Numerical Solution of Schrödinger Equation: Automatic Generation of Wave Functions and Densities that can be used as input to Machine Learning Methods

M.Sc. Thesis

By

BHUSAN JYOTI BORAH
(2203131013)

DEPARTMENT OF CHEMISTRY

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May, 2024

Numerical Solution of Schrödinger Equation: Automatic Generation of Wave Functions and Densities that can be used as input to Machine Learning Methods

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF SCIENCE

by

BHUSAN JYOTI BORAH

DEPARTMENT OF CHEMISTRY

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May, 2024

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work is being presented in the thesis entitled **"Numerical Solution of Schrödinger Equation: Automatic Generation of Wave Functions and Densities that can be used input to Machine Learnig Methods"** and partial fulfillment of the requiredments for the award of the degree of **MASTER OF SCIENCE** and submitted in the **DEPARTMENT OF CHEMISTRY, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from July 2023 to April, 2024 under the supervision of **Prof. Satya S. Bulusu**, Professor, Indian Institute of Technology Indore, India. I have adequately cited and referenced the original source.

The matter presented in this report has not been submitted by me for the award of any other degree of this or any other institute.

Bhusan Jyoti Borah
06.05.2024

Signature of the Student with date
Bhusan Jyoti Borah

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

06.05.2024
Signature of the Supervisor with date
Prof. Satya S. Bulusu

Bhusan Jyoti Borah has successfully given his M.Sc. Oral Examination held on 08.05.2024.

Signature of Supervisor of M.Sc. thesis
Prof. Satya S. Bulusu
Date: 10.05.2024

Convener, DPGC
Dr. Umesh A. Kshirsagar
Date:

ACKNOWLEDGEMENTS

I would like to thank my supervisor **Prof. Satya S. Bulusu**(Professor, Department of Chemistry, IIT Indore) for his continuous support, conviction, encouragement and invaluable advice during my lab work. I have been fortunate to have a supervisor who is deeply invested my work, consistentely addressing my questions and concerns promptly. I would also like to thank my PSPC members **Prof. Tushar Kanti Mukharjee** and **Dr. Tridib Kumar Sarma**.

I would like to acknowledge this assistance given to me by lab senior **Ms. Aparna Gangwar**. She helped me a lot by giving me guidance and encouragement whenever I was stuck in any problem. Their contribution was precious to me in this project. I feel lucky to work under her guidance. I also thank my other lab members for their valuable suggestions and help.

I would like to thank all my classmates and friends. Finally, I would like to thank my parents who have always been a constant source of support for me throughout my life.

Bhusan Jyoti Borah
(2203131013)

Abstract

I know that the Schrödinger equation is solvable for simple systems like particle in a box, harmonic oscillator and hydrogen atom. But, solution of the Schrödinger equation for complex systems is a non-trivial problem. Our main objective in the thesis is to solve the Schrödinger equation by using numerical methods for One and Two dimensional systems. In our project, I have implemented two different type numerical methods one is Numerov and the other one is Finite-difference method. Firstly, I employed both these methods to solve the Schrödinger equation for Harmonic potential well. I have compared the results with analytical solution and it is found that the our results for Harmonic potential well is quite accurate. So in the next we employed the above numerical methods for the Gaussian potential well to get the eigenvalues and eigenvectors. This project has taken as a prototype for typical machine learning problems. I have automated this scheme to generate wave functions and densities for 10000 random potentials.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

$\boldsymbol{\psi}$	Psi
$\boldsymbol{\hbar}$	Hbar
$\boldsymbol{\alpha}$	Alpha
$\boldsymbol{\beta}$	Beta
$\boldsymbol{\gamma}$	Gamma
$\boldsymbol{\delta}$	Delta
$\boldsymbol{\theta}$	Theta

ACRONYMS

ML	Machine Learning
1D	One Dimensional
2D	Two Dimensional
NM	Numerov Method
FDM	Finite Difference Method
GS	Ground State
FES	First Excited State
SES	Second Excited State
TES	Third Excited State
ANN	Artificial Neural Network

Chapter 1

Introduction

1.1 Overview

The Schrödinger equation[7] is a one basic differential equation in quantum mechanics known as the wave equation, describing behaviour of subatomic particles like electrons. In chemistry, the Schrödinger equation have various application in computational chemistry, chemical bonding, spectroscopy and quantum chemistry. There are two forms of Schrödinger equation such as time-dependent and time-independent. It play crucial role in understanding the evolves and behaviour at atomic and molecular level. We present the time-independent Schrödinger equation that does not contain time as a variable. Solution of time-independent Schrödinger equation is gives stationary state wave functions and corresponding allowed energy level for a systems. So, the time-independent Schrödinger equation confined a quantum particle is,

$$-\frac{\hbar^2}{2m}\nabla^2\psi + v\psi = E\psi \tag{1.1}$$

Where ∇^2 is given by,
For 1D,

$$\nabla^2 = \frac{d^2}{dx^2} \text{ With, } \psi(-\infty) = \psi(\infty) = 0$$

For 2D,

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \text{ With, } \psi(x, \pm\infty) = 0, \ -\infty < x < \infty$$

$$\text{and, } \psi(\pm\infty, y) = 0, \ -\infty < y < \infty$$

As we know that the solving Schrödinger equation for complex system is a non-trivial problem. So, we can used approximation methods such as numerical, variational method.... etc.

In this work, we used Numerical methods to solve the time-independent Schrödinger equation. We will disclose two powerful numerical technique for solving differential equations such as Numerov[7, 2]

and Finite-difference methods[10] applied it to time-independent Schrödinger equation. Using numerical methods, we calculated eigenvalues and eigenvectors for various potential like harmonic, gaussian and double-well potential. The parameters needed for the solution of Schrödinger equation using Numerov and Finite-difference method will be in the respective sections. We found that the Numerov method is more specialized and excel in quantum mechanics, where it offer high accuracy.

1.2 Motivation

1. To make various potentials wave functions and densities data easily available for ML methods.

2. To enhance the accuracy, effectiveness and efficiency of eigenvalue and eigenvectors calculation.

3. To decrease the computational cost and complexity.

1.3 Objective

The main aim of the thesis is numerical solve time-independent Schrödinger equation for One and Two dimensional systems. In the study, we use the Numerov and Finite-difference numerical method for solve the time-independent Schrödinger equation. We consider time-independent Schrödinger equation for gaussian potential well. Since, the time-independent Schrödinger equation in gaussian potential well no genelalised analytical[8] solution. We use the Numerov and Finite-difference method for numerically solve to calculate eigenvalues and eigenvectors.

So, Gaussian potential formula is:
For 1D,

$$v(x) = -\sum_{i=1}^{3} \alpha_i e^{-\frac{(x-\beta_i)^2}{2\gamma_i^2}}$$

For 2D,

$$v(x,y) = \sum_{i=1}^{3} \alpha_i e^{-\frac{1}{2}\left(\frac{(x-\beta_i)^2}{\gamma_i^2}+\frac{(y-\delta_i)^2}{\theta_i^2}\right)}$$

1.4 Organization of the Thesis

The work done in the present thesis is organised in four chapters. The present chapter gives brief overview about the Schrödinger equation and Numerical methods. Further it discusses the motivation towards numerical solution of Schrödinger equation. This

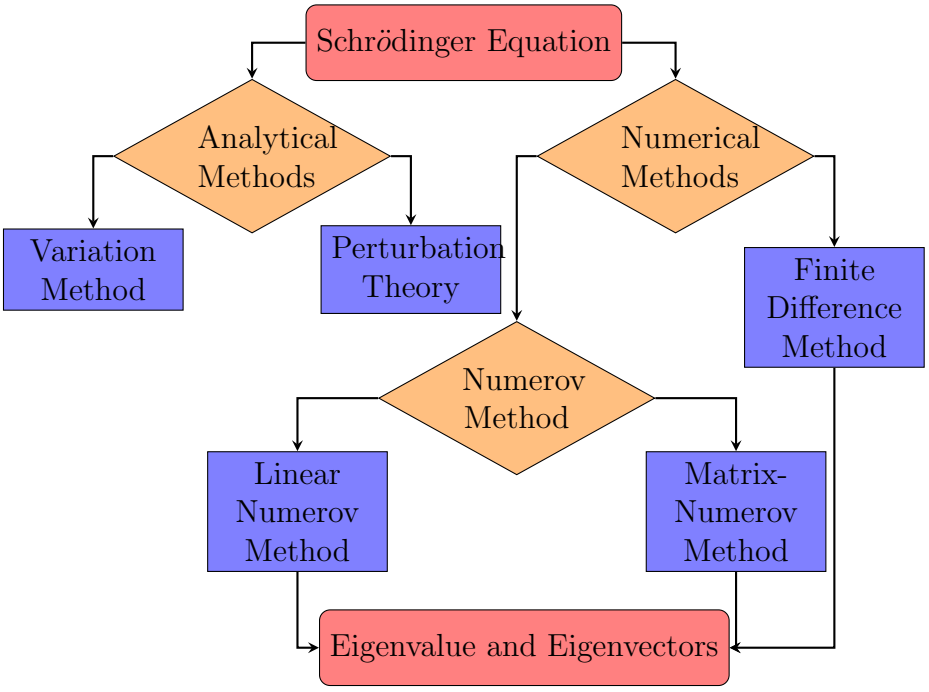
chapter also discuss the different types of numerical methods such as Numerov and Finite-difference method.

- **Chapter 2,** presents the theory and methods behind the work.
- **Chapter 3,** sumarises the work done in the thesis.
- **Chapter 4,** presents the conclusion of whole work and describe the future scope.

Chapter 2

Theory and Methods

There are various methods available to solve the Schrödinger equation. Here are some key methods,



I. Analytical Method: An Analytical solution involves framing the problem in a well-understood form and calculating the exact solution.

II. Numerical Method: A Numerical solution is an approximation of the solution of a mathematical equation, often used where Analytical solution are complex to find.

We used two Numerical methods to solved Schrödinger equation for One and Two dimensional systems.

2.1 One-dimensional System

2.1.1 Numerov Method

The Numerov is a numerical method used to solve second-order linear differential equations in which first order term does not appear. The time-independent Schrödinger does not involves first derivative. So, Numerov method is a suitable algorithm for the Quantum mechanics.

From Schrodinger equation,

$$\frac{d^2\psi(x)}{dx^2} + K^2(x)\psi(x) = 0 \tag{2.1}$$

Where,

$$K^2(x) = \frac{2m}{\hbar^2}[E - V(x)] \tag{2.2}$$

From Numerov method,

$$\psi''(x) = f(x) \cdot \psi(x) \tag{2.3}$$

Let initial condition is,

$$\psi(x_n) = \psi_n$$
$$\psi'(x_n) = \psi'_n$$

Taylor series of expansion of $\psi(x_k)$ at $x_k \pm s$ is,

$$=> \psi(x_n \pm s) = \psi(x_n) \pm s\psi'(x_n) + \frac{s^2}{2}\psi''(x_n) \pm \frac{s^3}{6}\psi'''(x_n) + \frac{s^4}{24}\psi''''(x_n) \tag{2.4}$$

Where, s is a arbitrarily small quantity.
On adding $\psi(x_n \pm s)$ -

$$=> \frac{\psi(x_n + s) - 2\psi(x_n) + \psi(x_n - s)}{s^2} = (1 + \frac{s^2}{12}\frac{d^2}{dx^2}) \times \psi''(x_n) \tag{2.5}$$

$$=> \frac{\psi(x_n + s) - 2\psi(x_n) + \psi(x_n - s)}{s^2} = -K^2(x_n) \times \psi(x_n) - \frac{s^2}{12} \times [\frac{K^2(x_n + s)\psi(x_n + s) - 2K^2(x_n)\psi(x_n) + K^2(x_n - s)\psi(x_n - s)}{s^2}] \tag{2.6}$$

Regrouping the terms,

$$=> \psi(x_n + s) = \frac{2[1 - \frac{5s^2}{12}K^2(x_n)]\psi(x_n) - [1 + \frac{s^2}{12}K^2(x_n - s)]\psi(x_n - s)}{[1 + \frac{s^2}{12}K^2(x_n + s)]} \tag{2.7}$$

For eigenvalue calculation,

Bisection Method

This method[11] is primarily used for finding eigenvalue based on the iterative methods value property.

Let us consider f(x) is continuous function lies between a and b, with f(a) is (-)ve and f(b) is (+)ve.

Then, the approximation of eigenvalue is intermediate value,

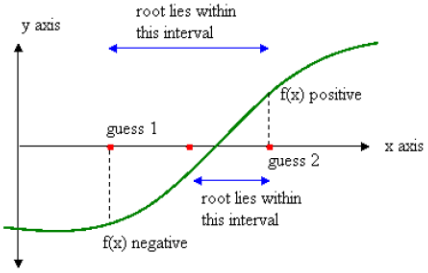


Figure 2.1: Graphical representation of Bisection Method.

Then, the first approximation of the root,

$$\therefore x_1 = \frac{a+b}{2}$$

If $f(x_1) = 0$, then x_1 is root of f(x)=0.

Otherwise, the root lies between a and x_1 or x_1 and b depending on $f(x_1)$ is positive or negative.

Then, we bisect the interval as before and continue the process until the root is found.

2.1.2 Finite-difference Method

The Finite-difference methods are numerical technique used for solve differential equations by approximating derivatives with finite differences. This allows for the differential equation to be transformed into discrete wave functions,

From Schrödinger equation,

$$\frac{d^2\psi(x)}{dx^2} + K^2(x)\psi(x) = 0 \tag{2.8}$$

where,

$$K^2(x) = \frac{2m}{\hbar^2}[E - V(x)] \tag{2.9}$$

Then, on putting x=yL to make equation dimensionless quantity,

$$-\frac{1}{2}\frac{d^2\psi(y)}{dy^2} + L^2 v(y)\psi(y) = L^2 E\psi(y) \tag{2.10}$$

Where, L is the length of bounded.

The second derivative is approximated as,

$$\frac{d^2\psi(x)}{dx^2} = \frac{\psi_{i+1} - 2\psi_i + \psi i - 1}{(\Delta x)^2} \tag{2.11}$$

Here, Δx = Spacing between the point.

