Introduction to Quantum Electronics: Numerically Solving the 1-D Time Independent Schrödinger Equation

ΚΟΣΜΑΣ ΑΡΧΟΝΤΗΣ, 1084020

University of Patras

January 26, 2024

Contents

1 Introduction **2**

2 Numerical Methods **3**

3 Problems Analyzed **9**

4 Python Code Overview **14**

5 Results **19**

6 Weaknesses and Improvements **28**

7 References **29**

1 Introduction

The purpose of this report is to present the theory and numerical methods implemented in the accompanied Python program, for the purpose of solving the 1-Dimensional, time-independent Schrödinger Equation.

The Schrödinger Equation is given by:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x) \tag{1.1}$$

Where $\psi(x)$ is the wavefunction, $V(x)$ is the potential energy, m the mass of the studied particle and $\hbar$ the reduced Planck constant.

There are various different methods and numerical approaches to solving the Schrödinger Equation, but we will focus on the Numerov-Cooley method, while briefly discussing Shooting and Bisection methods.

2 Numerical Methods

The **Shooting Method** relies on nothing more than the Discretised Time-Independent Schrödinger Equation in order to approximate the wavefunction, and convert an initial value problem to a *discretised initial value problem*:

$$\begin{aligned} \psi_{j+1} &= 2\left[\frac{m(\Delta x)^2}{\hbar^2}(V_j - E) + 1\right]\psi_j - \psi_{j-1}, \qquad j = 1, 2, \ldots, N-2 \\ \psi_0 &= 0, \psi_1 = s \end{aligned} \tag{2.1}$$

where $s \neq 0$ is the free shooting parameter, typically 10^{-5} to avoid huge global maximums in our solutions.

This approach assumes of course, that we already know the energy of the system E, whereas we more often than not want to solve for it! If we take this into account, we can *guess* an energy value for the system then *shoot* from the left boundary to the right boundary with our discretised SE. Using a converging algorithm to make the Energy guessing for us, like the **Bisection Method**, we arrive at an algorithm that solves both for E and $\psi(x)$ iteratively, assuming a quantum number n

If we want to solve for a specific eigenstate or eigenvalue, we employ **Node Counting**. After each guess for the wavefunction, we count its roots (including the origin if n is even). The number of roots, or *nodes* has to be equal to our quantum number, otherwise we are solving for the wrong eigenstate and need to adjust our range for the Bisection Method accordingly.

Thus, we arrive at the following algorithm, as outlined by Joshua Izaac , Jingbo Wang in their book `Computational Quantum Mechanics`:

To numerically solve for the nth eigenstate and the eigenvalue of the time dependent Schrödinger equation:

(1) Estimate lower (E_{min}) and upper (E_{max}) bounds for the energy

(2) Bisect the energy range to determine an estimate for the energy

$$E = \frac{1}{2}(E_{min} + E_{max})$$

(3) Shoot from the left boundary to the right boundary by choosing $s \sim 10^{-5}$ and solving the initial value problem

$$\psi_{j+1}^{(E)} = 2\left[\frac{m(\Delta x)^2}{\hbar^2}(V_j - E) + 1\right]\psi_j^{(E)} - \psi_{j-1}^{(E)}, \quad \psi_0^{(E)} = 0, \quad \psi_1^{(E)} = s$$

(4) Count the number of nodes n' in the solution $\psi^{(E)}$

- (a) If $n' < n$, then set $E_{min} = E$ and return to step (2)
- (b) If $n' > n$, then set $E_{max} = E$ and return to step (2)
- (c) if $n' = n$, then the correct eigenstate is being computed; proceed to step (5)

(5) Bisect the energy eigenvalue bounds to better approximate the boundary condition $\psi_{N-1}^{(E)} \approx 0$

- (a) If $\psi_{N-1}^{(E_{min})}\psi_{N-1}^{(E)} > 0$, then set $E_{min} = E$
- (b) If $\psi_{N-1}^{(E_{min})}\psi_{N-1}^{(E)} < 0$, then set $E_{max} = E$

(6) Check energy eigenvalue convergence:

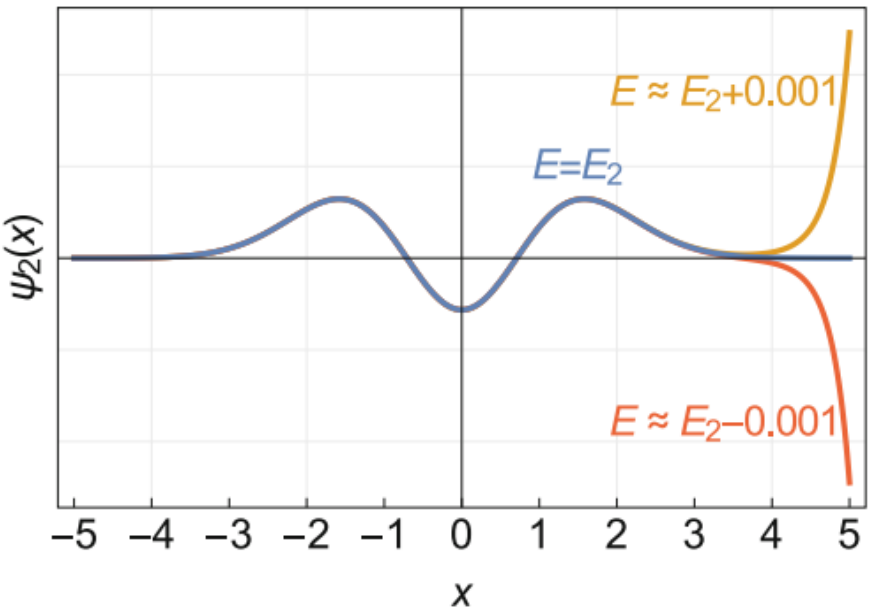
$$|E_{max} - E_{min}| \leq \epsilon \quad \text{or} \quad |\psi_{N-1}^{(E)}| \leq \epsilon \quad \text{for some} \quad \epsilon \ll 1$$

- (a) if the solution is not convergent, return to step (2)
- (b) if the solution is convergent, use numerical integration to normalise the solution $\psi_{N-1}^{(E)}$. This then provides the nth eigenstate $\psi_n(x)$, with energy $E_n = E$.

Figure 2.1: Basic algorithm for numerically solving the SE.

One problem with the shooting method, is that it tends to become *nu-*

merically unstable near the boundary we are shooting towards.



With just a small deviation from the true value of E we end up with great instability. The solution presented by the **Matching Method** is to shoot simultaneously from both boundaries, and match them up to a certain *matching point*.

However, if we want to achieve greater accuracy than the previously mentioned methods, we need to employ the fourth-order method known as **Numerov's Method**.

The **Numerov Method** is given by:

$$\psi_{j+1} = \frac{1}{1 - \frac{h^2}{12} P_{j+1}} \left[2\psi_j \left(1 - \frac{5h^2}{12} P_j \right) - \psi_{j-1} \left(1 + \frac{h^2}{12} P_{j-1} \right) \right] \tag{2.2}$$

where Pj is:

$$P_j = -\frac{2m}{\hbar^2}(E - V_j) \tag{2.3}$$

However, Numerov's Method only affects how *well* we shoot, not how fast the Bisection Method utilized will converge. To fix this, we employ **Cooley's Energy Correction Formula**

Cooley's Energy Correction Formula uses perturbation theory to make a far better approximation each iteration for the energy:

$$\Delta E \approx \frac{\psi_m^{(E_0)*}}{\sum_{j=0}^{N-1} \left|\psi_j^{(E_0)}\right|^2} \left[-\frac{\hbar^2}{2m} \frac{Y_{m+1}^{(E_0)} - 2Y_m^{(E_0)} + Y_{m-1}^{(E_0)}}{(\Delta x)^2} + (V_m - E_0)\psi_m^{(E_0)} \right]$$

Figure 2.2: Cooley's Energy Correction

Utilizing both Numerov and Cooley, we end up with the **Numerov-Cooley Shooting Method**, a highly accurate and rapidly converging numerical method.

