    # dx ==> increment along x-axis
    #
    # Returns:
    #   yy ==> y-array of the propagator at current iteration
    #
    N = len(x)
    yy = [y[i] for i in range(N)]
    for i in range(2, N):
        d = 1 - dx**2/12 * q(pr, x[i])
        a = 2*(1 + 5*dx**2/12* q(pr, x[i-1]))
        b = -(1 - dx**2/12* q(pr, x[i-2]))
        c = dx**2/12*( r(x[i]) + 10*r(x[i-1]) + r(x[i-2]) )

        yy[i] = a/d * yy[i-1] + b/d * yy[i-2] + c/d
    return yy

def numerovEigVal(prMn, prMx, q, r, x0, y0, xN, yN, y1, dx, nodes, mxItr):
    # Arguments:
    #   prMn, prMx ==> lower and upper bounds of eigenvalue
    #   p ==> User defined function with parameter 'lb', in the form p(lb, x)
    #   q ==> User defined function
    #   r ==> User defined function
    #   x0, y0 ==> left boundary condition
    #   xN, yN ==> right boundary condition
    #   y1 ==> estimate of solution at the next point after left boundary
    #   dx ==> increment along x-axis
    #   nodes ==> Number of nodes of the eigenvalue
    #   mxItr ==> Number maximum allowed iterations
    #
    # Returns:
    #   pr ==> eigenvalue
```

```
#   x ==> x-array of solution
#   yy ==> y-array of solution
#
N = int( (xN - x0)/ dx )
dx = (xN - x0)/N
x = [x0+i*dx for i in range(N+1)]
y = [0 for i in range(N+1)]
y[0], y[1], y[N] = y0, y1, yN

itr = 0
while abs(prMx - prMn) > 1e-6 and itr < mxItr:
    pr = 0.5*(prMn + prMx)
    yy = propNumerov(pr, q, r, x, y, dx)

    cnt = 0
    for i in range(1, N-2):
        if yy[i]*yy[i+1] < 0:
            cnt += 1
    if cnt > nodes:
        prMx = pr
    elif cnt < nodes:
        prMn = pr
    else:
        if yy[N-1] > yN:
            prMn = pr
        elif yy[N-1] < yN:
            prMx = pr
    itr += 1
if itr < mxItr:
    return pr, x, yy
else:
    return None, None, None
```

To understand the use of Python function numerovEigVal( ) , let us find the first four eigen values and corresponding eigen functions of the following eigen value equation

numerovEigVal.py

$$\frac{d^2y}{dx^2} = -\lambda y \qquad \text{with } y(0) = 0, \; y(2\pi) = 0$$

Python code is given below. Here the bound of π are taken as [0.2, 30].

Example 3.3.2.1—numerovEigValEx01.py

```
from math import *
import matplotlib.pyplot as plt

from numerovEigVal import numerovEigVal

def q(lb, x):
    return -lb

def r(x):
    return 0

stln = ['k', 'k--', 'k-.', 'k:'] # list of line styles
dx = 0.001 # increment along x-axis
mxItr = 100 # maximum number of allowed iteration

lbMn, lbMx = 0.2, 30 # bound of lambda
x0, y0, xN, yN = 0, 0, 2*pi, 0

for nodes in range(4):
    y1 = (-1)**nodes*1e-4

    lb, x, y = numerovEigVal(lbMn, lbMx, q, r, x0, y0, xN, yN, y1,
                             dx, nodes, mxItr)