After some manipulation,

$$-\frac{1}{2\Delta y^2}\psi_{i+1} + (\frac{1}{\Delta y^2} + L^2 v_i)\psi_i - \frac{1}{2\Delta y^2}\psi_{i-1} = L^2 E\psi_i \tag{2.12}$$

Now, the One-dimensional time independent Schrödinger equation can be written matrix equation i=1 to i=N,

$$=> \begin{bmatrix} \frac{1}{\Delta y^2} + L^2 v_1 & -\frac{1}{2\Delta y^2} & 0 & 0 & \dots \\ -\frac{1}{2\Delta y^2} & \frac{1}{\Delta y^2} + L^2 v_2 & -\frac{1}{2\Delta y^2} & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots & -\frac{1}{2\Delta y^2} \\ \dots & 0 & 0 & -\frac{1}{2\Delta y^2} & \frac{1}{\Delta y^2} + L^2 v_{N-1} \end{bmatrix}$$

$$\times \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-1} \end{bmatrix} = L^2 E \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-1} \end{bmatrix} \tag{2.13}$$

The matrix problem solved using computer, and we get eigenvalue and eigenvectors.
For example, i=1, 2 and 3

$$=> -\frac{1}{2\Delta y^2}\psi_2 + (\frac{1}{\Delta y^2} + L^2 v_1)\psi_1 - \frac{1}{2\Delta y^2}\psi_0 = L^2 E\psi_1 \tag{2.14}$$

$$=> -\frac{1}{2\Delta y^2}\psi_3 + (\frac{1}{\Delta y^2} + L^2 v_2)\psi_2 - \frac{1}{2\Delta y^2}\psi_1 = L^2 E\psi_2 \tag{2.15}$$

$$=> -\frac{1}{2\Delta y^2}\psi_4 + (\frac{1}{\Delta y^2} + L^2 v_3)\psi_3 - \frac{1}{2\Delta y^2}\psi_2 = L^2 E\psi_3 \tag{2.16}$$

The matrix equation i=1 to i=3 is:

$$=> \begin{bmatrix} \frac{1}{\Delta y^2} + L^2 v_1 & -\frac{1}{2\Delta y^2} & 0 \\ -\frac{1}{2\Delta y^2} & \frac{1}{\Delta y^2} + L^2 v_2 & -\frac{1}{2\Delta y^2} \\ 0 & -\frac{1}{2\Delta y^2} & \frac{1}{\Delta y^2} + L^2 v_3 \end{bmatrix} \times \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{bmatrix} = L^2 E \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \end{bmatrix} \tag{2.17}$$

2.2 Two-dimensional System

2.2.1 Numerov Method

2.2.1.1 Linear Numerov Method:

The linear Numerov is a numerical method used to solve second-order ordinary differential equations.

From Schrodinger equation,

$$=> -\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\psi(x,y)+v(x,y)\psi(x,y)=E\psi(x,y) \quad (2.18)$$

We know,

$$\psi(x,y)=\psi(x)\cdot\psi(y) \quad (2.19)$$

$$\text{And,} \quad E=E_x+E_y \quad (2.20)$$

Then, the equation 2.18 is separated into two parts,

$$-\frac{\hbar^2}{2m}\left(\frac{d^2}{dx^2}+v(x)\right)\psi(x)=E_x\psi(x) \quad (2.21)$$

$$=> \frac{d^2\psi(x)}{dx^2}+K^2(x)\psi(x)=0 \quad (2.22)$$

Where,

$$K^2(x)=\frac{2m}{\hbar^2}[E_x-V(x)] \quad (2.23)$$

$$\text{And,} \quad -\frac{\hbar^2}{2m}\left(\frac{d^2}{dy^2}+v(y)\right)\psi(y)=E_y\psi(y) \quad (2.24)$$

$$=> \frac{d^2\psi(y)}{dy^2}+K^2(y)\psi(y)=0 \quad (2.25)$$

Where,

$$K^2(y)=\frac{2m}{\hbar^2}[E_y-V(y)] \quad (2.26)$$

From Numerov method,

$$\psi''(x)=f(x)\cdot\psi(x) \quad (2.27)$$

For x-direction,

Let initial condition is,

$$\psi(x_n)=\psi_n$$

$$\psi'(x_n)=\psi'_n$$

Taylor series of expansion of $\psi(x_n)$ at $x_n \pm s$ is -

$$=> \psi(x_n\pm s)=\psi(x_n)\pm s\psi'(x_n)+\frac{s^2}{2}\psi''(x_n)\pm\frac{s^3}{6}\psi'''(x_n)+\frac{s^4}{24}\psi''''(x_n) \quad (2.28)$$

Where, s is a arbitrarily small quantity.

On adding $\psi(x_n \pm s)$,

$$=> \frac{\psi(x_n+s)-2\psi(x_n)+\psi(x_n-s)}{s^2} = (1+\frac{s^2}{12}\frac{d^2}{dx^2}) \times \psi''(x_n) \tag{2.29}$$

$$=> \frac{\psi(x_n+s)-2\psi(x_n)+\psi(x_n-s)}{s^2} = -K^2(x_n) \times \psi(x_n) - \frac{s^2}{12} \times [\frac{K^2(x_n+s)\psi(x_n+s)-2K^2(x_n)\psi(x_n)+K^2(x_n-s)\psi(x_n-s)}{s^2}] \tag{2.30}$$

Regrouping the terms,

$$=> \psi(x_n+s) = \frac{2[1-\frac{5s^2}{12}K^2(x_n)]\psi(x_n)-[1+\frac{s^2}{12}K^2(x_n-s)]\psi(x_n-s)}{[1+\frac{s^2}{12}K^2(x_n+s)]} \tag{2.31}$$

Similarly,

For y-direction,

$$=> \psi(y_n+s) = \frac{2[1-\frac{5s^2}{12}K^2(y_n)]\psi(y_n)-[1+\frac{s^2}{12}K^2(y_n-s)]\psi(y_n-s)}{[1+\frac{s^2}{12}K^2(y_n+s)]} \tag{2.32}$$

From equation 2.19,

$$\therefore \psi(x_n+s, y_n+s) = \frac{2[1-\frac{5s^2}{12}K^2(x_n)]\psi(x_n)-[1+\frac{s^2}{12}K^2(x_n-s)]\psi(x_n-s)}{[1+\frac{s^2}{12}K^2(x_n+s)]} \times \frac{2[1-\frac{5s^2}{12}K^2(y_n)]\psi(y_n)-[1+\frac{s^2}{12}K^2(y_n-s)]\psi(y_n-s)}{[1+\frac{s^2}{12}K^2(y_n+s)]} \tag{2.33}$$

In Linear Numerov method, we used bisection methods for eigenvalue calculation.

2.2.1.2 Matrix Numerov Method:

The matrix-Numerov is a numerical method used to solve second-order partial differential equations.

From Schrodinger equation,

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\psi(x,y)+v(x,y)\psi(x,y) = E\psi(x,y) \tag{2.34}$$

$$=> \left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right) = -\frac{2m}{\hbar^2}[E-v(x,y)]\psi(x,y)$$

$$=> \left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\psi(x,y) = -f(x,y)\psi(x,y) \tag{2.35}$$

With,

$$f(x,y) = \frac{2m}{\hbar^2}[E - v(x,y)]$$

Adding four Taylor series of expansion of $\psi(x_n, y_n)$ at $x_n \pm s$ and $y_n \pm s$,

$$=> \psi(x_n + s, y_n + s) + \psi(x_n + s, y_n - s) + \psi(x_n - s, y_n + s) +$$

$$\psi(x_n - s, y_n - s) = 4\psi(x_n, y_n) + 2s^2 \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\psi(x_n, y_n) + \frac{s^2}{6}$$

$$\left(\frac{\partial^4}{\partial x^4} + 6\frac{\partial^4}{\partial x^2 \partial y^2} + \frac{\partial^4}{\partial y^4}\right)\psi(x_n, y_n) + 0(s^6) \tag{2.36}$$

Where, s is a arbitrarily small quantity.

Let $\psi(x_n, y_n) = \psi_{x_n, y_n}$ and $\psi(x_n \pm s, y_n \pm s) = \psi_{x_\pm, y_\pm}$

From equation 2.35,

$$-\frac{\partial^2(f\psi)}{\partial x^2} = -\left[\frac{\partial^4\psi}{\partial x^4} + \frac{\partial^4\psi}{\partial x^2 \partial y^2}\right] \tag{2.37}$$

Similarly,

$$-\frac{\partial^2(f\psi)}{\partial y^2} = -\left[\frac{\partial^4\psi}{\partial x^2 \partial^2} + \frac{\partial^4\psi}{\partial y^4}\right] \tag{2.38}$$

Now, we will evaluate $\dfrac{\partial^4}{\partial x^2 \partial y^2}$,

$$\therefore \frac{\partial^2\psi}{\partial x^2 \partial y^2} = \frac{1}{s^4}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \tag{2.39}$$

On putting equation 2.37, 2.38 and 2.39 in 2.36,

$$\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}\begin{bmatrix} \psi_{(x_-,y_-)} \\ \vdots \\ \psi_{(x_+,y_+)} \end{bmatrix} = -\frac{s^2}{2}\begin{bmatrix} 0 & f_{(x_-,y)} & 0 \\ f_{(x,y_-)} & 8f_{(x,y)} & f_{(x,y_+)} \\ 0 & f_{(x_+,y)} & 0 \end{bmatrix}\begin{bmatrix} \psi_{(x_-,y_-)} \\ \vdots \\ \psi_{(x_+,y_+)} \end{bmatrix}$$

After some manipulation,

$$\therefore -\frac{\hbar^2}{ms^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix} + \begin{bmatrix} 0 & v_{(x_+,y)} & 0 \\ v_{(x,y_-)} & 8v_{(x,y)} & v_{(x,y_+)} \\ 0 & v_{(x_-,y)} & 0 \end{bmatrix} = E\begin{bmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.40}$$

2.2.2 Finite-difference Method

The Finite-difference is a numerical method used to solve second-order partial differential equations.

From Schrodinger equation,

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right)\psi(x,y)+v(x,y)\psi(x,y)=E\psi(x,y) \tag{2.41}$$

The state representation 2D is as a grid.

$$=> \begin{bmatrix} \psi_{11} & \psi_{12} & \dots & \psi_{1N} \\ \psi_{21} & \psi_{22} & \dots & \psi_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{N1} & \psi_{N2} & \dots & \psi_{NN} \end{bmatrix} \tag{2.42}$$

The matrix 2.42 is converting vector is to take each row and stack them up,

$$=> \begin{bmatrix} \psi_{11} \\ \vdots \\ \psi_{1N} \\ \psi_{21} \\ \vdots \\ \psi_{2N} \\ \vdots \\ \psi_{NN} \end{bmatrix} \tag{2.43}$$

Here, N row and each row has N length. i.e., total size of length is N^2.

For example,

Let us consider 3 point in x-axis and 3 point in y-axis,

$$=> \begin{bmatrix} \psi_{(1,1)} & \psi_{(1,2)} & \psi_{(1,3)} \\ \psi_{(2,1)} & \psi_{(2,2)} & \psi_{(2,3)} \\ \psi_{(3,1)} & \psi_{(3,2)} & \psi_{(3,3)} \end{bmatrix} \tag{2.44}$$

The matrix equation 2.44 is converting vector is to take each row and stack them up,

$$=> \begin{bmatrix} \psi_{(1,1)} \\ \psi_{(1,2)} \\ \psi_{(1,3)} \\ \psi_{(2,1)} \\ \psi_{(2,2)} \\ \psi_{(2,3)} \\ \psi_{(3,1)} \\ \psi_{(3,2)} \\ \psi_{(3,3)} \end{bmatrix} \tag{2.45}$$

Here, the grid has 3 row and each row has 3 length. i.e., total size of length is 9.

The second derivative along x-direction is approximated as,

$$\frac{d^2\psi_i}{dx^2} = \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{(\Delta x)^2} \tag{2.46}$$

For i = 1, 2 and 3,

$$\frac{d^2\psi_1}{dx^2} = \frac{\psi_2 - 2\psi_1 + \psi_0}{(\Delta x)^2} \tag{2.47}$$

$$\frac{d^2\psi_2}{dx^2} = \frac{\psi_3 - 2\psi_2 + \psi_1}{(\Delta x)^2} \tag{2.48}$$

And,

$$\frac{d^2\psi_3}{dx^2} = \frac{\psi_4 - 2\psi_3 + \psi_2}{(\Delta x)^2} \tag{2.49}$$

Now, the approximation is written in matrix equation i = 1 to i = 3,

$$\therefore D_{xx} = \frac{1}{(\Delta x)^2}\begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \tag{2.50}$$

Similarly, the approximation along y-direction is,

$$\therefore D_{yy} = \frac{1}{(\Delta y)^2}\begin{bmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{bmatrix} \tag{2.51}$$

Assume that,

$$=> \frac{\partial^2\psi(x,y)}{\partial x^2} = D_{xx}\psi(x,y) \tag{2.52}$$

Also, x and y are swapped that $\psi^T(x,y)$ represent $\psi(x,y)$

$$=> \frac{\partial^2\psi(x,y)}{\partial y^2} = D_{yy}\psi^T(x,y) \tag{2.53}$$

In the end,

$$=> \frac{\partial^2\psi(x,y)}{\partial x^2} + \frac{\partial^2\psi(x,y)}{\partial y^2} = D_{xx} \oplus D_{yy} \tag{2.54}$$

$$\therefore \frac{\partial^2\psi(x,y)}{\partial x^2} + \frac{\partial^2\psi(x,y)}{\partial y^2} = I \otimes D_{xx} + D_{yy} \otimes I \tag{2.55}$$

Where, I is identity matrix. The Laplacian is the Kronecker sum of two sparse matrices D_{xx} and D_{yy}.

The term $I \otimes D_{xx}$ and $D_{yy} \otimes I$ can be written as,

$$=> I \otimes D_{xx} = \frac{1}{(\Delta x)^2}\begin{bmatrix} D & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & D \end{bmatrix} \tag{2.56}$$

$$\therefore I \otimes D_{xx} = \frac{1}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix} \tag{2.57}$$

And,

$$=> D_{yy} \otimes I = \frac{1}{(\Delta y)^2} \begin{bmatrix} -2I & 1I & 0 \\ 1I & -2I & 1I \\ 0 & 1I & -2I \end{bmatrix} \tag{2.58}$$

$$\therefore D_{yy} \otimes I = \frac{1}{(\Delta y)^2} \begin{bmatrix} -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -2 \end{bmatrix} \tag{2.59}$$

The potential energy also a grid we stretched out along that diagonal,

$$vI = \begin{bmatrix} v_{(1,1)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & v_{(1,2)} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_{(1,3)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & v_{(2,1)} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & v_{(2,2)} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & v_{(2,3)} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & v_{(3,1)} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{(3,2)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & v_{(3,3)} \end{bmatrix} \tag{2.60}$$

Therefore, the Schrödinger equation is,

$$\left(-\frac{\hbar^2}{2m} (D_{xx} \oplus D_{yy}) + v(x,y)I \right) \psi(x,y) = E\psi(x,y) \tag{2.61}$$

$\therefore$ The dimension of E is 81 and 9 for $\psi(x,y)$.
Similarly, For N points along x and y direction, dimension of E is $N^2 \times N^2$ and N^2 for $\psi(x,y)$.