Proof for Numerov's Method:

In essence, the Schrödinger Equation is a second-order differential equation with respect to the wavefunction ψ, which can be written in the form of:

$$y''(x) + Q(x)y'(x) + P(x)y(x) = R(x) \tag{2.4}$$

with $Q(x) = 0$, which reduces the equation to:

$$y''(x) + P(x)y(x) = R(x) \tag{2.5}$$

Let $\Delta x \ll 1$ be a small deviation in x. Then, from the Taylor series expansion around Δx:

$$y(x + \Delta x) = y(x) + y'(x)\Delta x + \frac{1}{2!}(\Delta x)^2 y''(x) + \frac{1}{3!}(\Delta x)^3 y'''(x) + \frac{1}{4!}(\Delta x)^4 y''''(x) + \dots \tag{2.6}$$

$$y(x - \Delta x) = y(x) - y'(x)\Delta x + \frac{1}{2!}(\Delta x)^2 y''(x) - \frac{1}{3!}(\Delta x)^3 y'''(x) + \frac{1}{4!}(\Delta x)^4 y''''(x) + \dots \tag{2.7}$$

And summing these two expansions together:

$$y(x+\Delta x) - y(x-\Delta x) = 2y(x) + (\Delta x)^2 y''(x) + \frac{1}{12}(\Delta x)^4 y^{(4)} + \mathcal{O}(\Delta x^6) \tag{2.8}$$

If we approximate $y^{(4)}$ by taking the second derivative y'' and employ the finite differences method:

$$y^{(4)} = \frac{d^2}{dx^2} y''(x) = \frac{(y(x + \Delta x) - 2y(x) + y(x - \Delta x)}{(\Delta x)^2} + \mathcal{O}(\Delta x^2) \tag{2.9}$$

And substituting $y''(x) = R(x) - P(x)y(x)$ from the differential equation into our combined Taylor series expansion, we end up with:

$$\begin{aligned} &y(x+\Delta x)\left(1+\frac{(\Delta x)^2}{12}P(x+\Delta x)\right) - 2y(x)\left(1-\frac{5(\Delta x)^2}{12}P(x+\Delta x)\right) \\ &+ y(x-\Delta x)\left(1+\frac{(\Delta x)^2}{12}P(x-\Delta x)\right) \\ &= \frac{(\Delta x)^2}{12}[R(x+\Delta x)+10R(x)+R(x-\Delta x)] + \mathcal{O}(\Delta x^6) \end{aligned} \tag{2.10}$$

In summary, if $y(x) = \psi(x)$, $P(x) = -2m[V(x) - E]/\hbar^2$ and $R(x) = 0$, with position space discretisation such that $x_{j+1} - x_j = \Delta x$ and use the notation $f(x_j) \equiv f_j$ we achieve the following relation:

$$\psi_{j+1} = \frac{1}{1-\frac{h^2}{12}P_{j+1}}\left[2\psi_j\left(1-\frac{5h^2}{12}P_j\right) - \psi_{j-1}\left(1+\frac{h^2}{12}P_{j-1}\right)\right] \tag{2.11}$$

Proof for Cooley's Energy Correction Formula:

The energy of a stationary state can be calculated by $E = \langle\psi|\hat{H}|\psi\rangle$. If $E^{(0)}$ is our initial guess for the energy of the system, it follows from **perturbation theory** that the deviation between our initial estimate and the actual value can be approximated by:

$$\Delta E = E - E_0 \approx \frac{\langle\psi^{(E_0)}|\hat{H} - E_0|\psi^{(E_0)}\rangle}{\langle\psi^{(E_0)}|\psi^{(E_0)}\rangle} + \mathcal{O}(\Delta E^2) \tag{2.12}$$

where $(E0)$ is the wavefunction solution after applying the Numerov method with E = E0, and the denominator is for normalisation. If Δx is sufficiently small $\Delta x \ll 1$ then we can approximate the integrals in the previous expression as Riemannian sums

$$\langle\psi^{(E_0)}|\psi^{(E_0)}\rangle = \int_{-\inf}^{\inf}\psi^{(E_0)}(x)^*\psi^{(E_0)}(x)dx \approx \Delta x\sum_{j=0}^{N-1}\left|\psi_j^{(E_0)}\right|^2 \tag{2.13}$$

and similarly:

$$\langle\psi^{(E_0)}|\hat{H}-E_0|\psi^{(E_0)}\rangle \approx \Delta x \sum_{j=0}^{N-1} \left|\psi_j^{(E_0)}\right|^2 \left[(\hat{H}-E_0)\psi^{(E_0)}\right]. \tag{2.14}$$

Now, we can replace $(\hat{H}-E_0)\psi^{(E_0)}$ with the Numerov discretisation:

$$\langle\psi^{(E_0)}|\hat{H}-E_0|\psi^{(E_0)}\rangle \approx \Delta x \sum_{j=0}^{N-1} \psi^{(E_0)*}\left[-\frac{\hbar^2}{2m}\frac{Y_{j+1}^{(E_0)}-2Y_j^{(E_0)}+Y_{j-1}^{(E_0)}}{(\Delta x)^2}+(V_j-E_0)\psi^{(E_0)}\right] \tag{2.15}$$

We used the Numerov method for the inward and outward shooting, so the term in the square brackets must vanish for $0 \leq j < m$ and $m < j \leq N-1$, where $x = x_m$ is the matching point. However at the matching point, since our energy $E^{(0)}$ is simply an estimate, the gradient of the inward and outward solutions will not match, leading to a discontinuity in the first derivative – no longer satisfying the Schrödinger equation.

The sum therefore collapses to the single value $j = m$:

$$\langle\psi^{(E_0)}|\hat{H}-E_0|\psi^{(E_0)}\rangle \approx \psi_m^{(E_0)*}\Delta x\left[-\frac{\hbar^2}{2m}\frac{Y_{m+1}^{(E_0)}-2Y_m^{(E_0)}+Y_{m-1}^{(E_0)}}{(\Delta x)^2}+(V_m-E_0)\psi^{(E_0)}\right] \tag{2.16}$$

Substituting this back to equation 2.12 we receive the formula:

$$\Delta E \approx \frac{\psi_m^{(E_0)*}}{\sum_{j=0}^{N-1}\left|\psi_j^{(E_0)}\right|^2}\left[-\frac{\hbar^2}{2m}\frac{Y_{m+1}^{(E_0)}-2Y_m^{(E_0)}+Y_{m-1}^{(E_0)}}{(\Delta x)^2}+(V_m-E_0)\psi^{(E_0)}\right] \tag{2.17}$$

3 Problems Analyzed

Armed with the necessary numerical methods, we can now proceed by looking at the problems we're interested in solving. In particular, we will look at the Particle In a Box Problem, a Finite Potential Square Well, the Quantum Harmonic Oscillator, a Pöschl-Teller Potential and a Double Potential Well. We want to produce the wavefunctions and eigenenergies for these problems.

The Particle In a Box Problem

The Particle In a Box Problem is a widely-known and simple Quantum Mechanics problem to solve. The premise in 1 dimension includes a particle trapped in an infinitely-deep potential well. Analytical solutions to this problem demonstrate clearly the quantization of energy states that the particle can inhabit, a fundamental basis for the quantum world.

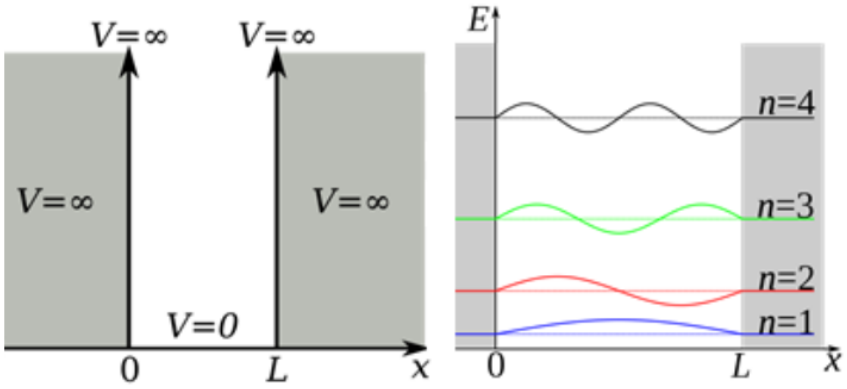


Figure 3.1: Particle In a Box and Wavefunction-Energy solutions.

Finite Potential Square Well

In contrast to the Particle In a Box, or Infinite Potential Well, the Finite Potential Well has a specific height of V_0 for everywhere except $0 \leq x \leq L$,

where it is 0 and L is the width of the well. By studying this problem, we can see the first hints of quantum tunneling, where our particle has a non-zero chance of being 'inside" the walls of the well.