    plt.plot(x, y, stln[nodes],
             label = r'Eigenfunction for $\lambda = %.3f$' %lb)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc='best', prop={'size':12})
plt.show()
```

First four eigen values and eigen functions those are plotted by the above code is illustrated below.

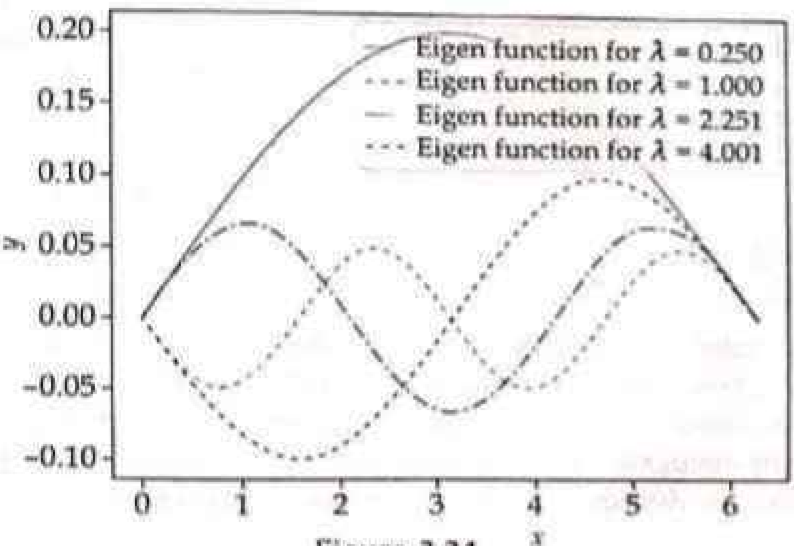

Figure 3.34

3.4 Time Independent Schrödinger Equation

Probably, the most studied eigen value problem in Physics is the time independent Schrödinger equation (TISE). Here, we will solve Schrödinger equation numerically. The time independent Schrödinger equation for a particle of mass m and energy E kept in potential $V(x)$ is given by

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

$$\frac{d^2\psi}{dx^2} = \frac{2m}{\hbar^2}[V(x) - E]\psi(x) \quad \text{... (3.36)}$$

where $2\pi\hbar$ is the Plank's constant.

In cases where $V(x) \to 0$ as $x \to \pm\infty$, a reasonable boundary condition to impose is

$$\psi(x) \to 0 \text{ as } x \to \pm\infty \quad \text{... (3.37)}$$

Though, the boundary conditions at $x \pm \infty$ are not really that resonable when we are trying to solve the system numerically. We have to approximate some finite boundary conditions. Assuming $V(x)$ is centered at origin, we will choose the finite domain $-a \le x \le a$, with a is chosen such that $V(\pm a)$ is sufficiently large to ensure that $\psi(\pm a) \approx 0$.

Thus, the finite boundary conditions to be satisfied are

$$\psi(-a) \approx 0, \quad \psi(a) \approx 0 \quad \text{... (3.38)}$$

In this way, the infinite boundary conditions are truncated to finite boundary conditions.

This process creates numerical instability in solving the TISE.

- **Instability in shooting method to solve TISE**

Exact reason of instability during the solution of TISE by finite difference shooting method, is still under active reasearch. We can argue about some difficulties created in solving TISE by finite difference shooting method.

When, the theoretical boundary condition is finite, then there is no problem. Let us consider the following boundary conditions

$$y(x_0) = y_0 \text{ and } y(x_N) = y_N$$

Starting with the initial condition $y(x_0) = y_0$ and $y(x_0 + h) = y_1$ (y_1 is taken arbitrarily), the eigenvalue E has to be tuned to match the boundary condition $y(x_N) = y_N$. So, the finite difference calculation has the sharp bound $x_0 \le x \le x_N$. Tuning E by bisection shooting method is quite successful for these kind of problems. But when boundary conditions are infinite like

$$y(-\infty) = 0 \text{ and } y(\infty) = 0$$

then problem starts. Numerically, we have to terminate the boundaries at finite values [$-a$, a] satisfying the boundary conditions

$$y(-a) = 0 \text{ and } y(a) = 0$$

The selection of termination points [$-a$, a] is completely arbitrary. We, can choose a quite wide boundary. But, boundary conditions are only be satsfied if the eigenvalue is determined with cent percent accuracy. Calculation of eigenvalue with highly precise algorithm, does not provide cent percent accurate eigenvalue. Thus, the solution near right end boundary shows instability which can be demonstrated by the following figure.

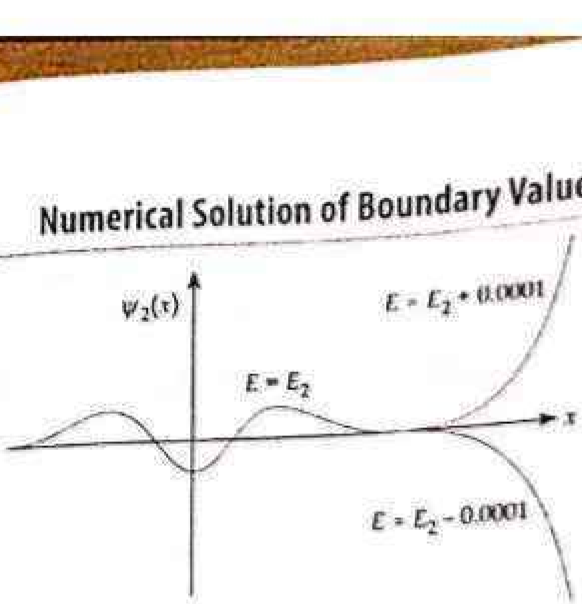

Figure 3.35

Careful observation of the above curve reveals that shooting method with just a 0.001 error in the value E leads to relatively accurate results ultil we approach the right end boundary – here, the shooting solutions tends to grow almost exponentially in magnitude away from x-axis! Shooting method has this limitation, - solution is unstable near right end boundary. There are several other numerical methods those which can handle this instability. But, those are not the shooting method. One of such succesful algorithm is **matching method** with multiple shooting. Due to the limitation of the size of this volume, we do not introduce this method here. We will continue using shooting method to solve TISE.

3.4.1 Numerical solution of TISE by finite-difference shooting method

Time independent Schrödinger equation-3.36 belongs to the the class of Strum-Liouville eigen value problem. Here, the default choice of the mathematical functions of equation-3.28 for equation-3.36 are as follows

$$\psi(x) \leftarrow y(x),\ \ p(x) \leftarrow 0,\ \ q(\lambda, x) \leftarrow \frac{2m}{\hbar^2}[V(x) - E] \text{ and } r(x) \leftarrow 0 \quad \text{... (3.39)}$$

Therefore, the numerical scheme for determining the eigenvalues and eigen functions for TISE remains same as that for equation-3.28. As in equation-3.28, the eigen values and eigen functions of TISE are also determined by two finite difference methods, viz. central-difference and Numerov methods.

- **Solving TISE by central-difference shooting method**

Applying finite-difference shooting method, as discussed in section-3.3, we can solve TISE. From equation-3.32, replacing $p(x_i) = 0$, $r(x_i) = 0$, $q(\lambda, x_i) = \frac{2m}{\hbar^2}[V(x_i) - E]$ for $i = 1, 2, ..., N-1$, the discretized TISE according to central difference method, can be written as follows

$$\psi(x_{i+1}) = 2\left[\frac{m\delta x^2}{\hbar^2}[V(x_i) - E] + 1\right]\psi(x_i) - \psi(x_{i-1}) \quad \text{for } i = 1, 3, ..., N-2$$

$$\Rightarrow \psi(x_i) = 2\left[\frac{m\delta x^2}{\hbar^2}[V(x_{i-1}) - E] + 1\right]\psi(x_{i-1}) - \psi(x_{i-2}) \quad \text{for } i = 2, 3, ..., N-1$$

$$\text{with } \psi(x_0) = \psi_0,\ \psi(x_1) = \psi_1 \text{ and } \psi(x_N) = \psi_N \quad \text{... (3.40)}$$

Here, ψ_1 is determined from the arbitrary local slope α as $\psi_1 = \psi_0 + \alpha h$. Where h is increment along x-axis. Since α is arbitrary, ψ_1 is arbitrarily. Let us consider $\psi_1 = \psi_0 + 10^{-4}$. For that reason, $\{\psi(x_i), i = 1, ..., N-1\}$ will be calculated with an arbitrary scaling. Later, normalizing $\{\psi(x_i), i = 1, ..., N-1\}$, the eigen function can be represented in the standard scaling.

According to the above formulation, the staright forward numerical scheme for the difference propagator for TISE can be represented with the following algorithm.

Algorithm 3.4.1.1—propagator of TISE according to central-difference method

Define the potential function $V(x)$

Get values of E, m, $\hbar$, N

$\delta x \leftarrow \dfrac{x_N - x_0}{N}$

for $i = 0, 1, 2, \ldots, N$

  $x_i \leftarrow x_0 + i\delta x$

$\psi(x_0) \leftarrow \psi_0$, $\psi(x_1) \leftarrow \psi_1$, $\psi(x_N) \leftarrow \psi_N$

for $i = 2, 3, \ldots, N - 1$

  $\psi(x_i) \leftarrow 2\left(1 + \dfrac{m\delta x^2}{\hbar^2}(V(x_{i-1} - E))\right)\psi(x_{i-1}) - \psi(x_{i-2})$

Python function **Psi()** calculates the propagator of TISE according to above algorithm. Alg for determining of eigenvalue and eigenfunction remains same as that of algorithm except a small change. Previously, the upper and lower bound of the internal parameter to be supplied by the user. Here the upper and lower bouunds of the energy eigen taken as the maximum and minimum value of the potential within the range of Introducing this change in the previous algorithm-3.3.1.2, the Python function cntDiff is developed to eigenvaalue and eigenfunction of TISE.

18

Program 3.4.1.1—cntDiffSchr.py