Chapter 3

Results and Discussion

We have numerically solved the Schrödinger equation using Numerov and Finite-difference method for One and Two ddimensional systems. We calculated potential, eigenvalues, eigenfunctions and densities for Harmonic and Gaussian potential systems.

3.1 One-dimensional System

3.1.1 Numerov Method

ALGORITHM 1: Find the eigenvalues and eigenvectors for subatomic particles.

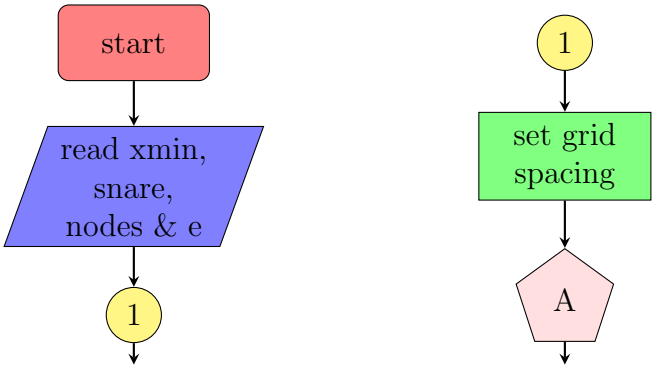
- **Step 1:** Start.
- **Step 2:** Read value xmin, snare, nodes and e.
- **Step 3:** Set grid spacing.
- **Step 4:** Start loop i = -snare-1 to snare.
  output x(i) and v(i).
- **Step 5:** Set the upper and lower energy.
- **Step 6:** Set the classical inversion point.
- **Step 7:** Start loop i = -snare-1 to snare.
  output f(i).
- **Step 8:** Collecting no. of sign changing at classical inversion point.
- **Step 9:** Check,
  if ilimit $\geq$ snare-2.
  stop.
  else if ilimit < 1.
  stop.

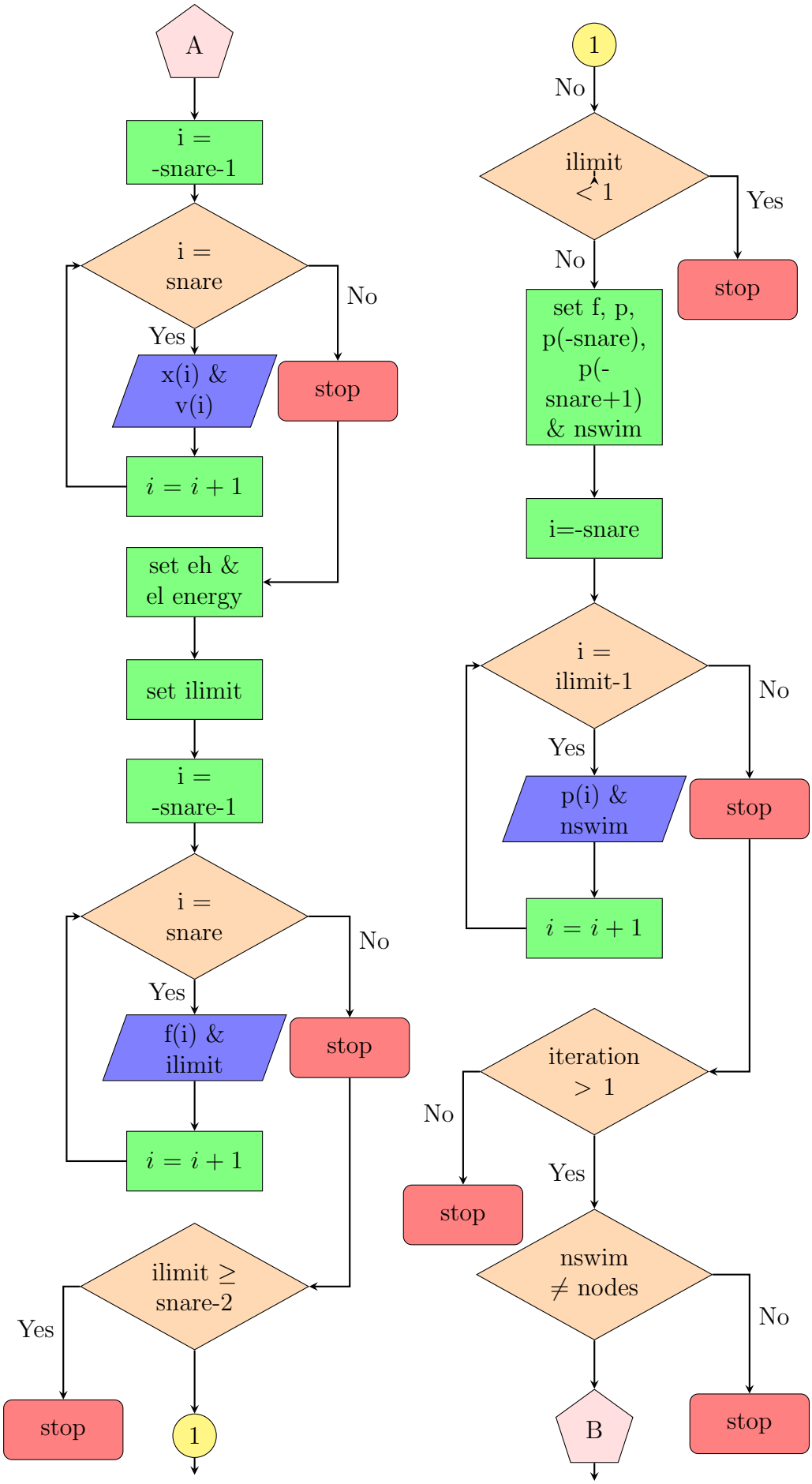
end if.

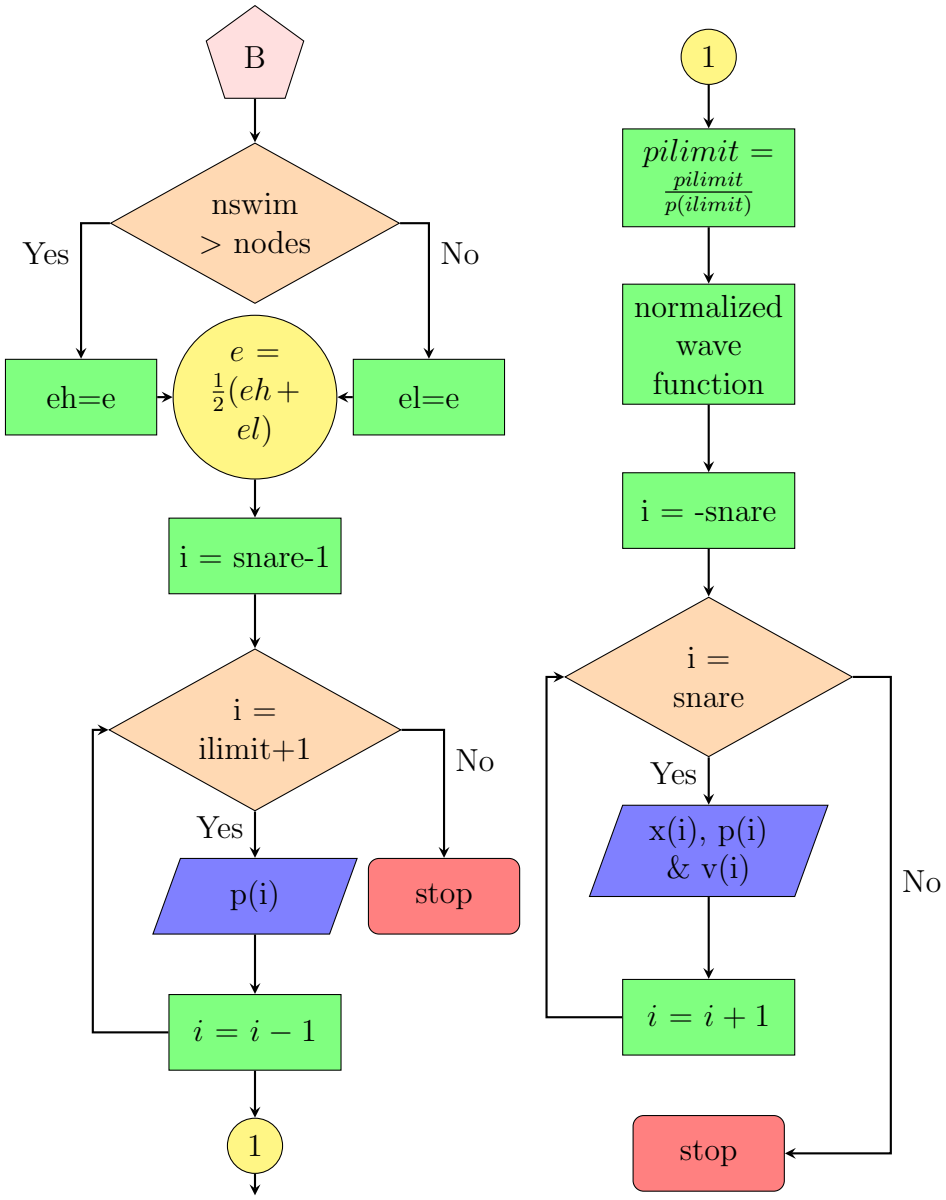
set the initial condition of f, y, y(-snare), y(-snare-1) and nswim.

- **Step 10:** Start loop i = -snare to ilimit-1.
  output p(i).
- **Step 11:** Collection no. of sign changing at p(i) to p(i+1).
- **Step 12:** Set the iteration for bisection method.
- **Step 13:** Check,
  if iteration $>$ 1. then go to step 14.
- **Step 14:** If nswim $\neq$ nodes. then go to the step 15.
- **Step 15:** If nswim $>$ nodes. then,
  upper energy = e.
  otherwise,
  lower energy = e.
- **Step 16:** Set the p(snare) and p(snare-1).
- **Step 17:** Start loop i = snare-1 to ilimit+1.
  output p(i).
- **Step 18:** Rescale function to match classical turning point.
- **Step 19:** Normalize the wave function.
- **Step 20:** Open a file to write the values x(i), p(i) and v(i).
- **Step 21:** Start loop i = -snare to snare.
  output x(i), p(i) and v(i).
- **Step 22:** Stop.

FLOWCHART 1: Find the eigenvalues and eigenvectors for sub-atomic particles.




A
i = -snare-1
i = snare
No
Yes
x(i) & v(i)
stop
$i = i + 1$
set eh & el energy
set ilimit
i = -snare-1
i = snare
No
Yes
f(i) & ilimit
stop
$i = i + 1$
ilimit ≥ snare-2
Yes
stop
1
1
No
ilimit < 1
Yes
No
stop
set f, p, p(-snare), p(-snare+1) & nswim
i=-snare
i = ilimit-1
No
Yes
p(i) & nswim
stop
$i = i + 1$
iteration > 1
No
stop
Yes
nswim ≠ nodes
No
B
stop



1. C Code for One-dimensional system wave functions and energies:

```
# include <stdlib.h>
# include <stdio.h>
# include <math.h>

int main() {
    double sqrt(double);
    int snare, i, ilimit;
    int nodes, nswim, kk, n_iter;
    double xmin, s, ds12, stand, pilimit, dup;
    double el, eh, e;
    double *x, *v, *f, *p;
    char output[50];
    FILE *out;