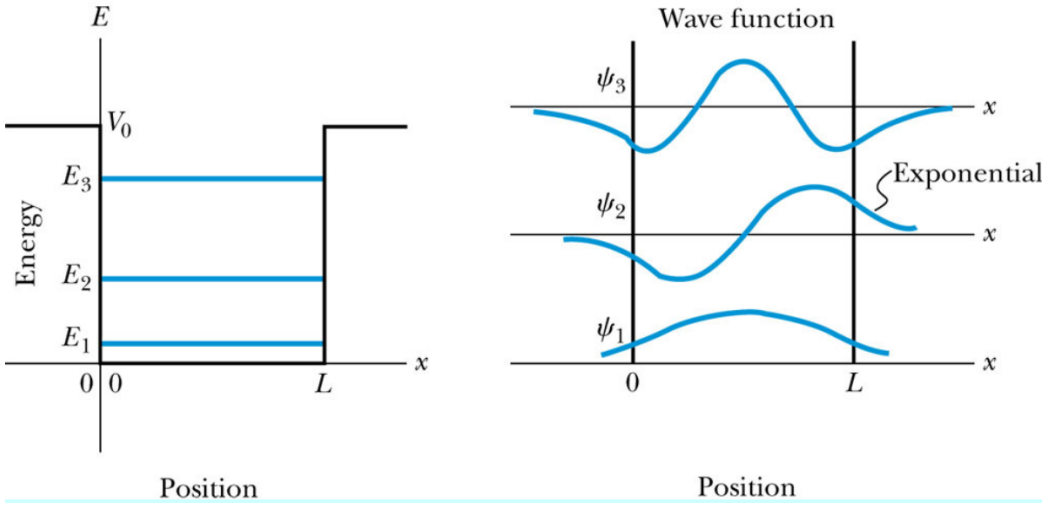


Figure 3.2: Finite Potential Well and Wavefunction solutions.

The wavefunction exponentially decays inside the walls, reaching an amplitude of $1/e$ of its original value with penetration depth equal to δ:

$$\delta = 1/L = \frac{\hbar}{\sqrt{2m(V-E)}} \tag{3.1}$$

Thus, we can say that at a distance $\delta$from the well the wavefunction is *almost* 0, and thus we can approximate the finite potential well by considering an infinite potential well with well width equal to $L + 2\delta$.

The Quantum Harmonic Oscillator

The Quantum Harmonic Oscillator is a fundamental problem in quantum mechanics that models the behavior of a particle under the influence of a quadratic potential energy. Unlike the Particle In a Box and Finite Potential Well, the harmonic oscillator potential is parabolic, resembling the potential energy of a mass-spring system.

The one-dimensional time-independent Schrödinger equation for the quantum harmonic oscillator is given by:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \frac{1}{2}m\omega^2x^2\psi(x) = E\psi(x) \tag{3.2}$$

Here, ω is the angular frequency of the oscillator, E is the energy eigenvalue, and $\psi(x)$ is the wavefunction.

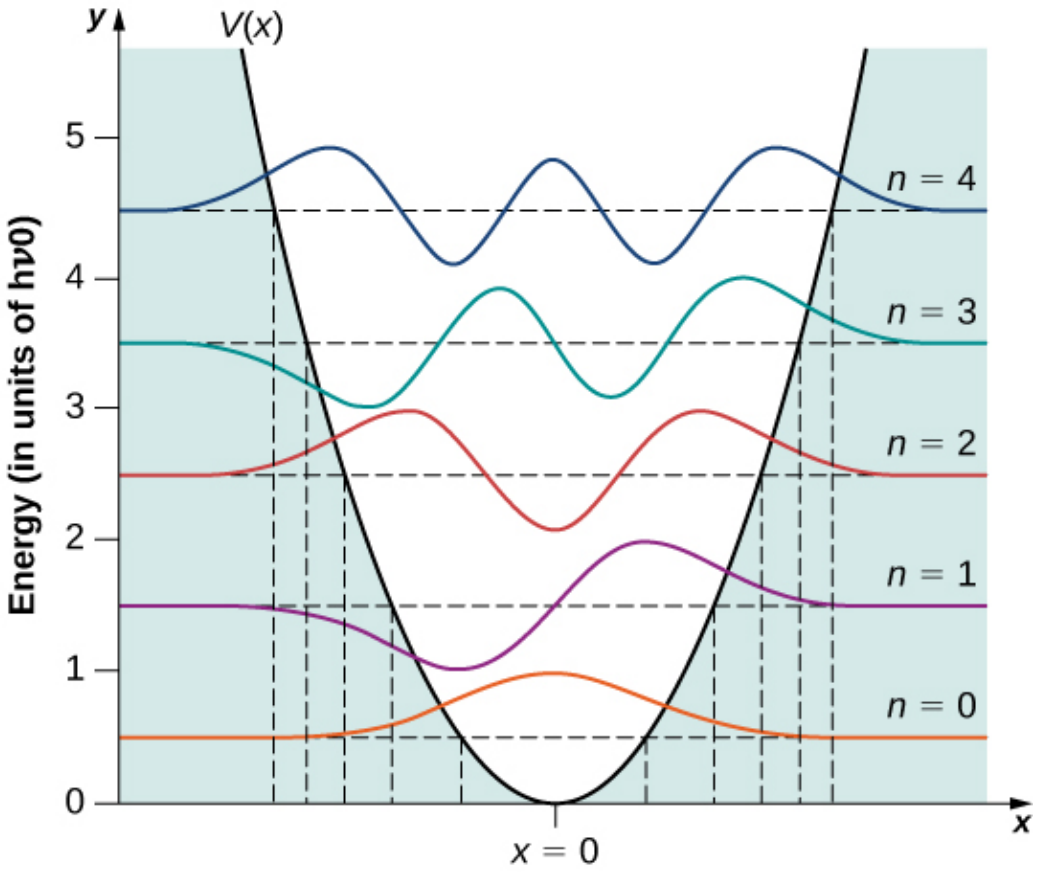


Figure 3.3: Quantum Harmonic Oscillator and Wavefunction solutions.

The solutions to the quantum harmonic oscillator problem reveal quantized energy levels, similarly to the Particle In a Box. However, unlike the rigid quantization of the box, the energy levels of the harmonic oscillator are equally spaced. The wavefunctions are given by Hermite polynomials, and the probability density distributions exhibit a characteristic bell-shaped curve reminiscent of classical harmonic motion.

Pöschl-Teller Potential

The Pöschl-Teller potential is a quantum mechanical potential that is frequently used to model certain physical systems, particularly in the context of molecular and atomic physics. It is named after the physicists Hans Pöschl and Eduard Teller, who introduced it in 1933.

The one-dimensional Pöschl-Teller potential is given by the expression:

$$V(x) = -\frac{V_0}{cosh^2(ax)} \tag{3.3}$$

where V_0 is the depth of the potential well and a is a positive constant determining the width of the well.

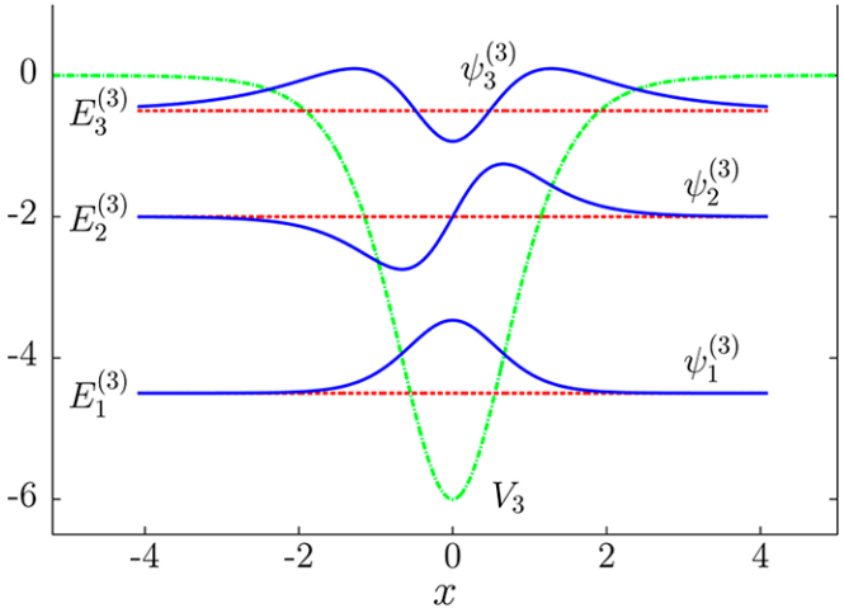


Figure 3.4: Pöschl-Teller Potential and Wavefunction-Energy solutions.