```
def Psi(mhdx2, psi, Vi, E):
    # Arguments:
    #       mhdx2   ==> parameter, m*dx**2/hbar**2
    #       psi     ==>  array of discrete initial wave function
    #       Vi      ==>  array of discrete potential
    #       E       ==>  energy of particle
    #
    # Returns:
    #       psiE ==> array of discrete final wavefunction
    #
    N = len(psi)
    psiE = [psi[i] for i in range(N)]
    for i in range(2, N):
        psiE[i] = 2* ( mhdx2*(Vi[i]-E) + 1)*psiE[i-1] - psiE[i-2]
    return psiE
```

```
def cntDiffSchr(mhdx2, Vi, psi0, psi1, psiN, nodes, mxItr):
    # Arguments:
    #       mhdx2 ==> parameter, m*dx**2/hbar**2
    #       Vi    ==> discrete potential
    #       psi0  ==> left end boundary value of wavefunction
    #       psi1  ==> next after near end boundary value of wavefunction
    #       psiN  ==> right end boundary value of wavefunction
    #     nodes   ==> Number of nodes
    #     mxItr   ==> maximum number of allowed iterations
    #
    #
    # Returns:
    #       E     ==> energy eigen value
    #       psi   ==> array of discrete wavefunction
    #
    N = len(Vi)-1
    Emx = max(Vi) # upper bound of Energy eigenvalue
    Emn = min(Vi) # lower bound of Energy eigenvalue

    psiIn = [0 for i in range(N+1)]
    psiIn[0], psiIn[1], psiIn[N] = psi0, psi1, psiN #

    itr = 0
    while abs(Emx-Emn) > 1e-6 and itr < mxItr:
        E = 0.5*(Emx+Emn) # trial energy value
        psi = Psi(mhdx2, psiIn, Vi, E) # Get wavefunction