```

```
15      printf("Minimum value of x(typical value: 10)
            > ");
16      scanf("%lf", &xmin);
17      printf("Number of snare points(typically a
            few hundreds) > ");
18      scanf("%d", &snare);
19
20      /*  Allocate arrays */
21
22      x = (double*) malloc(2 * (snare + 1) * sizeof
            (double));
23      v = (double*) malloc(2 * (snare + 1) * sizeof
            (double));
24      f = (double*) malloc(2 * (snare + 1) * sizeof
            (double));
25      p = (double*) malloc(2 * (snare + 1) * sizeof
            (double));
26
27      s = xmin / snare;
28      ds12 = s * s / 12.;
29
30      /* Set the potential */
31
32      for (i = -snare-1; i <= snare; ++i) {
33      x[i] = (double) i * s;
34      // v[i] = 0.5 * x[i] * x[i];
35      // v[i] = (-3.3663525186199528*exp(-0.5*pow
            (((x[i]-0.17414787183635361)
            /1.6551400298191712),2)))
            -(7.8403723246565304*exp(-0.5*pow(((x[i
            ]-0.25716093609048035)/1.9364915388178923)
            ,2))) - (5.7377190919155829*exp(-0.5*pow(((
            x[i]-0.28340217582377836)
            /2.1894828763818577),2)));
36      v[i] = -1.1687*exp(-pow(((x[i]+2.2670)/
            1.5946), 2))+ 1.0378*exp(-pow(((x[i
            ]-0.2930)/0.3315),2))-0.7423*exp(-pow(((x[i
            ]+0.9267)/1.2013),2))+1.4192*exp(-pow(((x[i
            ]-0.3000)/0.6005),2) )-10.6312*exp(-pow(((x
            [i]-0.5275)/2.7498),2));
37      }
38
39      /* Read input data */
40
41      printf("Output file name > ");
42      scanf("%50s", output);
43      out = fopen(output, "w");
44
45      for (;;) {
46          printf("Number of nodes(type -ve value to
                stop) > ");
```

```
47          scanf("%d", &nodes);
48      if (nodes < 0) {
49              fclose(out);
50          exit(0);
51          }
52          eh = v[snare];
53          el = eh;
54          for (i = -snare-1; i <= snare; ++i) {
55          if (v[i] < el)
56                  el = v[i];
57          if (v[i] > eh)
58              eh = v[i];
59          }
60      printf("Trial energy(0=search with bisection)
            > ");
61      scanf("%lf", &e);
62      if (e == 0.) {
63          e = 0.5 * (eh + el);
64          n_iter = 1000;
65      }
66      else {
67          n_iter = 1;
68          }
69
70          for (kk = 1; (kk <= n_iter) && (eh-el >
               1.e-10); kk++) {
71              ilimit = 1;
72              for (i = -snare-1; i <= snare; ++i) {
73              f[i] = 2 * ds12 * (v[i] - e);
74                  if (f[i] == 0.) {
75                  f[i] = 1e-20;
76                  }
77                  if (f[i] != copysign(f[i], f[i
                       -1])) {
78                  ilimit = i;
79              }
80              }
81              if (ilimit >= snare-2) {
82              fprintf(stderr,"Error: last change of
                   sign too far");
83              exit(1);
84          }
85          if (ilimit < 1) {
86              fprintf(stderr, "Error: no classical
                   turning point");
87              exit(1);
88              }
89
90              for (i = -snare-1; i <= snare; ++i) {
91          f[i] = 1. - f[i];
92          }
```

```
93
94              for (i = -snare-1; i <= snare; ++i) {
95              p[i] = 0.;
96              }
97                  p[-snare - 1] = 0.;
98                  p[-snare] = s;
99
100                 nswim = 0;
101             for (i = -snare; i <= ilimit-1; ++i)
                    {
102             p[i + 1] = ((12. - 10. * f[i]) * p[i]
                    - f[i - 1] * p[i - 1]) / f[i + 1];
103                 if (p[i] != copysign(p[i], p[i +
                        1]))
104                ++nswim;
105             }
106             pilimit = p[ilimit];
107
108             if (n_iter > 1) {
109                if (nswim != nodes) {
110                        printf("%5d%25.15e%5d\n",
                           kk, e, nswim);
111
112                         if (nswim > nodes) {
113                            eh = e;
114                         }
115                         else {
116                            el = e;
117                         }
118                         e = 0.5 * (eh + el);
119                    }
120                }
121             else {
122                  printf("%25.15e%5d%5d\n", e,
                       nswim, nodes);
123             }
124                 if (n_iter == 1 || nswim == nodes
                        ) {
125                p[snare] = s;
126                    p[snare - 1] = (12. - 10. * f[
                       snare]) * p[snare] / f[snare
                       -1];
127            for (i = snare-1; i >= ilimit+1; --i)
                   {
128                p[i - 1] = ((12. - 10. * f[i]) * p
                       [i] - f[i + 1] * p[i + 1]) / f[i
                       - 1];
129                }
130                    pilimit /= p[ilimit];
131                    for (i = ilimit; i <= snare;
                           ++i) {
```

```
                p[i] *= pilimit;
                }

            stand = 0.;
            for (i = -snare; i <= snare; ++i) {
                    stand += p[i] * p[i];
            }
                stand = s * (stand + p[-snare-1] *
                    p[-snare-1]);
            stand = sqrt(stand);
            for (i = -snare; i <= snare; ++i) {
                p[i] /= stand;
                 }
                    if (n_iter > 1) {
                   i = ilimit;
                       dup = (p[i + 1] + p[i - 1]
                          - (14. - 12. * f[i]) * p[
                          i]) / s;
               printf("%5d%25.15e%5d%14.8f\n",kk,e
                  ,nodes,dup);
                       if (dup * p[i] > 0.) {
                  eh = e;
                   }
                       else {
                      el = e;
                       }
                       e = 0.5 * (eh + el);
                    }
                }
             }
             fprintf(out, "# x        y(x)       y(x
                )^2         v\n");

             /* Write value of eigenvectors and
                densities in a file */

             for (i = -snare; i <= snare; ++i) {
             fprintf(out, "%7.3f%16.8e%16.8e%16.8e
                \n",
                          x[i],  p[i],  p[i]*p[i],
                             v[i]);
        }
        fprintf(out, "\n\n");
        }
        free(x); free(v); free(f); free(p);
}
```

2. Fortran Code for One-dimensional system wave functions and energies:

```
      program potential
      implicit none
      integer, parameter :: er =
         selected_real_kind(14,100)
      integer :: snare, i, ilimit
      integer :: nodes, nswim, kk, n_iter
      real(er) :: xmin, s, ds12, stand, dup,
         pilimit
      real(er) :: eh, el, e
      real(er), allocatable :: x(:), v(:), f(:),
         p(:)
      character(len=50) :: output

      write(*,"('Minimum value of x(typical value
         : 10) > ')", advance = 'no')
      read(*,*) xmin
      write(*,"('Number of snare points(typically
          a few hundreds) > ')", advance = 'no')
      read(*,*) snare
      allocate(x(-snare-1:snare), v(-snare-1:
         snare), f(-snare-1:snare), p(-snare-1:
         snare))

      s = xmin / snare
      ds12 = s * s / 12.0_er

      ! Set the potential

      do i = - snare - 1, snare
        x(i) = float(i) * s
        ! v(i) = 0.5_er * x(i) * x(i)
        ! v(i) = (-3.3663525186199528*exp
           (-0.5*(((x(i)-0.17414787183635361)
           /1.6551400298191712))**2)) -&
        !(7.8403723246565304*exp(-0.5*(((x(i)
           -0.25716093609048035)
           /1.9364915388178923))**2)) -&
        !(5.7377190919155829*exp(-0.5*(((x(i)
           -0.28340217582377836)
           /2.1894828763818577))**2))
        v(i) = -1.1687*exp(-((x(i)+2.2670) /
           1.5946)**2)+&
        1.0378*exp(-((x(i)-0.2930)/0.3315)**2)
           -0.7423*exp(-((x(i)+0.9267)/1.2013)**2)
           +&
        1.4192*exp(-((x(i)-0.3000)/0.6005)**2)
           -10.6312*exp(-((x(i)-0.5275)/2.7498)
           **2)

      enddo

```

```
34        ! Read input data
35
36        write(*,"('Output file name > ')", advance
             = 'no')
37        read(*,'(a)') output
38        if(output /= ' ') &
39           open(1, file = output, status = 'unknown
                ', form = 'formatted')
40
41        do
42        write(*,"('Number of nodes(type -ve value
             to stop) > ')", advance = 'no')
43        read(*,*) nodes
44        if(nodes < 0) then
45          close(1)
46          deallocate(x, v, f, p)
47          stop
48        endif
49          eh = maxval(v(:))
50          el = minval(v(:))
51          write(*,"('Trial energy(0 = search with
                bisection) > ')", advance = 'no')
52          read(*,*) e
53          if(e == 0.0_er) then
54            e = 0.5_er * (eh + el)
55            n_iter = 1000
56          else
57            n_iter = 1
58          endif
59
60          do kk = 1, n_iter
61            ilimit = 1
62            do i = -snare-1, snare
63              f(i) = 2.0_er * ds12 * (v(i) - e)
64              if(f(i) == 0.0_er) f(i) = 1.d-20
65              if(f(i) /= sign(f(i),f(i - 1)))
                   ilimit = i
66            enddo
67            if(ilimit >= snare - 2) then
68              deallocate(x, v, f, p)
69              print*, 'Error: last change of sign
                   too far'
70              stop 1
71            elseif(ilimit < 1) then
72              deallocate(x, v, f, p)
73              print*, 'Error: no classical turning
                   point'
74              stop 1
75            endif
76
77            f = 1.0_er - f
```

```
78            p = 0.0_er
79
80            p(- snare - 1) = 0.0_er
81            p(- snare) = s
82            p(- snare + 1) = ((12.0_er - 10.0_er *
                 f(- snare)) * p(- snare) - f(- snare
                 - 1) * p(- snare - 1)) / f(- snare +
                 1)
83
84            nswim = 0
85            do i = - snare + 1, ilimit - 1
86              p(i + 1) = ((12.0_er - 10.0_er * f(i)
                   ) * p(i) - f(i - 1) * p(i - 1)) / f
                   (i + 1)
87              if(p(i) /= sign(p(i), p(i + 1)))
                   nswim = nswim + 1
88            enddo
89            pilimit = p(ilimit)
90
91            if(n_iter > 1) then
92              if(nswim /= nodes) then
93                print'(i5, f25.15, i5)', kk, e,
                     nswim
94                if(nswim > nodes) then
95                  eh = e
96                else
97                  el = e
98                endif
99                  e = 0.5_er * (eh + el)
100                 cycle
101             endif
102           else
103             print*, e, nswim, nodes
104           endif
105           p(snare) = s
106           p(snare - 1) = (12.0_er - 10.0_er * f(
                 snare)) * p(snare) / f(snare - 1)
107           do i = snare - 1, ilimit + 1, -1
108             p(i - 1) = ((12.0_er - 10.0_er * f(i)
                   ) * p(i) - f(i + 1) * p(i + 1)) / f
                   (i - 1)
109           enddo
110           pilimit = pilimit / p(ilimit)
111           p(ilimit:) = p(ilimit:) * pilimit
112           stand = 0.0_er
113           do i = -snare, snare
114           stand = stand + p(i) * p(i)
115           enddo
116           stand = s * (stand + p(-snare-1) * p(-
                 snare-1))
117           p = p / sqrt(stand)
```

```
118             if(n_iter > 1) then
119               dup = (p(ilimit + 1) + p(ilimit - 1)
                     - (14.0_er - 12.0_er * f(ilimit))*
                     p(ilimit)) / s
120               print '(i5, f25.15, i5, f14.8)', kk,
                     e, nodes, dup
121               if(dup * p(ilimit) > 0.0_er) then
122                 eh = e
123               else
124                 el = e
125               endif
126               e = 0.5_er * (eh + el)
127               if(eh - el < 1.d-10) exit
128             endif
129           enddo
130
131           write(1, '("# x    p(x)    p(x)^2    v(x)")'
                 )
132
133           ! Write value of eigenvectors and
                densities in a file
134
135           do i = - snare, snare
136             write(1, '(f8.3, 3e16.8, f12.6)') &
137               x(i), p(i), p(i) * p(i), v(i)
138           enddo
139           write(1,'(/)')
140         enddo
141
142         close(1)
143         deallocate(x, v, f, p)
144         endprogram potential
```

A. Harmonic potential:

$$v(x) = \frac{1}{2}kx^2$$

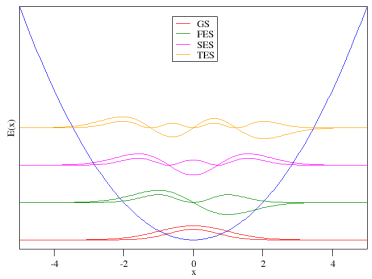


Figure 3.1: 1D Harmonic potential with wave functions and densities.

Sr No.	Nodes (n_x)	Energy(E_n)(a.u.)	
		Analytical	Numerov
1	0	0.5	0.5000
2	1	1.5	1.5000
3	2	2.5	2.5000
4	3	3.5	3.5000

Table 3.1: Represents state and energy of 1D Harmonic potential.

B. Gaussian potential:

$$v(x) = -\sum_{i=1}^{3} \alpha_i e^{-\frac{(x-\beta_i)^2}{2\gamma_i^2}}$$

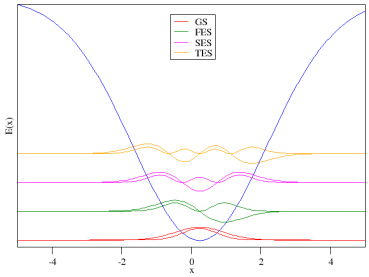


Figure 3.2: 1D Gaussian potential with wave functions and densities.

Sr No.	Nodes(n_x)	Energy(E_n)(a.u.)
1	0	-15.9042
2	1	-13.8855
3	2	-11.9753
4	3	-10.1770

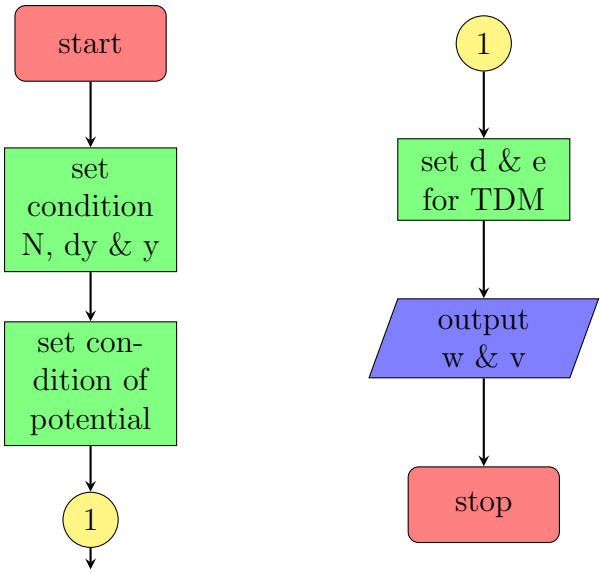
Table 3.2: Represents state and energy of 1D Gaussian potential.

3.1.2 Finite-difference Method

ALGORITHM 1: Find the eigenvalues and eigenvectors for subatomic particles.

- **Step 1:** Start.
- **Step 2:** Set the condition N, dy & y.
- **Step 3:** Set the condition for potential.
- **Step 4:** Set the main and 2 off diagonals for tridiogonal matrix.
- **Step 5:** Output eigenvalue and eigenvectors.
- **Step 6:** Stop.

FLOWCHART 1: Find the eigenvalues and eigenvectors for subatomic particles.