The solutions to this equation involve hypergeometric functions, and the energy eigenvalues are quantized.

Double Finite Potential Well

The Double Finite Potential Well is a quantum mechanical model that expands on the finite potential well with another one text to it. The two wells are separated by a potential barrier, which may be equal to the potential in the outer walls/barriers or not. This potential configuration is often used to study the concept of quantum tunneling and resonant tunneling in quantum mechanics.

The one-dimensional Double Finite Potential Well can be represented by the following potential energy function:

$$V(x) = \begin{cases} V_0, & \text{for } x < L_{11} \\ 0, & \text{for } -L_{11} \leq x \leq -L_{12} \\ V_{0,barrier}, & \text{for } -L_{12} < x < L_{21} \\ 0, & \text{for } L_{21} \leq x \leq L_{22} \\ V_0, & \text{for } L_{22} < x \end{cases} \tag{3.4}$$

where Lij is the start and end of each potential well, and V_0 is the height of the finite potential barriers.

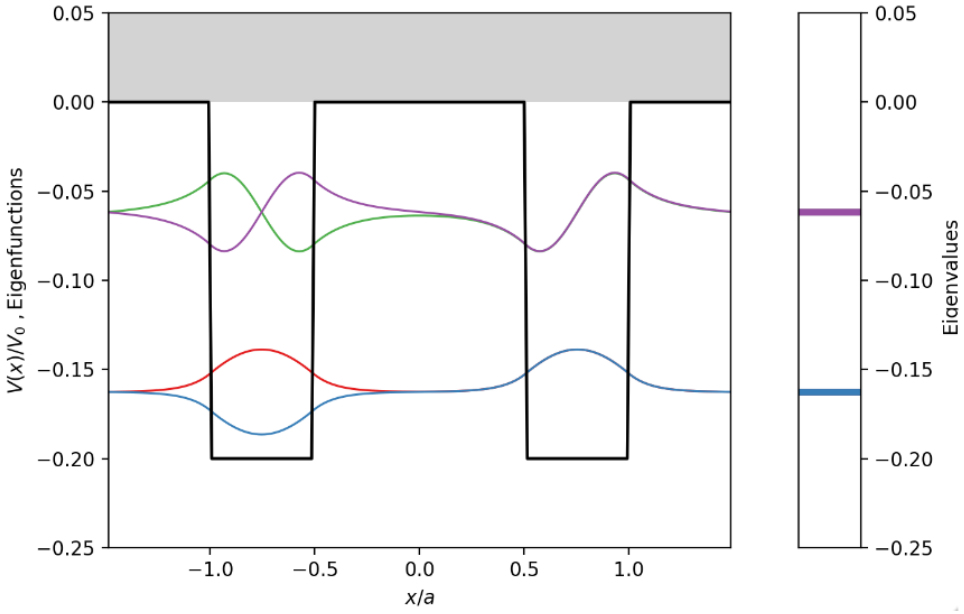


Figure 3.5: Double Potential Well and Wavefunction-Energy solutions.

In the region with zero potential, the solutions are sinusoidal functions, while in the region with finite potential, the solutions are decaying exponentials. The energy eigenvalues are quantized, leading to discrete energy levels for the particle in the Double Finite Potential Well.

This model is particularly interesting because it allows for the possibility of quantum tunneling through the barriers when the energy of the particle is less than the height of the potential barriers ($E < V_0$).

4 Python Code Overview

In this chapter, we're going to go over briefly the program that numerically solves the aforementioned quantum mechanical models. The program uses the `numpy` and `scipy` python libraries to construct the implmentation of the Numerov-Cooley Method, as well as `time` for timing how long the algorithm runs. `Matplotlib.pyplot` is used for plotting the wavefunctions calculated.

Code Structure

For the python code, we utilized an object-oriented approach for better organization of each problem's parameters. The spine of the program is `MainClass`, located in *main.py*. An implementation of a solution for a new problem inserted into the code follows the following steps:

1. Define a function/method that returns list of the model's parameters, like $\hbar$, `well_width` and `x_matching_point` among all necessary others.

```
def qho_problem_parameters(self):
    '''Parameters for the Quantum Harmonic Oscillator'''
    h_bar = 1
    m = 1
    omega = 1
    quant_num = self.get_value_from_user("Quantum number", "Quantum Harmonic Oscillator")
    x_matching_point = 0.0
    parameter_list = [h_bar, m, omega, quant_num, x_matching_point]
    return parameter_list
```

2. Define a function/method that returns the potential values for a `numpy` array of x values passed as input.

```
def potential_harmonic_osc(self, x, mass, omega, *args):
    return 1/2 * mass* (omega*x)**2
```

3. Create a separate file named *NumerovMethodXYZClass.py*, where *XYZ* are the initials of the quantum mechanics model. In there, define

the new class tailor-made for that specific model (correctly defining and passing parameters to the `Potential_function`, for example) by making sure it inherits from the original `NumerovSolverPIB` class, in *NumerovMethodPIBClass.py*.

```
from NumerovMethodPIBClass import NumerovSolverPIB

class NumerovSolverQHO(NumerovSolverPIB):
    def __init__(self, phys_attrib_list, Energy, Potential_function, npoints=1000):...

    def set_phys_parameters(self, phys_attrib_list):...

    def set_x_grid(self, x_max, npoints):...

    def set_potential(self, x, Potential_function):...

    def reset_parameters(self, phys_attrib_list, Energy, Potential_function, npoints=1000):...

    def plot_show(self, matching=False):...
```

4. Create a function/method within the main class that retrieves the parameters, defines the Energy range for bisection, makes the first `Energy_guess` and creates two objects of the corresponding-to-the-problem `NumerovSolver` class. They're then passed to the solver function, and after its done, the `numerov_solver` object contains the eigenfunctions and eigenvalues. They're then plotted by calling the object's respective `show_plot` method.

```
    def quantum_harmonic_oscillator(self, param_list, potential_func, tol=1e-2):
        Npoints = self.get_value_from_user("Number of Points N", "Quantum Harmonic Oscillator")
        h_bar = param_list[0]
        mass = param_list[1]
        omega = param_list[2]
        quant_num = param_list[3]
        x_matching_point = param_list[4]

        # To numerically solve for the nth eigenstate and the eigenvalue of the
        # time dependent Schrodinger equation:
        # Estimate Emin and Emax
        Emin = 0.1
        Emax = 10 *quant_num**2
        # Make an estimation for the energy
        E_first_guess = 1/2 * (Emax + Emin)

        #main solver for wavefunction and eigen energies
        numerov_solver = NumerovSolverQHO(param_list, E_first_guess, potential_func, Npoints)
        #secondary solver for wavefunction of Emin energy
        numerov_solver_secondary = NumerovSolverQHO(param_list, E_first_guess, potential_func, Npoints)
        Energy_guess = E_first_guess
        numerov_solver.run_solver()

        self.solve_quantum_harmonic_oscillator(numerov_solver, numerov_solver_secondary, param_list, E_first_guess, \
                                               Emin, Emax, Energy_guess, potential_func, Npoints, tol)
        self.plot_show(numerov_solver, matching=True)
```

5. Define the solver function/method that implements the algorithm as described earlier 2.1 for solving for the correct eigenstate/eigenvalue.