        # Node counting method
        cnt = 0
        for i in range(1, N-2):
            if psi[i] * psi[i+1] < 0:
                cnt += 1

        if cnt > nodes:
            Emx = E
        elif cnt < nodes:
            Emn = E
        else:
            if psi[N-1] > psiN:
                Emn = E
            elif psi[N-1] < psiN:
                Emx = E
        itr += 1
    if itr < mxItr:
        return E, psi
    else:
        return None, None
```

Using the Python function cntDiffSchr( ) , the eigenfunction of Schrödinger equ... obtained in arbitrary scaling. Now to transform this eigen function in a uniform... normalization is to be done.

- **Normalization of eigen function**

The eigen function $\psi(x)$ could be normalized using the relation-3.31. The normalized function is determined according to the following mathematical steps.

$$\int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1$$

$$\Rightarrow A^2 \int_{-\infty}^{\infty} |\phi(x)|^2 dx = 1 \quad \text{from equation-3.30}$$

$$\Rightarrow A = \frac{1}{\sqrt{\int_{-\infty}^{\infty} |\phi(x)|^2 \, dx}}$$

Therefore equation-3.30 provides

$$\psi(x) = \frac{1}{\sqrt{\int_{-\infty}^{\infty} |\phi(x)|^2 \, dx}} \phi(x) \qquad \text{... (3.41)}$$

Equation-3.41 is the representation of the normalized wavefunction. Since calculated... is discrete function, so Simpson-$\frac{1}{3}$ rule for discrete function* will be used for integr...

Program 3.4.1.2–SchrNorm.py

7

```
# Normalization of discrete wavefunction
from simp13Xdis import simp13Xdis

def psiNorm(psi, dx):
    # Arguments :
    #     psi ==> all eigenstates
    #     dx ==> space increment
    #
    # Returns :
    #     nrmPsi ==> normalized eigenstates
    #
    N = len(psi)
    psi2 = [psi[i]**2 for i in range(N)]
    psiMod2 = simp13Xdis(dx, psi2)
    nrmPsi = [ psi[i]/psiMod2**0.5 for i in range(N)]
    return nrmPsi
```

(a) It is interesting as well as necessary to visualize, how central-difference shooting meth... **converges towards the exact eigenvalue and eigenfunction.** For this demonstration... seperate Python code is provided below. The propagator is determined according to ...

* Please consult authors' previous book, *Physics in Laboratory* for Sem-III.

0 2 4 6 8 10 12 14 16

r

Figure 3.41

(f) The problem is to solve the **radial wave function of Hydrogen atom**. The radial Schr... equation for a particle in central potential $V(r)$ is given by the below equation

$$\frac{d^2\psi_R}{dr^2} = -\frac{2}{r}\frac{d\psi_R}{dr} - \frac{2m}{\hbar^2}(E - V(r))\psi_R \qquad -(3.47)$$

ready

The central potential for Hydrogen atom is the Coulomb potential.

$$V(r) = \frac{e^2}{4\pi\varepsilon_0 r}$$

Equation-3.47 contains both the functions $p()$ and $q()$ of the Srtum-Liouville eqn ... equation-3.28. Since the Python function **cntDiffSchr( )** do not contain the function ... it can not be used here. Here we have to use the general Python function CntDiff... for solving eigen value equation-3.28. The Python function is as follows. (The code ... explanatory).

Example 3.4.1.6—cntDiffSchrH2Rad.py

```
from math import *
import matplotlib.pyplot as plt
from CntDiffEigVal import CntDiffEigVal
from SchrNorm import psiNorm
hbar, m = 0.1, 1 # constants
e2 = 0.2 # constant
mh2 = 2*m/hbar**2
# potential
def V(e2, r):
    return -e2/r
# function p() in Srtum-Liouville eigenvalue equation
def P(r):
    return -2/r
```