1. Python Code for One-dimensional system wave functions and energies:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import eigh_tridiagonal
from scipy.integrate import quad, simps
import math

N = 201
dy = 10/(N-1)
y = np.linspace(-5, 5, N+1)

# Set up the potential

def mL2V(y):
    # return 0.5*y**2
    # return (-3.3663525186199528*np.exp(-0.5*(((
       y-0.17414787183635361)/1.6551400298191712))
       **2)) - (7.8403723246565304*np.exp(-0.5*(((
       y-0.25716093609048035)/1.9364915388178923))
       **2)) - (5.7377190919155829*np.exp(-0.5*(((
       y-0.28340217582377836)/2.1894828763818577))
       **2))
    return -1.1687*np.exp(-(((y-(-2.2670))/
       1.5946)**2))+1.0378*np.exp(-(((y-(0.2930))
       /0.3315)**2))-0.7423*np.exp(-(((y-(-0.9267)
       )/1.2013)**2))+1.4192*np.exp(-(((y-(0.3000)
       )/0.6005)**2) )-10.6312*np.exp(-(((y
       -(0.5275))/2.7498)**2))

d = 1/dy**2 + mL2V(y)[1:-1]
e = -1/(2*dy**2)*np.ones(len(d)-1)
```

```
20
21 w, v = eigh_tridiagonal(d,e)
22 ground_state_psi= v.T[0]
23 ground_state_psi /= np.sqrt(np.trapz(
      ground_state_psi**2, y[1:-1]))
24 first_excited_state_psi = v.T[1]
25 first_excited_state_psi /= np.sqrt(np.trapz(
      first_excited_state_psi**2, y[1:-1]))
26 second_excited_state_psi = v.T[2]
27 second_excited_state_psi /= np.sqrt(np.trapz(
      second_excited_state_psi**2, y[1:-1]))
28 third_excited_state_psi = v.T[3]
29 third_excited_state_psi /= np.sqrt(np.trapz(
      third_excited_state_psi**2, y[1:-1]))
30
31 ground_state_density = ground_state_psi**2
32 first_excited_state_density =
      first_excited_state_psi**2
33 second_excited_state_density =
      second_excited_state_psi**2
34 third_excited_state_density =
      third_excited_state_psi**2
35
36 V=mL2V(y)
37
38 print("GS energy:", w[0])
39 print("FES energy:", w[1])
40 print("SES energy:", w[2])
41 print("TES energy:", w[3])
42
43 # Plots the eigenvectors
44
45 plt.plot(y, V)
46 plt.plot(y[1:-1],ground_state_psi, label='GS',
      color='orange')
47 plt.plot(y[1:-1],ground_state_density, color='
      orange')
48 plt.plot(y[1:-1],2+first_excited_state_psi, label
      ='FES', color='green')
49 plt.plot(y[1:-1],2+first_excited_state_density,
      color='green')
50 plt.plot(y[1:-1],4+second_excited_state_psi,
      label='SES', color='red')
51 plt.plot(y[1:-1],4+second_excited_state_density,
      color='red')
52 plt.plot(y[1:-1],6+third_excited_state_psi, label
      ='TES', color='blue')
53 plt.plot(y[1:-1],6+third_excited_state_density,
      color='blue')
54
55 plt.ylabel('E(x)')
```

```
plt.xlabel('x')
plt.legend(edgecolor="black", loc=0)
plt.show()
```

A. Harmonic potential:

$$v(x) = \frac{1}{2}kx^2$$

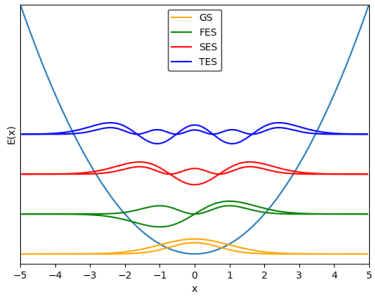


Figure 3.3: 1D Harmonic potential with wave functions and densities.

Sr No.	Nodes (n_x)	Energy(E_n)(a.u.)		
		Analytical	Numerical	
			NM	FDM
1	0	0.5	0.5000	0.4974
2	1	1.5	1.5000	1.4921
3	2	2.5	2.5000	2.4865
4	3	3.5	3.5000	3.4806

Table 3.3: Represents state and energy of 1D Harmonic potential.

B. Gaussian potential:

$$v(x) = -\sum_{i=1}^{3} \alpha_i e^{-\frac{(x-\beta_i)^2}{2\gamma_i^2}}$$

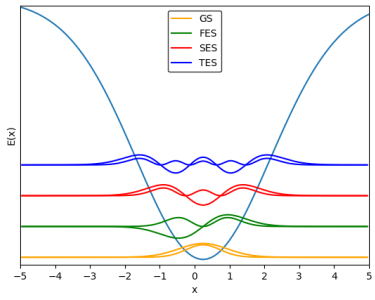


Figure 3.4: 1D Gaussian potential with wave functions and dnsities.

Sr No.	Nodes (n_x)	Energy(E_n)(a.u.)	
		NM	FDM
1	0	-15.9042	-15.9096
2	1	-13.8855	-13.9015
3	2	-11.9753	-12.0018
4	3	-10.1770	-10.2135

Table 3.4: Represents state and energy of Gaussian potential.

1. Numerov Method: **2. Finite-difference Method:**

In harmonic potential, we randomly change force constant value between 1.0-2.5 for 10000 potentials.

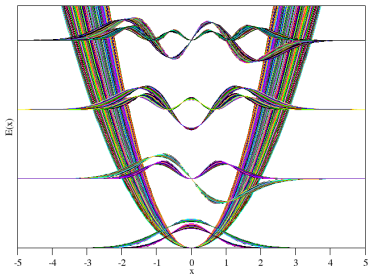


Figure 3.5: 1D Harmonic potentials with wave functions and densities.

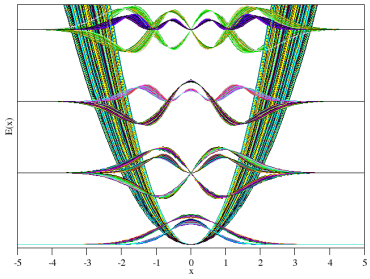


Figure 3.6: 1D Harmonic potentials with wave functions and densities.

In Gaussian potential, we randomly change three parameters α_i, β_i and γ_i value between 1.0-2.5, 0.1-0.6 and 1.5-2.5 for 10000 potentials.

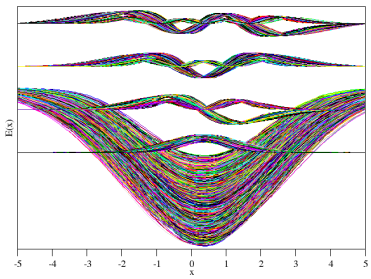


Figure 3.7: 1D Gaussian potentials with wave functions and densities.

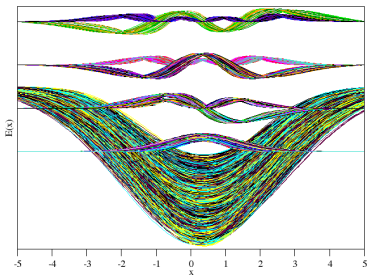


Figure 3.8: 1D Gaussian potentials with wave functions and densities.

3.2 Two-dimensional System

3.2.1 Numerov Method

3.2.1.1 Linear Numerov Method

ALGORITHM 1: Find the eigenvalues and eigenvectors for sub-atomic particles.

- **Step 1:** Start.
- **Step 2:** Read value hmin, snare, nodes_x, nodes_y and e_x.
- **Step 3:** Set grid spacing and classical inversion point.
- **Step 4:** Set the condition x, y, v_x and v_y.
- **Step 5:** Start loop i = -snare-1 to snare.
  output x(i) and v_x(i).
- **Step 6:** Start loop j = -snare-1 to snare.
  output y(j) and v_y(j).
- **Step 7:** Start loop i = -snare-1 to snare.
  output f_x(i).
- **Step 8:** Set upper energy and lower energy.
- **Step 9:** Collecting no. of sign changed at classical inversion point along x-direction.
- **Step 10:** Check,
  if limit $\geq$ snare-2.
  stop.
  else if limit $<$ 1.
  stop.
  end if.
  set the initial condition f_x, p_x, p_x(-snare-1), p_x(-snare) and nswim.
- **Step 11** Start loop i = -snare to limit-1.
  output p_x(i).
- **Step 12:** Collecting no. sign changed at p_x(i) to p_x(i+1).
- **Step 13:** Set the iteration for bisection method.
- **Step 14:** Check,
  if iteration $>$ 1, then go to step 14.

- **Step 15:** If nswim $\neq$ nodes_x. then go to step 15.
- **Step 16:** If nswim $>$ nodes_x. then,

  upper energy = e_x.

  otherwise,

  lower energy = e_x.
- **Step 17:** Set the p_x(snare) and p_x(snare-1).
- **Step 18:** Start loop i = snare-1 to limit+1.

  output p_x(i).
- **Step 19:** Rescale function p_x to match classical turning point.
- **Step 20:** Normalize the p_x wave functions.
- **Step 21:** Start loop j = -snare-1 to snare.

  output f_y(j).
- **Step 22:** Collecting no. of sign changed at classical inversion point along y-direction.
- **Step 23:** Check,

  if limit $\geq$ snare-2.

  stop.

  else if limit $<$ 1.

  stop.

  end if.

  set the initial condition of f_y, p_y, p_y(-snare-1), p_y(-snare) and nswim.
- **Step 24:** Start loop j = -snare to limit-1.

  output p_y(j).
- **Step 25:** Collecting no. of sign changed p_y(j) to p_y(j+1).
- **Step 26:** Set iteraction for bisection method.
- **Step 27:** Check,

  if iteration $>$ 1, then go to step 28.
- **Step 28:** If nswim $\neq$ nodes_y. then go to step 29.
- **Step 29:** If nswim $>$ nodes_y. then,

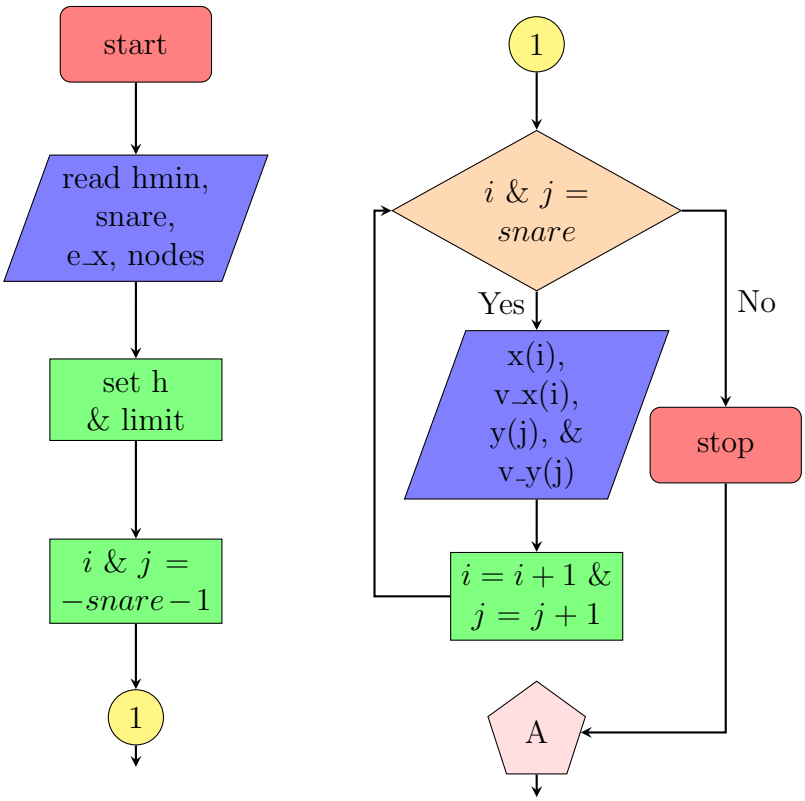
  upper energy = e_y.

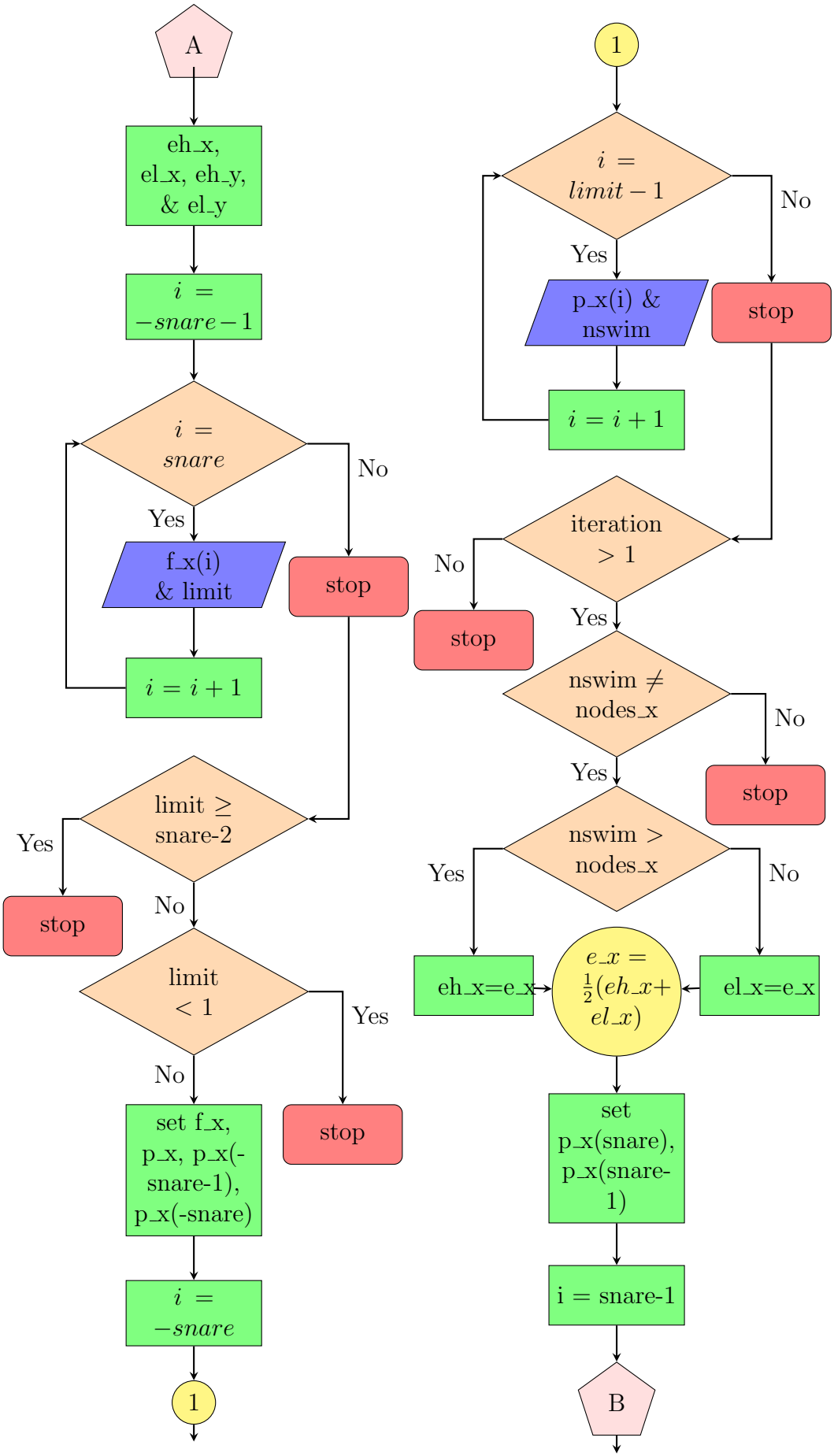
  otherwise,

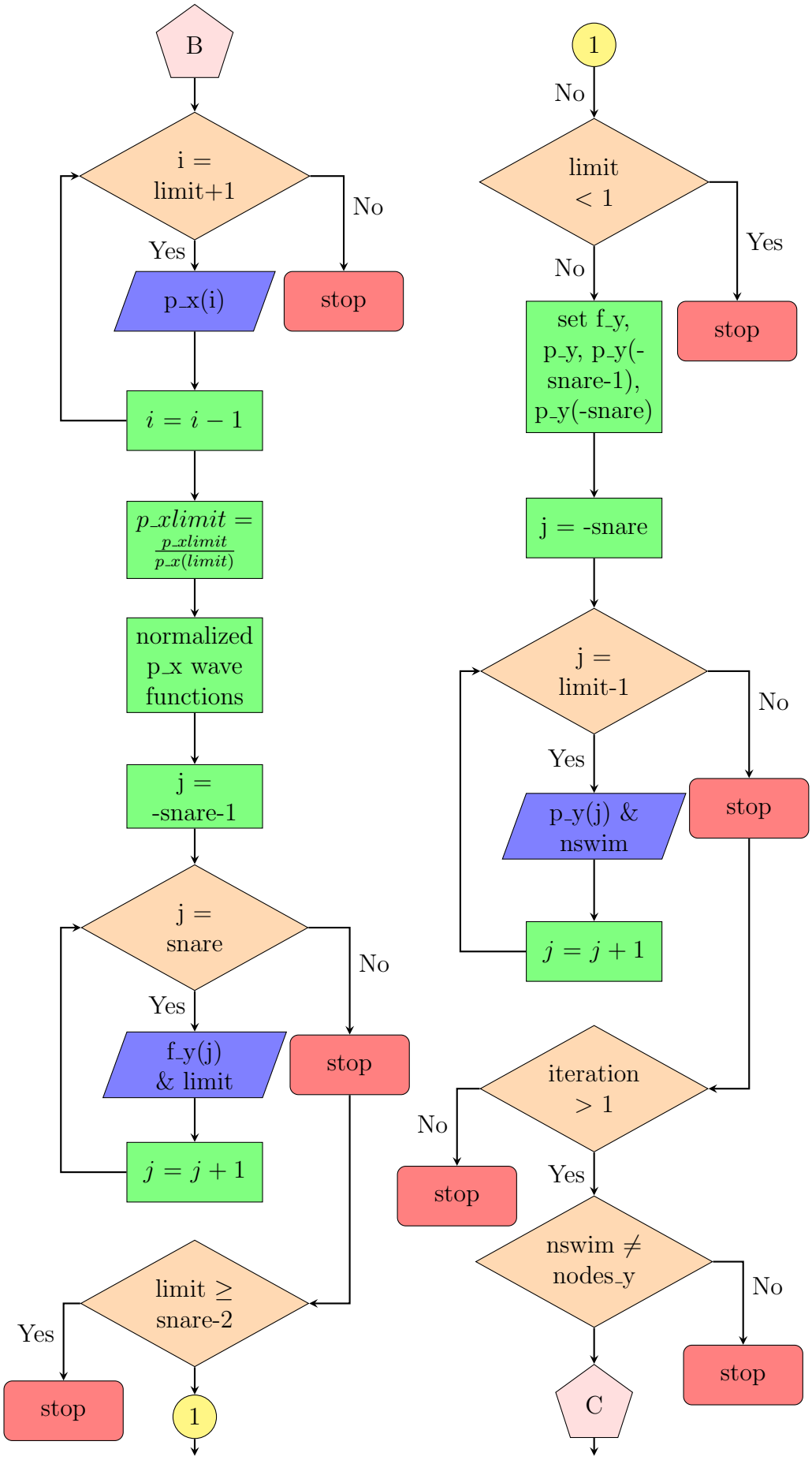
  lower energy = e_y.

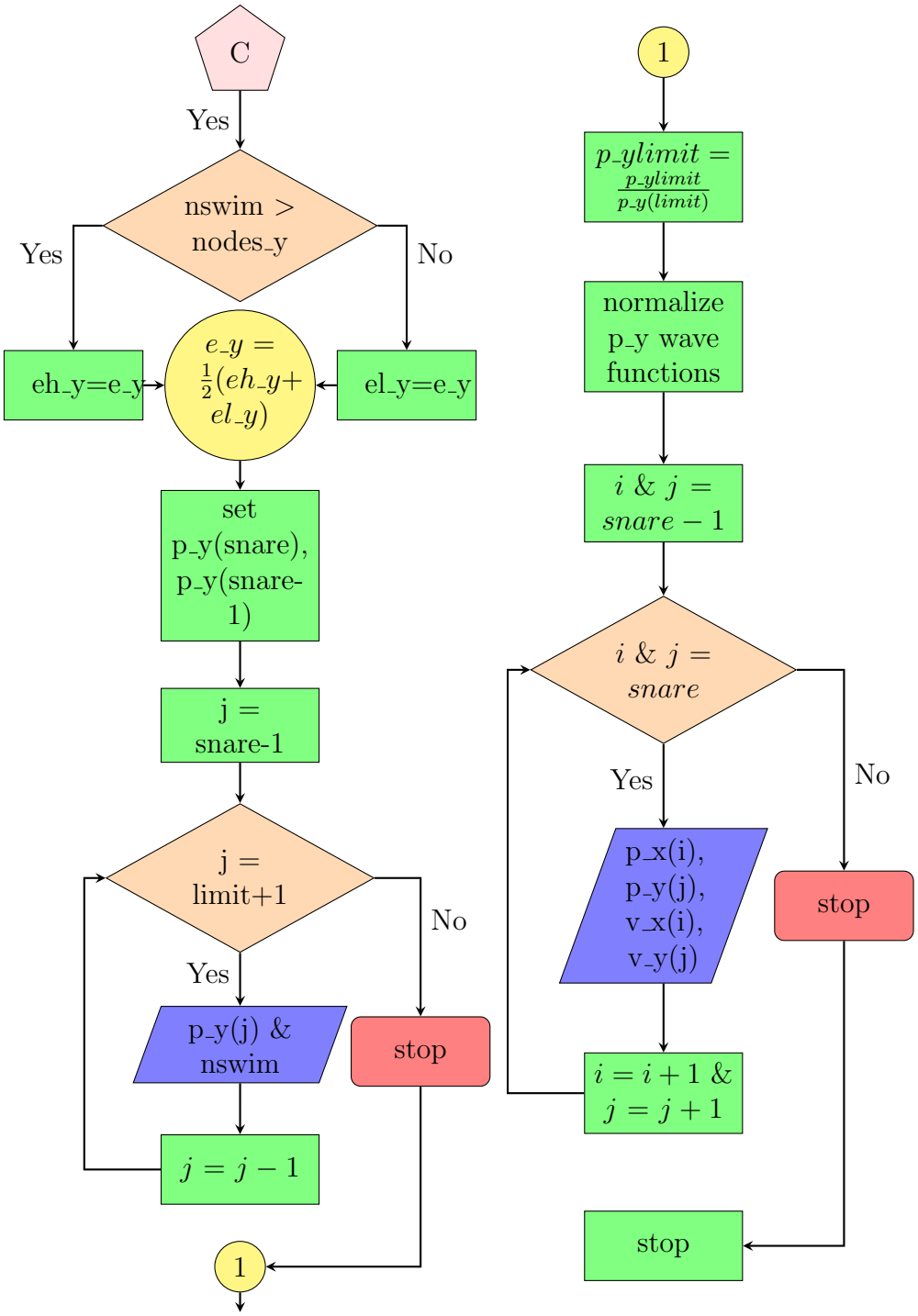
- **Step 30:** Set the p_y(snare) and p_y(snare-1).
- **Step 31:** Start loop j = snare-1 to limit+1.
  output p_y(j).
- **Step 32:** Rescale function p_y to match classical turning point.
- **Step 33:** Normalize p_y wave functions.
- **Step 34:** Open a file to write the value of x, y, p_x, p_y, v_x, v_y.
- **Step 35:** Start i and j is -snare to snare.
  output p_x(i), p_y(j), v_x(i) and v_y(j).
- **Step 36:** Stop.

FLOWCHART 1: Find the eigenvalues and eigenvectors for sub-atomic particles.




A
eh_x, el_x, eh_y, & el_y
$i = -snare - 1$
$i = snare$
No
Yes
f_x(i) & limit
stop
$i = i + 1$
limit ≥ snare-2
Yes
stop
No
limit < 1
Yes
No
set f_x, p_x, p_x(-snare-1), p_x(-snare)
stop
$i = -snare$
1
1
$i = limit - 1$
No
Yes
p_x(i) & nswim
stop
$i = i + 1$
iteration > 1
No
stop
Yes
nswim ≠ nodes_x
No
stop
Yes
nswim > nodes_x
Yes
No
eh_x=e_x
$e\_x = \frac{1}{2}(eh\_x + el\_x)$
el_x=e_x
set p_x(snare), p_x(snare-1)
i = snare-1
B


B
i = limit+1
No
Yes
p_x(i)
stop
$i = i - 1$
$p\_xlimit = \frac{p\_xlimit}{p\_x(limit)}$
normalized p_x wave functions
j = -snare-1
j = snare
No
Yes
f_y(j) & limit
stop
$j = j + 1$
limit ≥ snare-2
Yes
stop
1
1
No
limit < 1
No
Yes
stop
set f_y, p_y, p_y(-snare-1), p_y(-snare)
j = -snare
j = limit-1
No
Yes
p_y(j) & nswim
stop
$j = j + 1$
iteration > 1
No
stop
Yes
nswim ≠ nodes_y
No
stop
C



1. Fortran Code for Two-dimensional system wave functions and energies:

```
      program potential
      implicit none
      integer, parameter :: er =
     &   selected_real_kind(14,100)
      integer :: snare, i, j, limit
      integer :: nodes_x, nodes_y, nswim, kk,
     &   n_iter
```

```
6       real(er) :: hmin, h, dh12, stand, dup_x,
           dup_y, p_xlimit, p_ylimit
7       real(er) :: eh_x, el_x, e_x, eh_y, el_y,
           e_y
8       real(er), allocatable :: x(:), y(:), v_x(:)
           , v_y(:), f_x(:), f_y(:), p_x(:), p_y(:)
9       character(len=100) :: output
10
11      write(*,"('Minimum value of snare points(
           typical value: 10) > ')", advance = 'no')
12      read(*,*) hmin
13      write(*,"('Number of snare points(typically
            a few hundreds) > ')", advance = 'no')
14      read(*,*) snare
15      allocate(x(-snare-1:snare), y(-snare-1:
           snare), v_x(-snare-1:snare), v_y(-snare
           -1:snare), &
16      f_x(-snare-1:snare), f_y(-snare-1:snare),
           p_x(-snare-1:snare), p_y(-snare-1:snare))
17
18      h = hmin / snare
19      dh12 = h * h / 12.0_er
20      limit = 1
21
22      ! Set the potential
23
24      do i = - snare - 1, snare
25        x(i) = float(i) * h
26        v_x(i) = 0.5_er * x(i) * x(i)
27      enddo
28      do j = - snare - 1, snare
29        y(j) = float(j) * h
30        v_y(j) = 0.5_er * y(j) * y(j)
31      enddo
32
33      ! Read input value
34
35      write(*,"('Output file name > ')", advance
           = 'no')
36      read(*,'(a)') output
37      if(output /= ' ') &
38         open(1, file = output, status = 'unknown
              ', form = 'formatted')
39
40      do
41      write(*,"('Number of nodes in x-axis(type -
           ve value to stop) > ')", advance = 'no')
42      read(*,*) nodes_x
43      write(*,"('Number of nodes in y-axis(type -
           ve value to stop) > ')", advance = 'no')
44      read(*,*) nodes_y
```

```
45        if((nodes_x < 0 .and. nodes_y < 0) .or.
             nodes_x < 0 .or. nodes_y < 0) then
46          close(1)
47          deallocate(x, y, v_x, v_y, f_x, f_y, p_x,
                p_y)
48          stop
49        endif
50          eh_x = maxval(v_x(:))
51          el_x = minval(v_x(:))
52          eh_y = maxval(v_y(:))
53          el_y = minval(v_y(:))
54          write(*,"('Trial energy(0 = search with
               bisection) > ')", advance = 'no')
55          read(*,*) e_x
56          e_y = e_x
57          if(e_x == 0.0_er .and. e_y == 0.0_er)
               then
58            e_x = 0.5_er * (eh_x + el_x)
59            e_y = 0.5_er * (eh_y + el_y)
60            n_iter = 1000
61          else
62            n_iter = 1
63          endif
64
65          do kk = 1, n_iter
66            do i = -snare-1, snare
67              f_x(i) = 2.0_er * dh12 * (v_x(i) -
                    e_x)
68              if(f_x(i) == 0.0_er) f_x(i) = 1.d-20
69              if(f_x(i) /= sign(f_x(i),f_x(i - 1)))
                     limit = i
70            enddo
71            if(limit >= snare - 2) then
72              deallocate(x, y, v_x, v_y, f_x, f_y,
                    p_x, p_y)
73              print*, 'Error: last change of sign
                    too far'
74              stop 1
75            elseif(limit < 1) then
76              deallocate(x, y, v_x, v_y, f_x, f_y,
                    p_x, p_y)
77              print*, 'Error: no classical turning
                    point'
78              stop 1
79            endif
80
81            f_x = 1.0_er - f_x
82            p_x = 0.0_er
83
84            p_x(- snare - 1) = 0.0_er
85            p_x(- snare) = h
```

```
86
87            nswim = 0
88            do i = - snare, limit - 1
89              p_x(i + 1) = ((12.0_er - 10.0_er *
                   f_x(i)) * p_x(i) - f_x(i - 1) * p_x
                   (i - 1)) / f_x(i + 1)
90              if(p_x(i) /= sign(p_x(i), p_x(i + 1))
                   ) nswim = nswim + 1
91            enddo
92            p_xlimit = p_x(limit)
93
94            if(n_iter > 1) then
95              if(nswim /= nodes_x) then
96                if(nswim > nodes_x) then
97                  eh_x = e_x
98                else
99                  el_x = e_x
100               endif
101                 e_x = 0.5_er * (eh_x + el_x)
102                 cycle
103             endif
104           else
105           endif
106           p_x(snare) = h
107           p_x(snare - 1) = (12.0_er - 10.0_er *
                 f_x(snare)) * p_x(snare) / f_x(snare
                 - 1)
108           do i = snare - 1, limit + 1, -1
109             p_x(i - 1) = ((12.0_er - 10.0_er *
                   f_x(i)) * p_x(i) - f_x(i + 1) * p_x
                   (i + 1)) / f_x(i - 1)
110           enddo
111           p_xlimit = p_xlimit / p_x(limit)
112           p_x(limit:) = p_x(limit:) * p_xlimit
113           stand = 0.0_er
114           do i = -snare, snare
115           stand = stand + p_x(i) * p_x(i)
116           enddo
117           stand = h * (stand + p_x(-snare-1) *
                 p_x(-snare-1))
118           p_x = p_x / sqrt(stand)
119           if(n_iter > 1) then
120             dup_x = (p_x(limit + 1) + p_x(limit -
                   1) - (14.0_er - 12.0_er * f_x(
                   limit))* p_x(limit)) / h
121             if(dup_x * p_x(limit) > 0.0_er) then
122               eh_x = e_x
123             else
124               el_x = e_x
125             endif
126             e_x = 0.5_er * (eh_x + el_x)
```