6. Add the routine to the menu as it exists in the `main_function` function.

```
def main_function(self):
    selection_message = self.get_selection_message()
    user_response = None
    while not user_response or user_response !=-1:
        try:
            user_response = int(input(selection_message))
            if user_response == 1:
                pib_param_list = self.pib_problem_parameters()
                self.particle_in_a_box(pib_param_list, self.potential_0)
            elif user_response == 2:
                fpw_param_list = self.fpw_problem_parameters()
                self.finite_potential_well(fpw_param_list, self.potential_V0)
            elif user_response == 3:
                qho_param_list = self.qho_problem_parameters()
                self.quantum_harmonic_oscillator(qho_param_list, self.potential_harmonic_osc)
            elif user_response == 4:
                ptw_param_list = self.ptw_problem_parameters()
                self.poschl_teller_well(ptw_param_list, self.potential_poschl_teller)
            elif user_response == 5:
                dwp_param_list = self.dwp_problem_parameters()
                self.double_well(dwp_param_list, self.potential_double_well)
            else:
                continue
        except Exception as e:
            print("ERROR:", e)
            continue
    print("Goodbye.")
    return None
```

NumerovSolverPIB

As the original `NumerovSolver` Parent class, from which all other `NumerovSolver`-type classes inherit, we're going to review how it implements the Numerov Method with Matching. This part of the class remains consistent amongst all other child classes.

```
def Numerov_left(self):
    self.psi_left = np.zeros(len(self.x))
    if self.quant_num % 2 == 0: #is even
        self.psi_left[0] = 0
        self.psi_left[1] = (self.get_P(self.Potential[0]) * self.delta**2 + 1) * 1 # derived from discretised SE, with symmetrical potential
        #print(self.psi_left[1])
    else: #is odd
        self.psi_left[0] = 0 #not necessary
        self.psi_left[1] = self.s
    for i in range(1, len(self.x) - 1):
        P_previous = self.get_P(self.Potential[i-1])
        P_current = self.get_P(self.Potential[i])
        P_next = self.get_P(self.Potential[i+1])
        self.psi_left[i+1] = self.calculateNumerov(self.psi_left[i-1], self.psi_left[i], [P_previous, P_current, P_next])
    self.prob_left = np.trapz(np.power(self.psi_left, 2), self.x)
    self.psi_left = self.psi_left / np.sqrt(self.prob_left)
```

```
def Numerov_right(self):
    self.psi_right = np.zeros(len(self.x))
    if self.quant_num % 2 == 0: #is even
        self.psi_right[-1] = 0
        self.psi_right[-2] = (self.get_P(self.Potential[-1]) * self.delta**2 + 1) * -1 # derived from discretised SE, with symmetrical potent
        #print(self.psi_right[-2])
    else: #is odd
        self.psi_right[-1] = 0 #not necessary
        self.psi_right[-2] = self.s
    for i in range(len(self.x) - 2, 0, -1):
        P_previous = self.get_P(self.Potential[i+1])
        P_current = self.get_P(self.Potential[i])
        P_next = self.get_P(self.Potential[i-1])
        self.psi_right[i-1] = self.calculateNumerov(self.psi_right[i+1], self.psi_right[i], [P_previous, P_current, P_next])
    self.prob_right = np.trapz(np.power(self.psi_right, 2), self.x)
    self.psi_right = self.psi_right / np.sqrt(self.prob_right)
```

Figure 4.1: Shooting from the left and right boundaries, up to a x_m matching point.

We can see the implementation of the Matching Method, with shooting from both boundaries, where `get_P` is:

```
def get_P(self, V_i):
    m = self.mass
    h_bar = self.h_bar
    E_g = self.Energy
    return -2 * m / (h_bar ** 2) * (V_i - E_g)
```

Figure 4.2: P as is presented in the Numerov Method.

And `calculateNumerov`:

```
def calculateNumerov(self, psi0, psi1, P_list):
    P0 = P_list[0]
    P1 = P_list[1]
    P2 = P_list[2]
    factor = 1 / (1 + (self.delta**2) / 12 * P2)
    psi2 = factor * (2 * psi1 * (1 - (5 * self.delta**2) / 12 * P1) - psi0 * (1 + (self.delta**2) / 12 * P0))
    return psi2
```

Figure 4.3: The Numerov's Method formula implementation.

Program Execution

To execute the program, it only requires to run *main.py*. The user will be led to a menu of choices, where they'll be prompted to choose a quantum mechanical model to see the wavefunction and energy of.

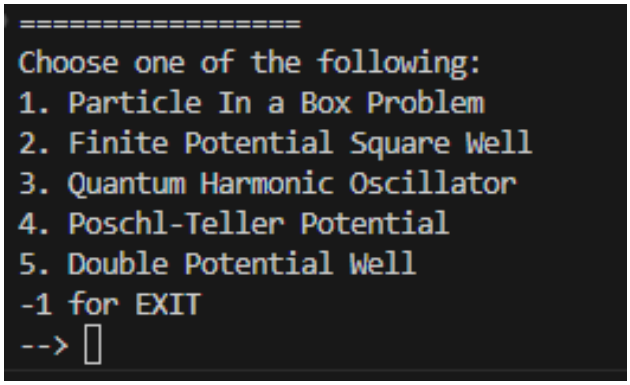


Figure 4.4: The program's menu .

Upon selection of a problem to solve, the user will be prompted to select the quantum state for which we will solve for ($n = 1$ for the ground state, $n = 2$ for the first excited, etc).

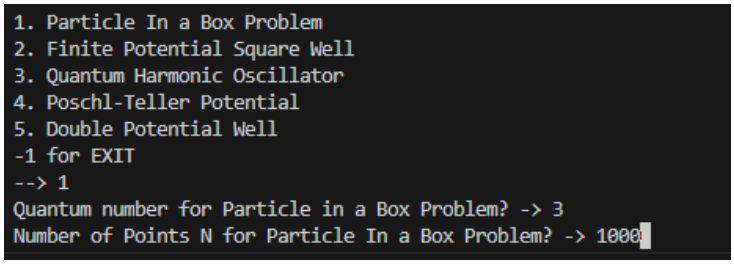


Figure 4.5: Selection of n and `N`.

It is possible when solving for a problem, the parameters given do not respond to the quantum state requested we're solving for. Thus, the user will be prompted to insert new parameters for the quantum mechanical model.

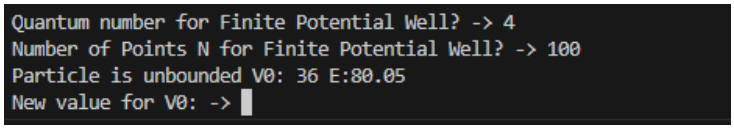


Figure 4.6: The program requires a different V_0 to solve the Schrödinger Equation.

5 Results

We've presented the theory behind the Numerov-Cooley Shooting Method, we've implemented it, and we've used it to solve the aforementioned quantum mechanical models. Some indicative results of the program's capabilities are presented here.

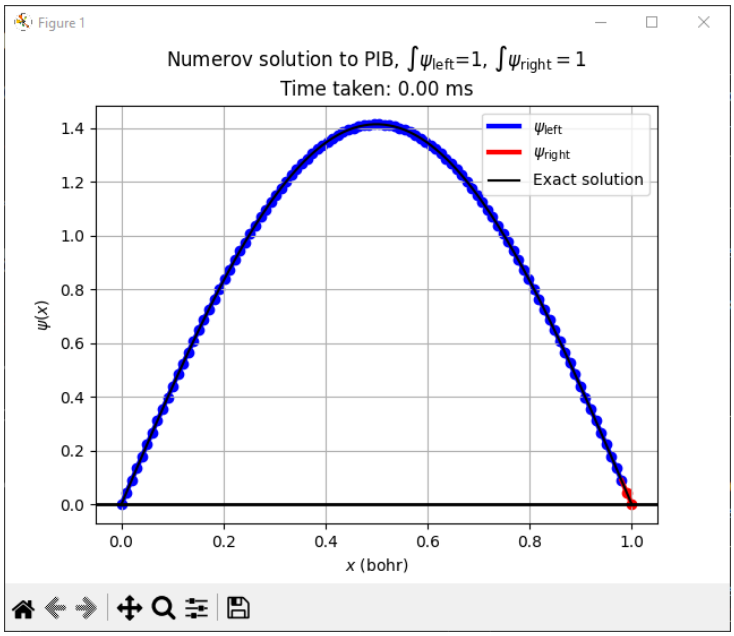


Figure 5.1: Particle In a Box with $n = 1$ and $N = 100$.