```
# function q() in Srtum-Liouville eigenvalue equation
def Q(E, r):
    return -mh2*( E - V(e2, r) )
# function r() in Srtum-Liouville eigenvalue equation
def R(r):
    return 0
stln = ['k', 'k--', 'k:', 'k-.'] # array line styles
dr = 0.01 # increment in radial direction
mxItr = 100 # maximum allowed iterations
# boundary conditions
r0, psi0, rN, psiN = 1e-6, 0, [0.7, 1.2, 2.2, 3.2], 0
# first four eigenfunctions
for nodes in range(4):
    Emn, Emx = V(e2, r0), V(e2, rN[nodes])
    psi1 = psi0 + (-1)**nodes * 1e-4
    E, r, psi = CntDiffEigVal(Emn, Emx, P, Q, R, r0, psi0, rN[nodes],
                              psiN, psi1, dr, nodes, mxItr)
    if E != None: # if solution is obtained
        psi = psiNorm(psi, dr) # normalization
        rPsi = [ r[i]*psi[i] for i in range(len(r)) ]
        plt.plot(r, rPsi, stln[nodes],
                 label = r'$r\psi_%d(r)$ for $E_%d$ = %.3f' %
                 (nodes+1,nodes+1,E))
        plt.xlabel('r')
        plt.legend(loc='best', prop={'size':12})
plt.show()
```

The plots of modified eigen functions $r\psi_R(r)$ for different energy eigen values are shown below.

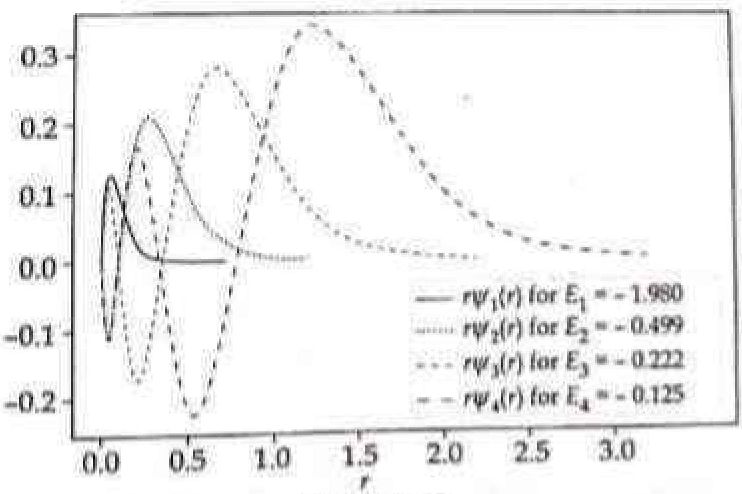

Figure 3.42

8

The WKB approximated energy eigen values are comparable with the numerical calcula

(e) The s-wave radial equation for a particle with mass m and energy E moving in a cen potential $V(r)$ is given by

$$\frac{d^2\psi(r)}{dr^2} = \frac{2m}{\hbar^2}[V(r) - E]\psi(r) \qquad \text{... (3.46)}$$

Which is same as that of equation-3.36, except the coordinate of representation of t problem here is spherical polar. Thus, we can use the Python function cntDiffSchr() solve this problem numerically. A simple central potential

$$V(r) = -\frac{e^2}{r}e^{-\frac{r}{a}} \quad r > 0$$

is taken for investigation. In present case, the constants of the above case are taken as 7.5, $\hbar$ = 0.1, m = 1. The ranges in radial direction are considered as $[10^{-4}, 7]$ and $[10^{-4},$ for determination the first two eigen functions. Obviously, the left boundary could not taken as zero as potential diverges at that point. The left boundary condition is take $\psi(0.0001) = 0$. We will calculate first two eigen values and eigen functions. The right bound conditions for these eigen functions are taken as

$\psi_0(7.0) = 0$, $\psi_1(16.1) = 0$

The Python code is given below

Example 3.4.1.5—cntDiffSchrCntPot01.py

```
from cntDiffSchr import cntDiffSchr
from SchrNorm import psiNorm
import matplotlib.pyplot as plt
from math import exp

def V(e2, r):
    a = 7.5 # constant
    return -e2/r*exp(-r/a)

# User defined parameters
hbar, m = 0.1, 1
R0, RN = 0.0001, [7.0, 16.1] # left and two right boundaries
dr = 0.005
mxItr = 100 # maximum number of iterations
psi0, psiN = 0, 0 # Psi(x_0), Psi(x_N)
e2 = 0.01

stln = ['k--', 'k:', 'k.-' ]

for nodes in range(2):

    N = int( (RN[nodes] - R0)/dr )
    dr = (RN[nodes] - R0)/N
    mhdr2 = m*dr**2/hbar**2
    r = [ R0 + i*dr for i in range(N+1)] # discretization of space
    Vi = [ V(e2, r[i]) for i in range(N+1) ] # discretization of potential

    psi1 = (-1)**nodes*1e-4
    E, psi = cntDiffSchr(mhdr2, Vi, psi0, psi1, psiN, nodes, mxItr)
    if E != None:
        psi = psiNorm(psi, dr)