```
127             if(eh_x - el_x < 1.d-10) exit
128           endif
129         enddo
130
131         do kk = 1, n_iter
132           do j = -snare-1, snare
133             f_y(j) = 2.0_er * dh12 * (v_y(j) -
                  e_y)
134             if(f_y(j) == 0.0_er) f_y(j) = 1.d-20
135             if(f_y(j) /= sign(f_y(j),f_y(j - 1)))
                   limit = j
136           enddo
137           if(limit >= snare - 2) then
138             deallocate(x, y, v_x, v_y, f_x, f_y,
                  p_x, p_y)
139             print*, 'Error: last change of sign
                  too far'
140             stop 1
141           elseif(limit < 1) then
142             deallocate(x, y, v_x, v_y, f_x, f_y,
                  p_x, p_y)
143             print*, 'Error: no classical turning
                  point'
144             stop 1
145           endif
146
147           f_y = 1.0_er - f_y
148           p_y = 0.0_er
149
150           p_y(- snare - 1) = 0.0_er
151           p_y(- snare) = h
152
153           nswim = 0
154           do j = - snare, limit - 1
155             p_y(j + 1) = ((12.0_er - 10.0_er *
                  f_y(j)) * p_y(j) - f_y(j - 1) * p_y
                  (j - 1)) / f_y(j + 1)
156             if(p_y(j) /= sign(p_y(j), p_y(j + 1))
                  ) nswim = nswim + 1
157           enddo
158           p_ylimit = p_y(limit)
159
160           if(n_iter > 1) then
161             if(nswim /= nodes_y) then
162               if(nswim > nodes_y) then
163                 eh_y = e_y
164               else
165                 el_y = e_y
166               endif
167                 e_y = 0.5_er * (eh_y + el_y)
168                 cycle
```

```
169             endif
170           else
171           endif
172           p_y(snare) = h
173           p_y(snare - 1) = (12.0_er - 10.0_er *
                 f_y(snare)) * p_y(snare) / f_y(snare
                 - 1)
174           do j = snare - 1, limit + 1, -1
175             p_y(j - 1) = ((12.0_er - 10.0_er *
                   f_y(j)) * p_y(j) - f_y(j + 1) * p_y
                   (j + 1)) / f_y(j - 1)
176           enddo
177           p_ylimit = p_ylimit / p_y(limit)
178           p_y(limit:) = p_y(limit:) * p_ylimit
179           stand = 0.0_er
180           do j = -snare, snare
181           stand = stand + p_y(j) * p_y(j)
182           enddo
183           stand = h * (stand + p_y(-snare-1) *
                 p_y(-snare-1))
184           p_y = p_y / sqrt(stand)
185           if(n_iter > 1) then
186             dup_y = (p_y(limit + 1) + p_y(limit -
                    1) - (14.0_er - 12.0_er * f_y(
                   limit))* p_y(limit)) / h
187             if(dup_y * p_y(limit) > 0.0_er) then
188               eh_y = e_y
189             else
190               el_y = e_y
191             endif
192             e_y = 0.5_er * (eh_y + el_y)
193             if(eh_y - el_y < 1.d-10) exit
194           endif
195         enddo
196
197         write(*,"('Energy = ')", advance = 'no')
198         write(*,'(2f18.12)') e_x + e_y
199
200         write(1, '("# x   p(x)   p(x)^2   v(x)")'
              )
201
202         ! Write value of eigenvectors and
              densities in a file
203
204         do i = - snare, snare
205           do j = - snare, snare
206             write(1, '(2f8.3, 6e16.8, 2f12.6)') &
207               x(i), y(j), p_x(i)*p_y(j), +p_x(i)*
                    p_x(i)*p_y(j)*p_y(j), v_x(i)+v_y(
                    j)
208           enddo
```

```
209           enddo
210           write(1,'(/)')
211         enddo
212
213         close(1)
214         deallocate(x, y, v_x, v_y, f_x, f_y, p_x,
              p_y)
215         endprogram potential
```

A. Harmonic potential:

$$v(x,y) = \frac{1}{2}k(x^2 + y^2)$$

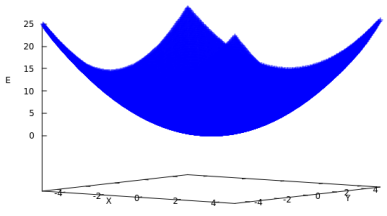


Figure 3.9: 2D Harmonic potential.

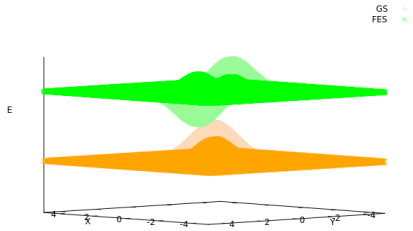


Figure 3.10: Wave functions and Densities of 2D Harmonic potential.

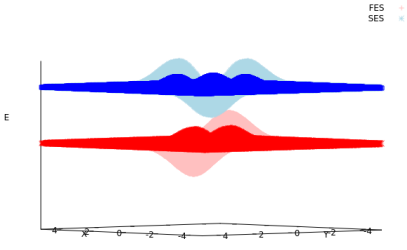


Figure 3.11: Wave functions and Densities of 2D Harmonic potential.

Sr No.	Nodes		Energy(E_n)(a.u.)	
	n_x	n_y	Analytical	Numerov
1	0	0	1	1.0000
2	1	0	2	2.0000
3	0	1	2	2.0000
4	1	1	3	3.0000

Table 3.5: Represents state and energy of 2D Harmonic potential.

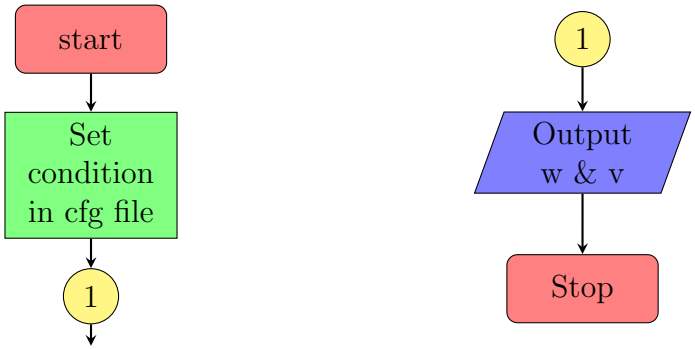
3.2.1.2 Matrix-Numerov method

ALGORITHM 1: Find the eigenvalues and eigenvectors for subatomic particles.

- **Step 1:** Start.
- **Step 2:** Set the condition 2D system in cfg file.

- **Step 3:** Output is eigenvalue and eigenvectors.
- **Step 4:** Stop.

FLOWCHART 1: Find the eigenvalues and eigenvectors for sub-atomic particles.



1. Python Code for Two-dimensional system wave functions and energies:

```
import numpy as np
import subprocess
import sys
import os
import os.path
from ConfigParser import SafeConfigParser
from lib.NuSol_cfg_obj import NuSol_cfg_obj
from lib.NuSol_matrices import NuSol_matrices
from lib.NuSol_version_checker import
   NuSol_version
from scipy.linalg import solve
import scipy.optimize as op
import scipy.sparse as sp

# Read config.cfg file

class numerov():
  def __init__ (self,cfgname):
    cfg = SafeConfigParser()
    cfg.read(cfgname)
    cfg = NuSol_cfg_obj(cfg)
    NuSolM = NuSol_matrices(cfg)

    if cfg.METHOD == 'numerov':
      if cfg.NDIM == 2:
        print ('Creating 2D grid -- %dx%d=%d
           points [XY] -- grid spacing %f Bohr' %
```

```
            (cfg.NGRIDX,cfg.NGRIDY,cfg.NGRIDX*cfg.
            NGRIDY,cfg.h))
26        A,M = NuSolM.Numerov_Matrix_2D()
27      if cfg.USE_FEAST == 'true' :
28        if os.path.exists("%s/NuSol_FEAST"%(cfg.
            FEAST_PATH)):
29          n = subprocess.Popen('ldd %s/
              NuSol_FEAST| grep "not found" | wc -l
              '% (cfg.FEAST_PATH),shell=True,
              stdout=subprocess.PIPE, stderr=
              subprocess.STDOUT)
30          libsloaded = int( n.stdout.readlines()
              [0].strip('\n') )
31          if libsloaded == 0:
32            p = subprocess.Popen('%s/NuSol_FEAST
                %f %f %d %s %s %s' % (cfg.
                FEAST_PATH,cfg.FEAST_E_MIN,cfg.
                FEAST_E_MAX,cfg.FEAST_M,cfg.
                FEAST_MATRIX_OUT_PATH,cfg.
                EIGENVALUES_OUT,cfg.
                EIGENVECTORS_OUT),shell=True,
                stdout=subprocess.PIPE, stderr=
                subprocess.STDOUT)
33            for line in p.stdout.readlines():
34                print (line,)
35            retval = p.wait()
36          else:
37            sys.exit()
38      else:
39
40        # Write value of eigenvales and
            eigenvectors in a eval.dat and evec.dat
            file respectively
41
42        eval,evec = sp.linalg.eigs(A=A,k=cfg.
            N_EVAL,M=M,which='SM')
43        cfg.WRITE_EVAL_AND_EVEC(eval,evec)
44
45  if __name__ == "__main__":
46    if len(sys.argv) == 2:
47      NuV = NuSol_version()
48      res = NuV.version_check()
49      if res == True:
50        if os.path.isfile(sys.argv[1]):
51          numerov(sys.argv[1])
52        else:
53          print ('%s not exist' % (sys.argv[1]) )
54          sys.exit()
55      else:
56        print ('exiting..')
57    else:
```

```
58      sys.exit(1)
```

A. Harmonic potential:

$$v(x,y) = \frac{1}{2}k(x^2+y^2)$$

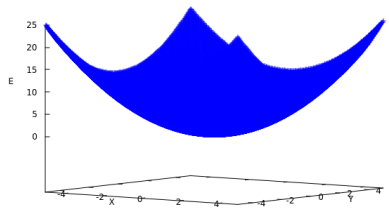


Figure 3.12: 2D Harmonic potential.

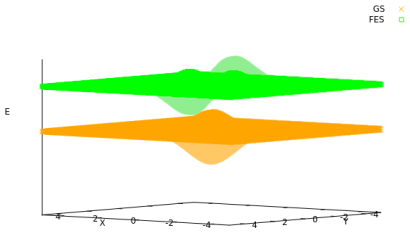


Figure 3.13: Wave functions and Densities of 2D Harmonic potential.

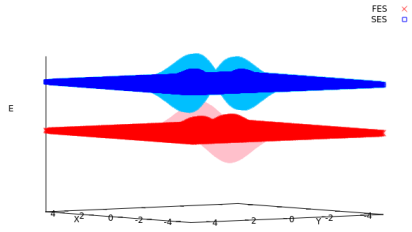


Figure 3.14: Wave functions and Densities of 2D Harmonic potential.

Sr No.	Nodes		Energy(E_n)(a.u.)	
	n_x	n_y	Analytical	Numerov
1	0	0	1	0.9999
2	1	0	2	1.9999
3	0	1	2	1.9999
4	1	1	3	2.9989

Table 3.6: Represents state and energy of 2D Harmonic potential.

B. Gaussian potential:

$$v(x,y) = \sum_{i=1}^{3} \alpha_i e^{-\frac{1}{2}\left(\frac{(x-\beta_i)^2}{\gamma_i^2}+\frac{(y-\delta_i)^2}{\theta_i^2}\right)}$$

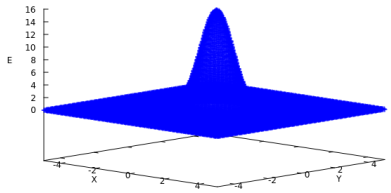


Figure 3.15: 2D Gaussian potential.

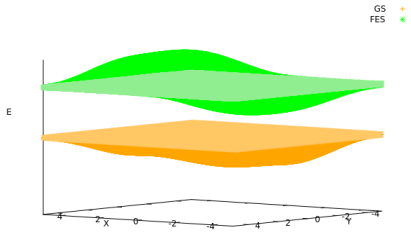


Figure 3.16: Wave functions and Densities of 2D Gaussian potential.

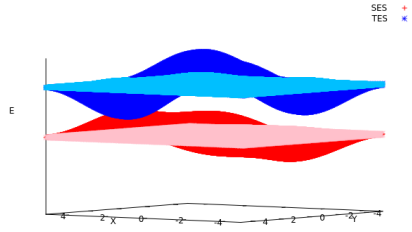


Figure 3.17: Wave functions and Densities of 2D Gaussian potential.

Sr No.	n_x	n_y	Energy(E_n)(a.u.)
1	0	0	0.2659
2	1	0	0.3058
3	0	1	0.3281
4	1	1	0.4089

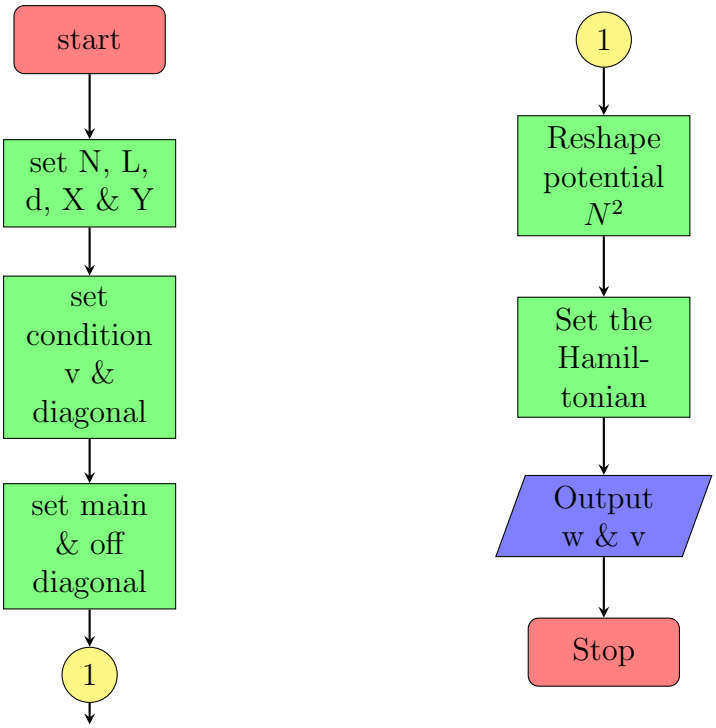
Table 3.7: Represents state and energy of 2D Gaussian potential.

3.2.2 Finite-difference Method

ALGORITHM 1: Find the eigenvalues and eigenvectors for subatomic particles.

- **Step 1:** Start.
- **Step 2:** Set the N, L, d, X and Y.
- **Step 3:** Set the condition of potential.
- **Step 4:** Set the condition of diagonal.
- **Step 5:** Set the main and off diagonal in matrix D.
- **Step 6:** Reshape the potential N^2.
- **Step 7:** Set the Hamiltonian.
- **Step 8:** Output is eigenvalue and eigenvectors.
- **Step 9:** Stop.

FLOWCHART 1: Find the eigenvalues and eigenvectors for subatomic particles.



1. Python Code for Two-dimensional system wave functions and energies:

```
import numpy as np
from scipy.sparse.linalg import eigsh
from scipy.sparse.linalg import eigs
import matplotlib.pyplot as plt
from scipy import sparse
from scipy.signal import square
from scipy.integrate import trapz

file1 = open("ground.dat", "w")
file2 = open("first.dat", "w")
file3 = open("second.dat", "w")
file4 = open("third.dat", "w")
file5 = open("poten.dat", "w")

N = 201
L = 10
d = L/(N-1)
x = np.linspace(-L/2, L/2, N)
y = np.linspace(-L/2, L/2, N)
Y, X = np.meshgrid(x,y)

# Set up the potential

def get_potential(X,Y):
    # return (X**2 + Y**2)/2
    return 6.8422439315313168*np.exp(-0.5*(((X
```

```
        -7.7536955648400463E-002)
        /0.67191003547625294)**2 + ((Y
        -7.5807538318280493E-002)
        /0.58102442458601489)**2)) +
        7.3665089077098562*np.exp(-0.5*(((X
        -5.0472449790760050E-002)
        /0.82306070065133252)**2 + ((Y
        -7.6467498056468242E-002)
        /0.57741901026136055)**2)) +
        1.6189654779676654*np.exp(-0.5*(((X
        -3.9395674140487003E-002)
        /0.86843269402942580)**2 + ((Y
        -4.5752897234166159E-002)
        /0.72697115586723582)**2))

diag = 1/d**2*np.ones([N])
diags = np.array([diag, -2*diag, diag])
D = sparse.spdiags(diags, np.array([-1,0,1]), N,
    N)
T = -1/2*sparse.kronsum(D,D)
U = sparse.diags(get_potential(X,Y).reshape(N**2)
    ,(0))
H = T+U

eigenvalues, eigenvectors = eigsh(H, k=10, which=
    'SM')
print(eigenvalues[0])
print(eigenvalues[1])
print(eigenvalues[2])
print(eigenvalues[3])

def get_v(n):
    return eigenvectors.T[n].reshape(N,N)

ground_state_psi = get_v(0)
ground_state_psi /= np.sqrt(np.trapz(np.trapz(
    ground_state_psi**2, X, axis=0), Y))
first_excited_state_psi = get_v(1)
first_excited_state_psi /= np.sqrt(np.trapz(np.
    trapz(first_excited_state_psi**2, X, axis=0), Y
    ))
second_excited_state_psi = get_v(2)
second_excited_state_psi /= np.sqrt(np.trapz(np.
    trapz(second_excited_state_psi**2, X, axis=0),
    Y))
third_excited_state_psi = get_v(3)
third_excited_state_psi /= np.sqrt(np.trapz(np.
    trapz(third_excited_state_psi**2, X, axis=0), Y
    ))

for X1, Y1, gsp in zip(X, Y, ground_state_psi):
```

```
    for X1, Y1, gsp in zip(X1, Y1, gsp):
        print("%2.5f %2.5f %2.10e" %(X1, Y1, gsp)
            , file=file1)
for X1, Y1, fesp in zip(X, Y,
    first_excited_state_psi):
    for X1, Y1, fesp in zip(X1, Y1, fesp):
        print("%2.5f %2.5f %2.10e" %(X1, Y1, fesp
            ), file=file2)
for X1, Y1, sesp in zip(X, Y,
    second_excited_state_psi):
    for X1, Y1, sesp in zip(X1, Y1, sesp):
        print("%2.5f %2.5f %2.10e" %(X1, Y1, sesp
            ), file=file3)
for X1, Y1, tesp in zip(X, Y,
    third_excited_state_psi):
    for X1, Y1, tesp in zip(X1, Y1, tesp):
        print("%2.5f %2.5f %2.10e" %(X1, Y1, tesp
            ), file=file4)
for X1, Y1, v in zip(X, Y, get_potential(X,Y)):
    for X1, Y1, v in zip(X1, Y1, v):
        print("%2.5f %2.5f %2.10e" %(X1, Y1, v),
            file=file5)

# Plots of the potential and eigenvectors

fig = plt.figure(1,figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, get_potential(X,Y), cmap='
    plasma')
ax.set_xlabel(r' $ X $ ')
ax.set_ylabel(r' $ Y $ ')
ax.set_zlabel(r' $ E $ ')
plt.show()

fig = plt.figure(2,figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, ground_state_psi, cmap='
    plasma', color='orange', label='GS')
ax.plot_surface(X, Y, ground_state_psi**2, cmap='
    plasma', color='orange')
ax.plot_surface(X, Y, 1+first_excited_state_psi,
    cmap='plasma', color='green', label='FES')
ax.plot_surface(X, Y, 1+first_excited_state_psi
    **2, cmap='plasma', color='green')
ax.set_xlabel(r' $ X $ ')
ax.set_ylabel(r' $ Y $ ')
ax.set_zlabel(r' $ E $ ')
ax.set_zticks([])
plt.legend()
plt.show()

```

```
fig = plt.figure(3,figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, second_excited_state_psi,
    cmap='plasma', color='red', label='SES')
ax.plot_surface(X, Y, second_excited_state_psi
    **2, cmap='plasma', color='red')
ax.plot_surface(X, Y, 1+third_excited_state_psi,
    cmap='plasma', color='blue', label='TES')
ax.plot_surface(X, Y, 1+third_excited_state_psi
    **2, cmap='plasma', color='blue')
ax.set_xlabel(r' $ X $ ')
ax.set_ylabel(r' $ Y $ ')
ax.set_zlabel(r' $ E $ ')
ax.set_zticks([])
plt.legend()
plt.show()
```

A. Harmonic potential:

$$v(x,y) = \frac{1}{2}k(x^2+y^2)$$

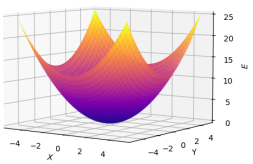


Figure 3.18: 2D Harmonic potential.

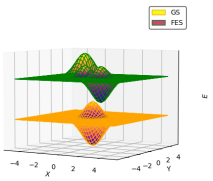


Figure 3.19: Wave functions and Densities of 2D Harmonic potential.

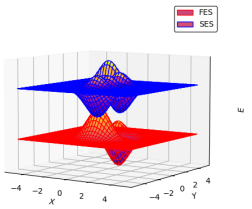


Figure 3.20: Wave functions and Densities of 2D Harmonic potential.

Sr No.	Nodes		Energy(E_n)(a.u.)		
	n_x	n_y	Analytical	Numerical	
				NM	FDM
1	0	0	1	0.9999	0.9998
2	1	0	2	1.9999	1.9995
3	0	1	2	1.9999	1.9995
4	1	1	3	2.9989	2.9980

Table 3.8: Represents state and energy 2D of Harmonic potential.

B. Gaussian potential:

$$v(x,y) = \sum_{i=1}^{3} \alpha e^{-\frac{1}{2}\left(\frac{(x-\beta_i)^2}{\gamma_i^2} + \frac{(y-\delta_i)^2}{\theta_i^2}\right)}$$

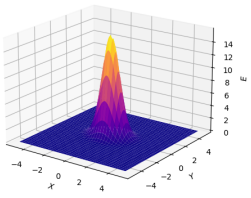


Figure 3.21: 2D Gaussian potential.

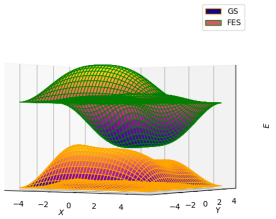


Figure 3.22: Wave functions and Densities of 2D Gaussian potential.

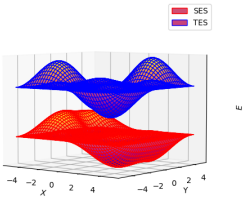


Figure 3.23: Wave functions and Densities of 2D Gaussian potential.

Sr No.	Nodes		Energy(E_n)(a.u.)	
	n_x	n_y	NM	FDM
1	0	0	0.2659	0.2659
2	1	0	0.3058	0.3057
3	0	1	0.3281	0.3281
4	1	1	0.4089	0.4089

Table 3.9: Represents state and energy of 2D Gaussian potential.

Chapter 4

Conclusion and Scope

In this project, we have successfully designed a code for numerical solution of One and Two dimensional Schrödinger eqution by using Numerov and Finite-diifence method. We have solved Schrödinger eqution for different type of symmetric and asymmetric potential wells. We have compared the eigenvalues and eigenvectors for Harmonic potential well with the analytical results and our results are quite similar. We found that Numerov and Finite-difference methods is suitable for various potential such as Gaussian potential and Double-well potential. Also we have generate 10000 wave functions and densities for Harmonic, Gaussian and Double-well potentials. The various symmetric and asymmetric potentials showed clear resemblance with the actual potentials given in text books. And through these exercises, I have been able to visualise the theories as well as apply those theories with the help of coding.

In machine learning models such as ANN required large data sets to avoid the overfitting, optimize the learning parameters effectively and training models that can generalize well to unseen data system. This is important to predicting the properties or behavior of new quantum system. A larger data sets adequately sample high dimensions space and learn meaningful patterns. The quantum system extremely complex chemistry applications and the researchers diverse wave function required for studying exotic quantum systems.

Appendix A

Double-well Potential

Here, we calculate potential, eigenvalues, eigenfunctions and densities for Double-well potential systems.

$$v(x) = \frac{1}{2}a(x^2 - b^2)^2 - cx$$

A.1 Numerov Method

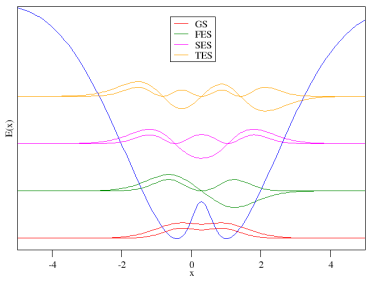


Figure A.1: 1D Double-well potential with wave functions and densities.

Sr No.	Nodes(n_x)	Energy(E_n)(a.u.)
1	0	-9.1717
2	1	-8.4669
3	2	-6.8829
4	3	-5.8053

Table A.1: Represents state and energy of 1D Double-well potential.

A.2 Finite-difference Method

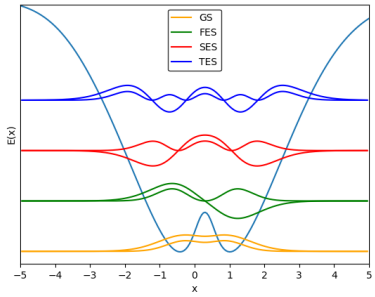


Figure A.2: 1D Double-well potential with wave functions and densities.

Sr No.	Nodes (n_x)	Energy(E_n)(a.u.)	
		NM	FDM
1	0	-9.1717	-9.1744
2	1	-8.4669	-8.4767
3	2	-6.8829	-6.8998
4	3	-5.8053	-5.8292

Table A.2: Represents state and energy of 1D Double-well potential.

1. Numerov Method: **2. Finite-difference Method:**

In Double-well potential, we randomly change three parameters a, b and c value between 1.0-1.5, 0.0-1.5 and 0.0-2.0 for 10000 potentials.

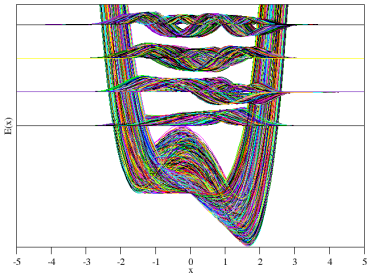


Figure A.3: 1D Double-well potentials with wave functions and densities.

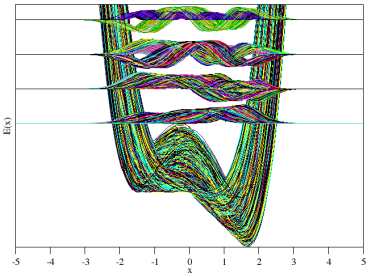


Figure A.4: 1D Double-well potentials with wave functions and densities.

REFERENCES

[1] Danny Bennett. Numerical solution of time-independent 1-d schrodinger equation. *Dec-15*, 2015.

[2] Francisco Caruso, Vitor Oguri, and Felipe Silveira. Applications of the numerov method to simple quantum systems using python. *Revista Brasileira de Ensino de Física*, 44:e20220098, 2022.

[3] Tang Dongjiao, Yeo Ye, and Mr Andreas Dewanto. Generalized matrix numerov solutions to the schrödinger equation. *Bachelorâ€™s thesis, National University of Singapore, Singapore*, 2014.

[4] Michael Eckert. Solving the 1-, 2-, and 3-dimensional schrödinger equation for multiminima potentials using the numerov-cooley method. an extrapolation formula for energy eigenvalues. *Journal of Computational Physics*, 82(1):147–160, 1989.

[5] Z Kalogiratou, Th Monovasilis, and TE Simos. Numerical solution of the two-dimensional time independent schrödinger equation with numerov-type methods. *Journal of mathematical chemistry*, 37(3):271–279, 2005.

[6] Ulrich Kuenzer, Jan-Andrè Sorarù, and Thomas S Hofer. Pushing the limit for the grid-based treatment of schrödinger's equation: a sparse numerov approach for one, two and three dimensional quantum problems. *Physical Chemistry Chemical Physics*, 18(46):31521–31533, 2016.

[7] Ira N. Levine. *Quantum Chemistry*. Asoke K. Ghosh, 2009.

[8] Munkhbaatar Purevkhuu and VI Korobov. On one implementation of the numerov method for the one-dimensional stationary schrödinger equation. *Physics of Particles and Nuclei Letters*, 18:153–159, 2021.

[9] Munkhbaatar Purevkhuu and VI Korobov. On one implementation of the numerov method for the one-dimensional station-

ary schrödinger equation. *Physics of Particles and Nuclei Letters*, 18:153–159, 2021.

[10] Wai Kui Wong. Solving 2d time independent schrodinger equation using numerical method. 12 2021.

[11] Nouredine Zettili. *Quantum Mechanics*. Wiley India Pvt. Ltd., 2018.