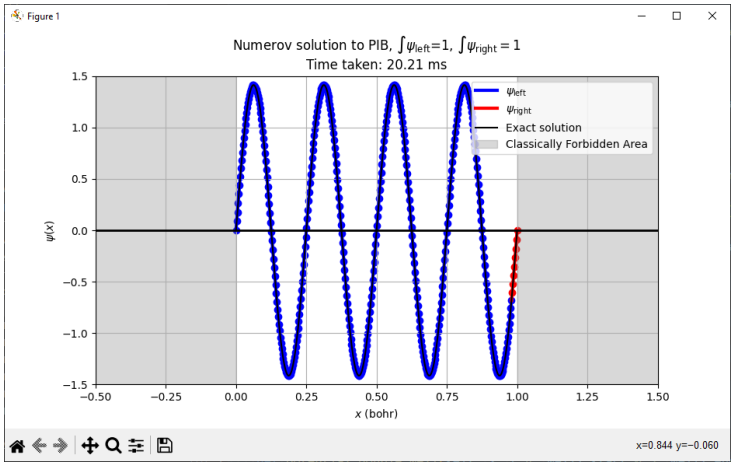


Figure 5.2: Particle In a Box with $n = 8$ and $N = 400$.

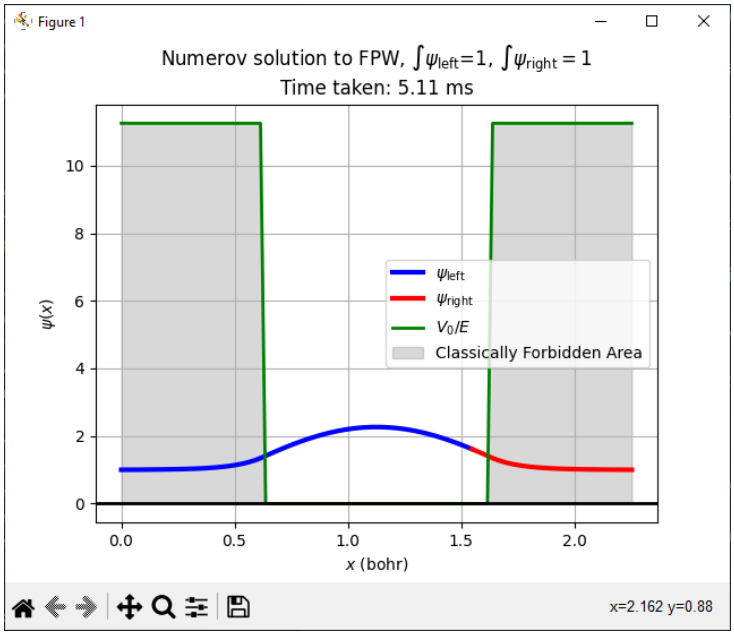


Figure 5.3: Finite Potential Well with $n = 1$ and $N = 100$.

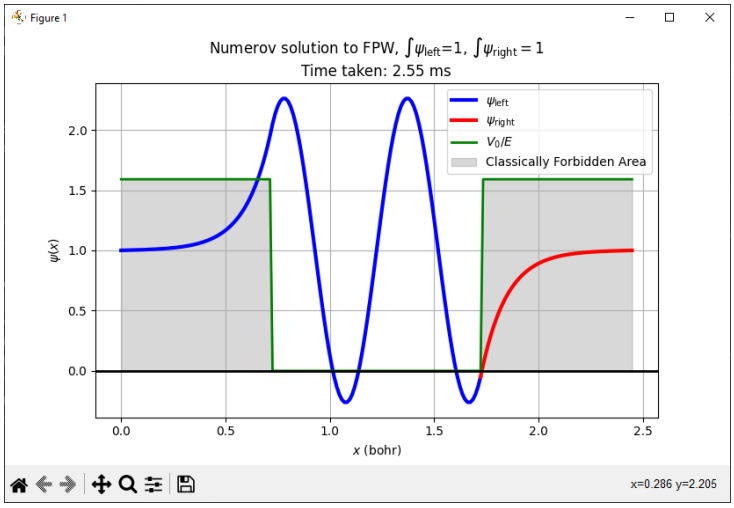


Figure 5.4: Finite Potential Well with $n = 4$ and $N = 200$.

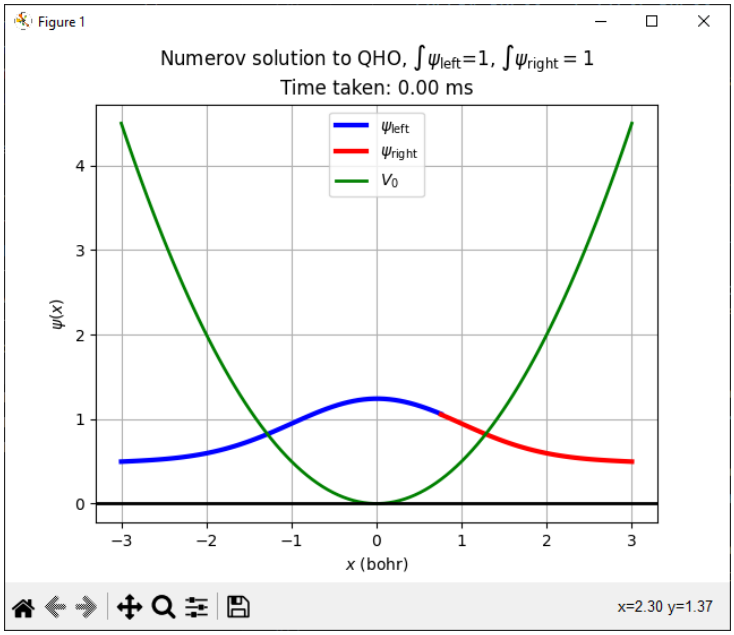


Figure 5.5: Quantum Harmonic Oscillator with quadratic $V(x)$, $n = 1$ and $N = 100$.

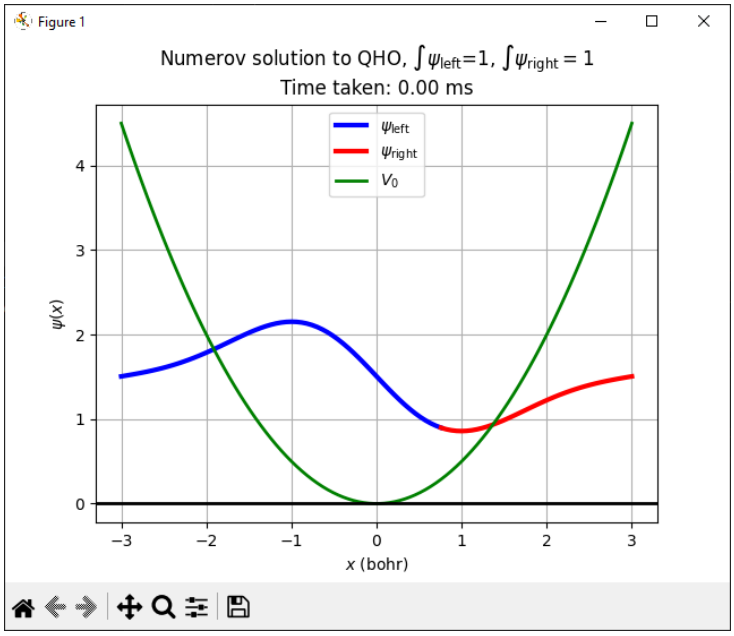


Figure 5.6: Quantum Harmonic Oscillator with quadratic $V(x)$, $n = 2$ and $N = 100$.

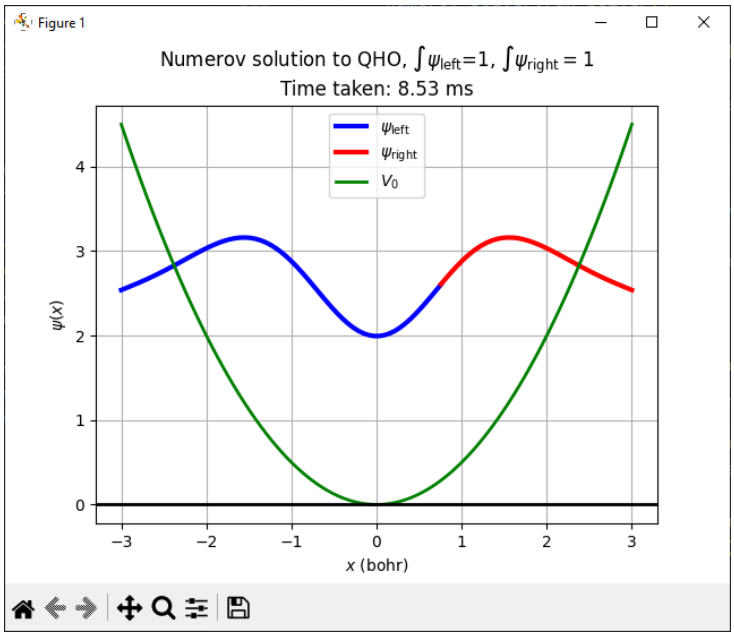


Figure 5.7: Quantum Harmonic Oscillator with quadratic $V(x)$, $n = 3$ and $N = 100$.