        ## Plotting
        plt.plot(r, psi, stln[nodes],label=r'E= %.5f $\psi_%d(r)$' %(E, nodes))
plt.ylim(-0.8, 0.8)
plt.plot(r, Vi, 'b', label='Potential')
xax = [0 for i in range(N+1)]
plt.plot(r, xax, 'k')
plt.legend(loc='best', prop={'size':12})
plt.xlabel('r')
plt.show()
```

The eigenfunctions are plotted along with the eigenvalues.

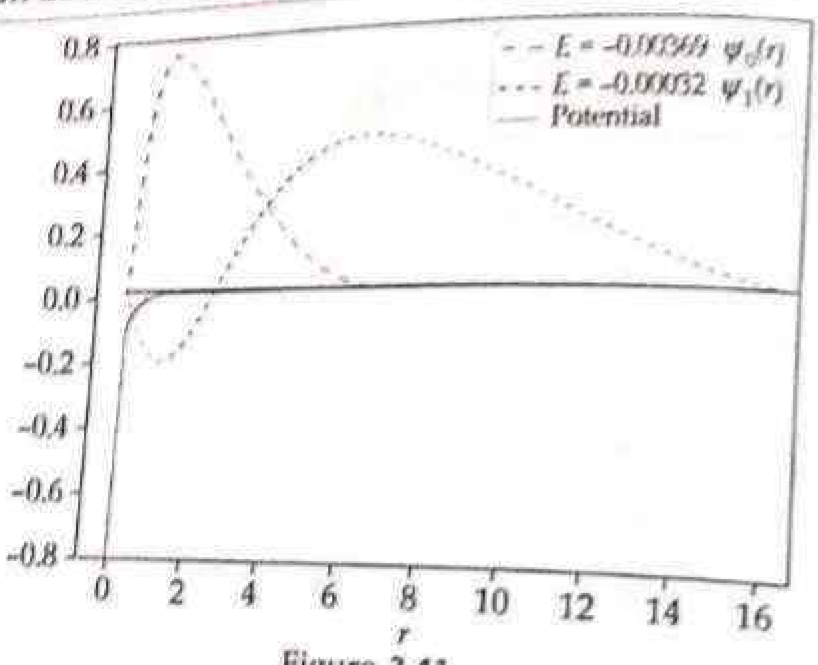

Figure 3.41

(f) The problem is to solve the **radial wave function of Hydrogen atom**. The radial equation for a particle in central potential

The energy eigen values can be determined analytically as

$$E_n = -\frac{m}{2\hbar^2}\left(\frac{e^2}{4\pi\varepsilon_0}\right)^2\frac{1}{n^2} \qquad \text{... (3.48)}$$

In our present problem the constants are taken as $\hbar = 0.1$, $m = 1$, $\frac{e^2}{4\pi\varepsilon_0} = 0.2$. Substi… these values the energy eigen values are given by

$$E_n = -\frac{2}{n^2}$$

Therefore, energy eigen values are $E_1 = -2$, $E_2 = -0.5$, $E_3 = -0.222$ ... Theoretica… numerical values are quite close.

■ Solving TISE by Numerov shooting method

Numerov method for TISE is simply reuse of the equation-3.35 with proper replacem… terms. Modifying the calculation of equation-3.35 replacing terms as $q(\lambda, x_i) \leftarrow \frac{2m}{\hbar^2}[V(x_i)$… and $r(x_i) \leftarrow 0$ for $i = 1, 2, ..., N-1$, the propagator of TISE according to the Numerov algori… can be wrtten as follows.

$$\psi(x_i) = \frac{a}{d}\psi(x_{i-1}) + \frac{b}{d}\psi(x_{i-2}) \quad \text{for } i = 2, 3, ..., N-1$$

$$\text{where } a = 2\left[1 + \frac{5\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_{i-1}) - E]\right],\ b = -\left[1 - \frac{\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_{i-2}) - E]\right]$$

$$\text{and } \quad d = \left[1 - \frac{\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_i) - E]\right] \qquad \text{... (3.49)}$$

Here, m and E are the mass and energy of the paticle. The potential in Cartesian coordinate is given by $V(x)$. The constant $2\pi\hbar$ is the Plank's constant. The increment of space discretization is δx.

The numerical steps corresponding to the above mathematical steps are summerized in the below algorithm.

Algorithm 3.4.1.2—Solving TISE by Numerov method

Define the potential function $V(x)$

Get values of E, m, $\hbar$, N

$\delta x \leftarrow \frac{x_N - x_0}{N}$

for $i = 0, 1, 2, ..., N$

  $x_i \leftarrow x_0 + i\delta x$

$\psi(x_0) \leftarrow \psi_0$, $\psi(x_1) \leftarrow \psi_1$, $\psi(x_N) \leftarrow \psi_N$

for $i = 2, 3, ..., N-1$

  $a \leftarrow 2\left[1 + \frac{5\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_{i-1}) - E]\right]$

  $b \leftarrow -\left[1 - \frac{\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_{i-2}) - E]\right]$

  $d \leftarrow \left[1 - \frac{\delta x^2}{12}\frac{2m}{\hbar^2}[V(x_i) - E]\right]$

  $\psi(x_i) \leftarrow \frac{a}{d}\psi(x_{i-1}) + \frac{b}{d}\psi(x_{i-2})$

Using the above algorithm, the Python function **Psi( )** is developed, which returns the TISE propagator according to Numerov method. Using the propagator, the energy eigenvalues and the eigenfunctions are determined by the Python function **numerovSchr( )**. Like previous, the node counting bisection shooting method is adopted for determing the desired energy eigenvalues. As soon as the eigenvalue is estimated by node counting method, bisection method is applied for satisfying the boundary conditions. Both of the Python functions Psi( ) and numerovSchr( ) remian in the Python script numerovSchr.py .

Program 3.4.1.5 – numerovSchr.py

```
# Finding the eigen value and eigen function of time independent
# Schrodinger equation by Numerov method. The
# eigen value is determined by node counting shooting method.