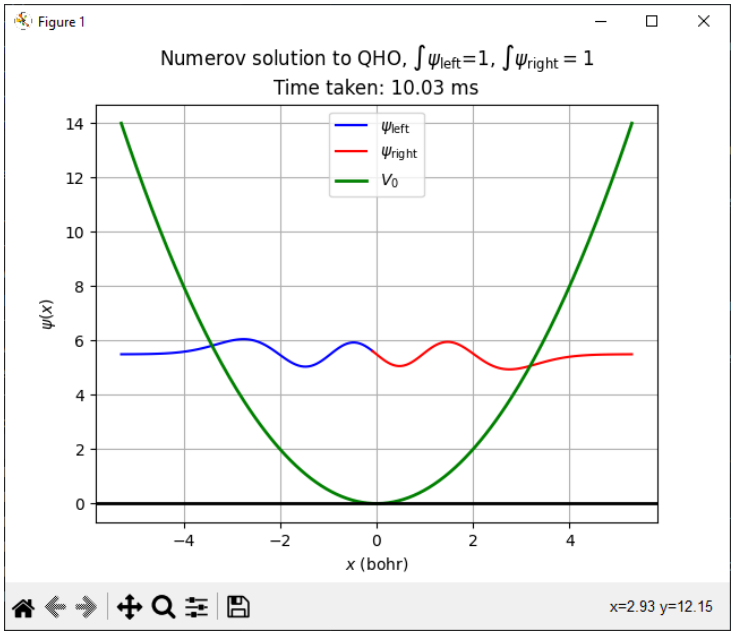


Figure 5.8: Quantum Harmonic Oscillator with quadratic $V(x)$, $n = 6$ and $N = 200$.

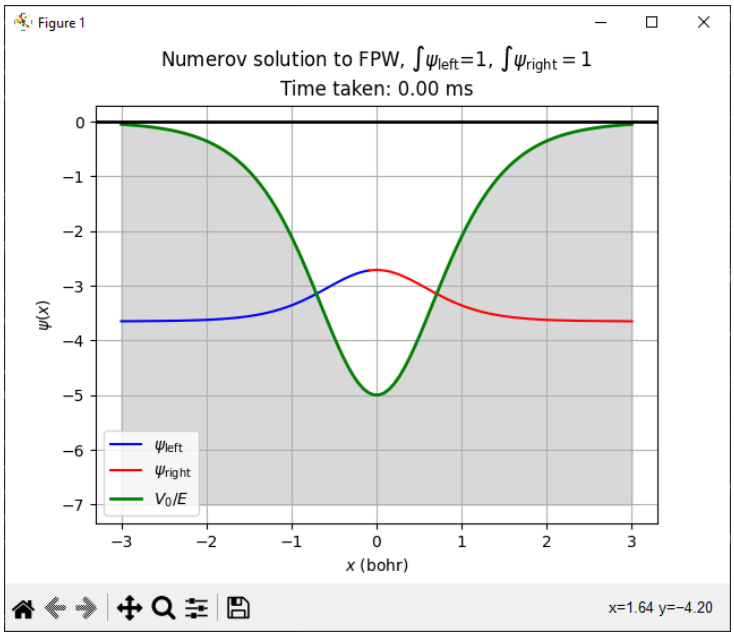


Figure 5.9: Pöschl-Teller Potential Well with $n = 1$ and $N = 100$.

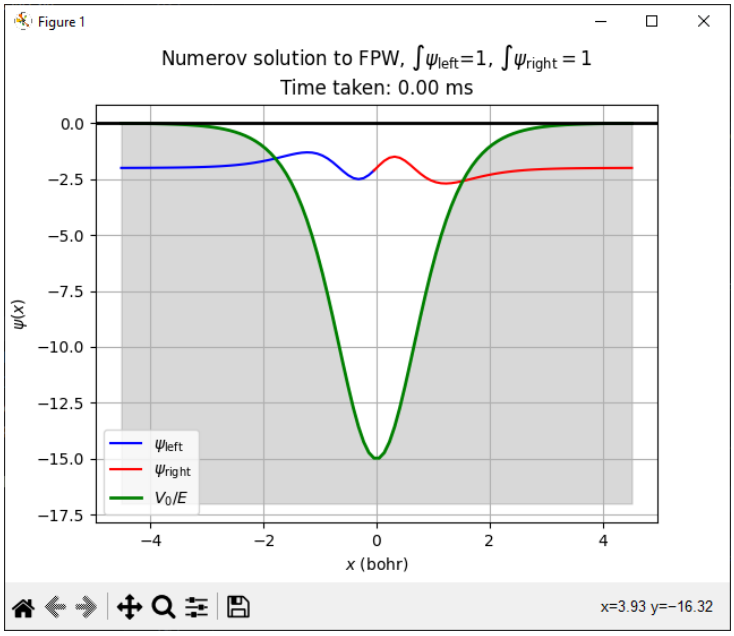


Figure 5.10: Pöschl-Teller Potential Well with $n = 4$ and $N = 100$.

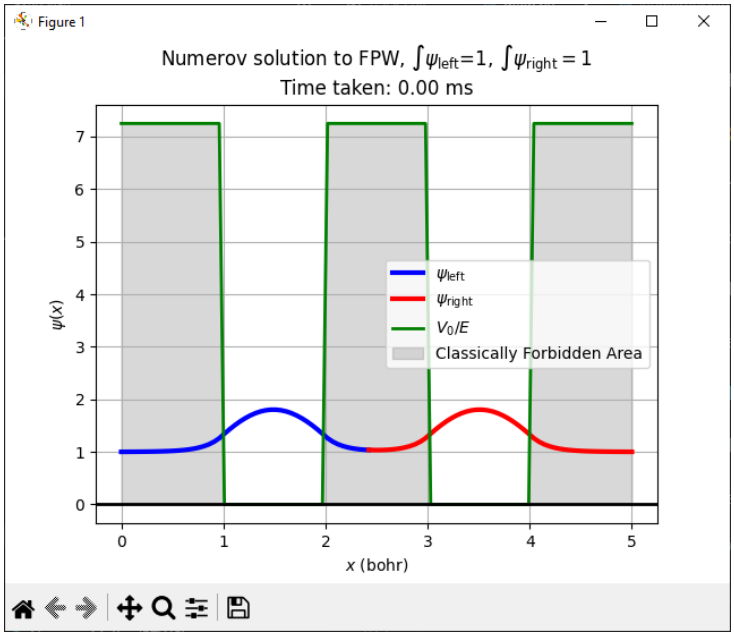


Figure 5.11: Double Finite Potential Well with $n = 1$ and $N = 100$.

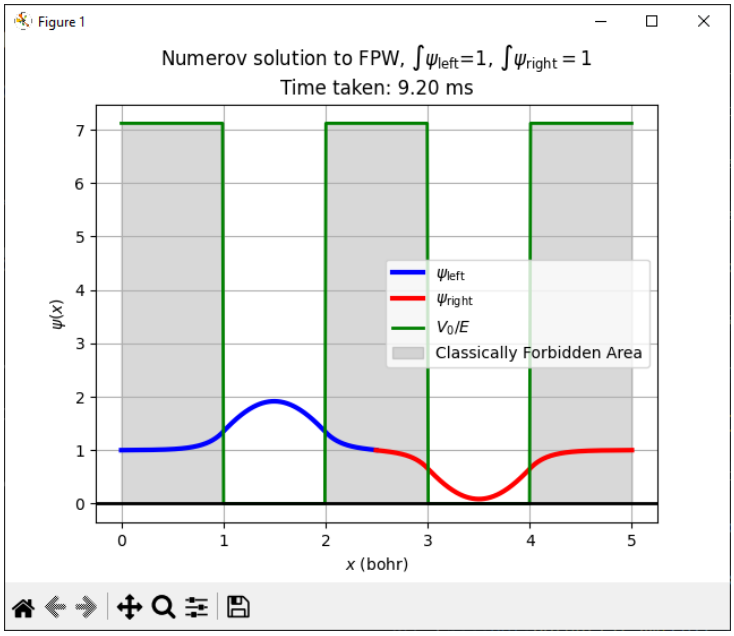


Figure 5.12: Double Finite Potential Well with $n = 2$ and $N = 500$.