# Wave function
def Psi(mhdx2, psi, Vi, E):
    # Arguments:
    #   mhdx2 ==> parameter, m*dx**2/hbar**2
    #   psi   ==> array of discrete initial wave function
    #   Vi    ==> array of discrete potential
    #   E     ==> energy of particle
    #
    # Returns:
    #   psiE  ==> array of discrete final wavefunction
    #
    N = len(psi)
    psiE = [psi[i] for i in range(N)]
    P = [mhdx2*(Vi[i] - E) for i in range(N)]
    for i in range(2, N):
        d = 1 - 1/12 * P[i]
        a = 2*(1 + 5/12 * P[i-1])
        b = -(1 - 1/12 * P[i-2])
        psiE[i] = a/d * psiE[i-1] + b/d * psiE[i-2]
    return psiE

def numerovSchr(mhdx2, Vi, psi0, psi1, psiN, nodes, mxItr):
    # Arguments:
    #   mhdx2 ==> parameter, m*dx**2/hbar**2
    #   V     ==> discrete potential
    #   prV   ==> parameter of potential function
    #   psi0  ==> left boundary value of wavefunction
    #   psi1  ==> next after left boundary value of wavefunction
    #   nodes ==> Number of nodes
    #   mxItr ==> maximum number of allowed iterations
```

```
#
# Returns:
#   E      ==> energy eigen value
#   psi    ==> array of discrete wavefunction
N = len(Vi)-1
Emx = max(Vi)
Emn = min(Vi)
psiIn = [0 for i in range(N+1)]
psiIn[0], psiIn[1], psiIn[N] = psi0, psi1, psiN
itr = 0
while abs(Emx-Emn) > 1e-6 and itr < mxItr:
    E = 0.5*(Emx+Emn) # trial energy value
    psi = Psi(mhdx2, psiIn, Vi, E) # Get wavefunction
    # Node counting method
    cnt = 0
    for i in range(1, N-2):
        if psi[i] * psi[i+1] < 0:
            cnt += 1
    if cnt > nodes:
        Emx = E
    elif cnt < nodes:
        Emn = E
    else:
        if psi[N-1] > psiN:
            Emn = E
        elif psi[N-1] < psiN:
            Emx = E
    itr += 1
if itr < mxItr:
    return E, psi
else:
    return None, None
```

The following TISE problems are solved using the Python functions `Psi()` and `numerovSchr( )` .

Numerical values ...

(b) In this problem, the potential is a central potential in spherical polar coord... the following form.

$$V(r) = \frac{1}{2}kr^2 + \frac{1}{3}br^3 \quad k \text{ and } b \text{ are constants.}$$

Here, we will solve the s-wave radial equation-3.46 only. The below Python pro... the above equation numerically by Numerov method.

Example 3.4.1.8—numerovSchrCntPot02.py

```
# Solving Radial equation under central force field
#
from numerovSchr import numerovSchr
from SchrNorm import psiNorm
import matplotlib.pyplot as plt
from math import exp

def V(pr, r):
    k, b = pr
    return 0.5*k*r**2+ (1/3)*b*r**3

# User defined parameters
hbar, m = 0.1, 1

dr = 0.005
mxItr = 100 # maximum number of iterations
psi0, psiN = 0, 0 # Psi(x_0), Psi(x_N)
k = 1

R0, RN = [0, 0, 0, 0], [1.25, 1.5, 1.7, 1.9] # Left and right boundaries
psi0, psiN = 0, 0
for nodes in range(4):

    N = int( (RN[nodes] - R0[nodes])/dr )
    dr = (RN[nodes] - R0[nodes])/N
    mhdx2 = 2*m*dr**2/hbar**2
    r = [ R0[nodes] + i*dr for i in range(N+1)]
    Vi = [ V([1, 0.01], r[i]) for i in range(N+1) ]

    psi1 = (-1)**nodes*1e-4

    E, psi = numerovSchr(mhdx2, Vi, psi0, psi1, psiN, nodes, mxItr)
    if E != None:
        psi = psiNorm(psi, dr)

        ## Plotting
        plt.plot(r, psi, label=r'E= %.4f $\psi_%d(x)$' %(E, nodes))

xax = [0 for i in range(N+1)]
plt.plot(r, xax, 'k')
plt.plot(r, Vi, 'k--', label='Potential')
plt.legend(loc='best', prop={'size':12})
plt.xlabel('r')
plt.show()
```

Plots of eigen functions and eigen values those obtained from above code are dem
in below figure.

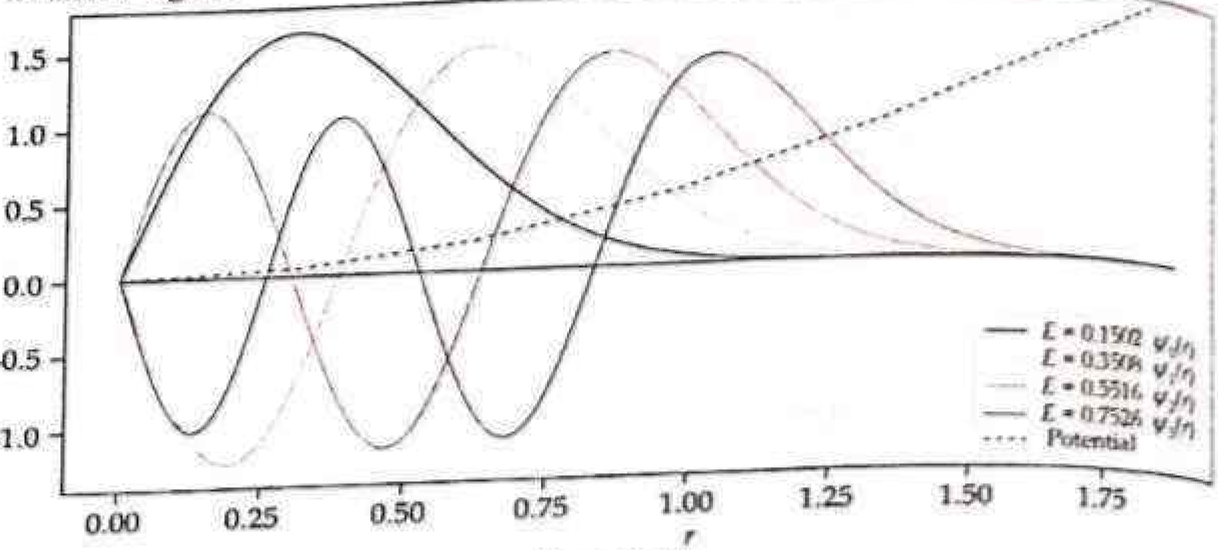

Figure 3.44

(c) Another s-wave radial Schrödinger equation to be soved for a particle in the pote
given by

$$V(r) = D[e^{-2\alpha r} - e^{-\alpha r}]$$

TISE is solved numerically by the following Python code for the above potential. The va
of the constants for the present problem are taken as $m = 1$, $\hbar = 0.1$, $D = 1$, $\alpha = 3$.

Example 3.4.1.9—numerovSchrCntPot03.py

```
# Solving Radial equation under central force field
#
from numerovSchr import numerovSchr
from SchrNorm import psiNorm
import matplotlib.pyplot as plt
from math import exp
# potential, D, al = 1, 3
def V(pr, r):
    D, al = pr
    return D*( exp(-2*al*r) - exp(-al*r) )
hbar, m = 0.1, 1 # constants
dr = 0.005
mxItr = 100 # maximum number of iterations
psi0, psiN = 0, 0 # Psi(x_0), Psi(x_N)
R0, RN = [0, 0, 0], [1.8, 3.5, 6.5] # Left and right boundaries
psi0, psiN = 0, 0 # boundary values
for nodes in range(3):
```

```
N = int( (RN[nodes] - R0[nodes])/dr )
dr = (RN[nodes] - R0[nodes])/N
mhdx2 = 2*m*dr**2/hbar**2 # constant
r = [ R0[nodes] + i*dr for i in range(N+1)] # space discretization
Vi = [ V([1, 3], r[i]) for i in range(N+1) ] # potential discretization
psi1 = (-1)**nodes*1e-4 # value of eigenfunction just after boundary
E, psi = numerovSchr(mhdx2, Vi, psi0, psi1, psiN, nodes, mxItr)
if E != None:
        psi = psiNorm(psi, dr) # normalization
        ## Plotting
        plt.plot(r, psi, label=r'E= %.4f $\psi_%d(r)$' %(E, nodes))
xax = [0 for i in range(N+1)]
plt.plot(r, xax, 'k')
plt.plot(r, Vi, 'k--', label='Potential')
plt.legend(loc='best', prop={'size':12})
plt.xlabel('r')
plt.show()
```

First three eigenfunctions along with the eigenvalues are plotted in the following figure.

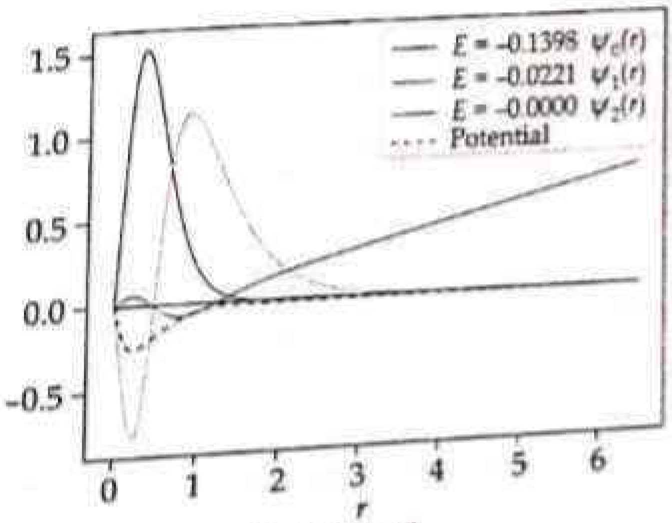

Figure 3.45