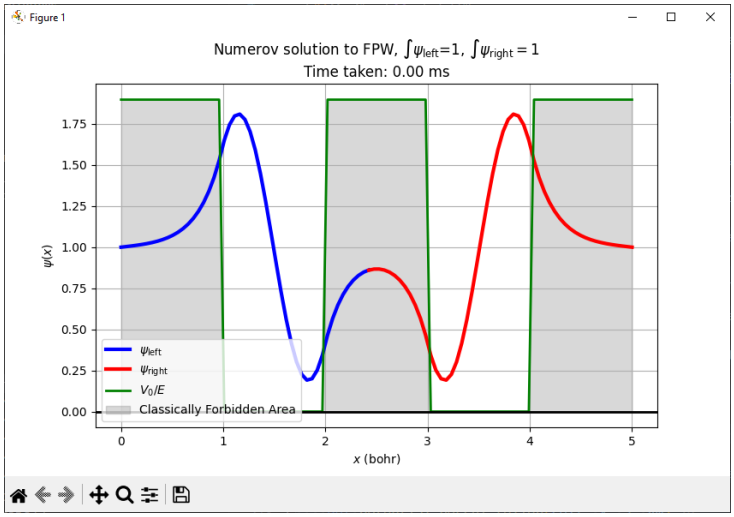


Figure 5.13: Double Finite Potential Well with $n = 3$ and $N = 100$.

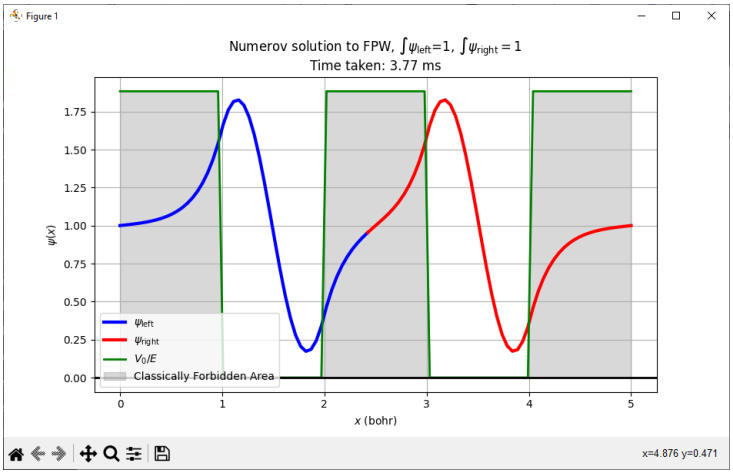


Figure 5.14: Double Finite Potential Well with $n = 4$ and $N = 100$.

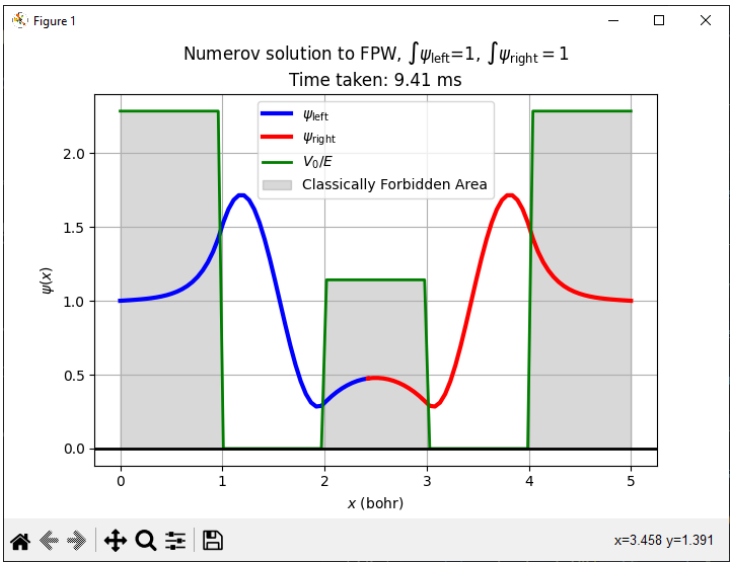


Figure 5.15: Double Finite Potential Well with uneven barrier in-between, $n = 1$ and $N = 100$.

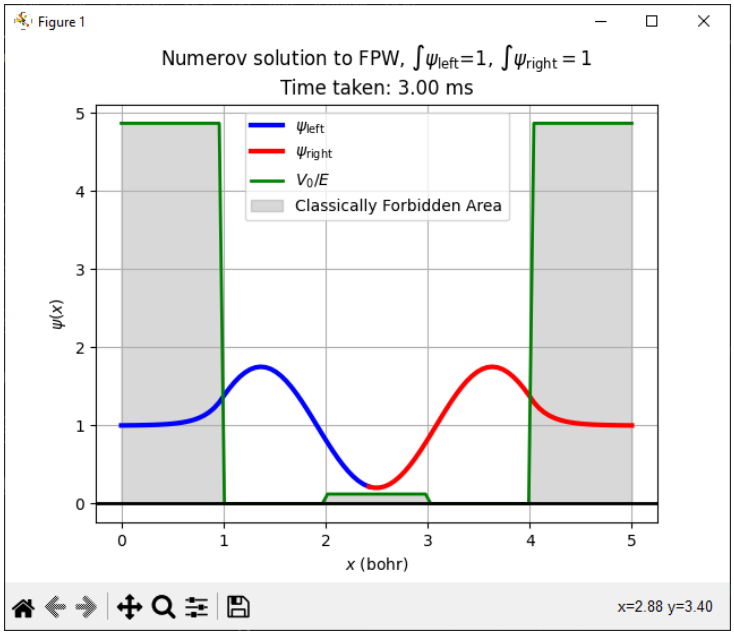


Figure 5.16: Double Finite Potential Well with 0 barrier in-between, $n = 1$ and $N = 100$. Approaches solution of FPW.

6 Weaknesses and Improvements

The program is relatively simple and primordial in nature, and there is a lot of room for improvement. On one front on code organization - there is a lot redudancy that can be relieved from better usage of Class structure and abstraction of methods, on the other in terms of the numerical solutions implemented.

- Detailed analysis of the precision of our implementation. A default tolerance of 10^{-5} is used, and the behavior of the program with higher or lower has not been accounted for.
- Only for the Particle In a Box Problem is there an ability for comparison between the numerical and the analytical solution.
- Multiple numerical methods aside from Numerov-Cooley implemented, for comparison between the numerical solutions and their effectiveness.
- An addition of plotting the probability density as well as the wavefunction - a relatively simple modification.
- Plotting the wavefunctions of all previous states, as well as the one currently solving for, all in a single plot.
- The `reset_parameters` method of the `NumerovSolver` classes is a stopgap to the real need of more detailed setters and getters.
- An archetypal `NumerovSolver` Class, from which all other models will inherit their methods, including `NumerovSolverPIB`, and not from it.
- An additional equation solver using `scipy`, for returning the proper values for V_0 and `well_width` in the Pöschl-Teller Potential.

7 References

1. Izaac, J., & Wang, J. (2018). *Computational Quantum Mechanics*. Springer.

2. Wilfrid Laurier University, Lecture 6.

3. de Forcrand, P., & Werner, P. (2009). *Computational Quantum Physics*. ETH Zürich. Script1.

4. Desroches, Felix. Github. explain_algorithm.pdf

5. Wikipedia. *Shooting Method*. Article

6. Brown, E., & Hernández de la Peña, L. (2018). *A Simplified Pöschl–Teller Potential: An Instructive Exercise for Introductory Quantum Mechanics*. ACS Publications. Paper.

7. Liu Lab Computational Chemistry Tutorials. (2021). *Tutorial 15. Interactive – Solve Schrödinger Eq Numerically with Numerov method*. Webpage