QUEEN'S UNIVERSITY BELFAST

Shooting methods for solving the Schrödinger equation in one dimension

by

Brian Rogers

2022

Abstract

This report investigates the one dimensional time independent Schrödinger equation. The equation will be solved numerically using the shooting method and compared where possible to the analytic solutions. A number of potential energy functions are considered including the infinite potential well and the harmonic potential well. The square potential well followed by potential steps and barriers are also computed. I aim to explain the properties of the eigenenergy solutions and the behaviours of the numerical eigenfunctions across each of the systems under analysis.

Contents

Introduction

This report is loosely split into two parts. Firstly, the structured part has some notable sections.

1. The program to determine the wavefunction of the infinite potential well is contained in the wavefunction class (Code Ref 5) by calling the "eigen energy solver" method.
2. The eigenenergies found by the program are shown in table 4.2. These are compared to the analytical values and will help guide the methodology adopted throughout the report.
3. Illustration of the generalisation of the program to find any wavefunction of level n is shown in section 4.1 and wavefunctions up to $n = 4$ are displayed in figure 4.11. Both the "eigen energy solver" and "plot wavefunction" methods accept any integer value of n.

The unstructured component of this report considers a number of other potentials and their respective eigenenergies and eigenfunctions. Inspiration for these topics has come in part from Eisberg and Resnick (1985). The potentials include:

1. The harmonic potential.
2. The Morse potential.
3. The square potential.
4. Step potentials.
5. Barrier potentials.

It is appropriate to introduce some guidance on how referencing will be made throughout. If I am pointing to a specific part of the report, i.e. section 4.1, one is able to click on the number to bring them to that section. I have adopted a similar practice to reference the code that is used in the report. Any code references will be pointed to by the label "(Code Ref 5)". If necessary, I introduce small code blocks to illustrate uses of the functions and classes written for this project. For example,

```
Wavefunction().eigen_energy_solver(n = 1, V = isw, ...)
```

This is designed to stop repetition of code in the report as many of the methods are re-used across sections.

Theory

2.1 Schrödinger theory

We can state the time independent Schrödinger theory in one dimension

$$(\frac{-\hbar^2}{2m}\frac{d^2}{dx^2} + V)\psi = E\psi \tag{2.1}$$

where $\hbar$ the reduced Planck's constant, m is the mass of the particle and x is the position co-ordinate. V refers to the potential that particle experiences at the position x and E is the total energy of the particle. ψ is the wavefunction of the particle.

Equation 2.1 is a wave equation. At this point I would like to introduce De Broglie's relation that states (DeBroglie (1925))

$$\lambda = \frac{h}{\rho} \tag{2.2}$$

where λ is the wavelength of a wave, h is the Planck's constant and ρ is the momentum of the corresponding particle. This equation is notorious for introducing the concept of wave-particle duality. For our purposes, this equation is consequential as it has the effect that higher energy particles have shorter wavelengths.

2.2 Computational approaches to solving the Schrödinger equation

Re-arranging equation 2.1 we can say

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

Further, by introducing another variable, ω

$$\omega = \frac{d\psi}{dx} \tag{2.3}$$

we can now discretize this equation using a number of different methods (Field (2022)).

2.2.1 Euler Method

The Euler method allows us to write the Schödinger equation in the form.

$$\psi_{i+1} = \psi_i + w\Delta x \tag{2.4}$$

$$\omega_{i+1} = \omega_{i+1} - \frac{2m}{\hbar^2}(E - V)\psi_i \tag{2.5}$$

2.2.2 Runge-Kutta method

Using the fourth order Runge-Kutta method we can write equation 2.1 as

$$\omega_{i+1} = \omega_i + \frac{1}{6}(a_\omega + 2b_\omega + 2c_\omega + d_\omega)\Delta x \tag{2.6}$$

$$\psi_{i+1} = \psi_i + \frac{1}{6}(a_\psi + 2b_\psi + 2c_\psi + d_\psi)\Delta x \tag{2.7}$$

The co-efficient a_ψ is the gradient of ψ evaluated at the point, b_ψ and c_ψ are the gradients evaluated half a step from the point and d_ψ is evaluated one full step from the point. Together they make up a weighted average gradient of ψ.

Methodology

In the previous section I introduced the theoretical underpinnings and now I will consider the programmatic implementation of these methods.

3.1 Shooting method

The shooting method is implemented in conjunction with the bisection method in order to solve the Euler (2.5) and Runge-Kutta (2.7) representations of the Schödinger equation. The "pseudo code" outline is as follows (Field (2022)):

1. Guess an initial energy, E for the quantum level n and use this solve the numerical differential equation (2.7) by "shooting" across the defined region of space.
2. Count the number of intersections of the wavefunction with the x axis. If it is greater than n, set E to be the maximum bound on the energy. Else set E to be the lower bound.
3. Repeat step 2) until the maximum and minimum bounds converge to each other to some threshold value. In my case, I will set the required threshold should be such that difference in the bounds divided their average value is less than 1×10^{-12}.

3.2 Adding potentials

As this project seeks to consider various potentials, I have written a generic solver for the eigenenergies that can be called using

```
Wavefunction().eigen_energy_solver(n = 1, V = V)
```

where V is any potential function. The list of these can be viewed in the appendix A. As most of the potentials use the position of the particle, one can partially initialise the function using functools' partial function. For example, for the harmonic potential I can get the $n = 1$ energy using

```
from functools import partial
Wavefunction().eigen_energy_solver(
    n = 1, V = partial(harmonic, k = 1e4),
    x_min = x_min, x_max = x_max,
    dx = dx, method = "rk4"
)
```

In order to create more complex systems for example the steps and barriers we will see in sections 4.6 and 4.7, I will create an infinite potential well and then put the eigenfunction through the potential. This can be shown as follows:

```
dx = 1e-13 # state step size

steps = np.arange(-3*L, 3*L, dx) # create region

#find the energy of the level
E = wf.eigen_energy_solver(n = n, V = ipw,
    x_min = -3*L, x_max = 3*L, dx = dx)

'''
Initialise the arrays to store
the numerical solutions
'''
w = np.zeros(len(steps))
psi = np.zeros(len(steps))

w[0] = 1
psi[0] = 0

for i, step in enumerate(steps[:-1]):
    '''
    Pass the wavefunction through the barrier defined
    by the function V to get the new system.
    '''
    w[i + 1] , psi[i + 1] =  wf.runge_kutta(w[i], psi[i], E,
    square(x = step, bound_1 = -0.5e-10,
        Vb1 = 0, bound_2 = 0.5e-10,
        Vb2 = 0,
        Vs = 1.5*E),
        dx
    )
```

This approach is more suited to my use cases over other methods as any generic potential can be solved by the function in a time efficient manner. The limitation is that large regions will require large spatial arrays to be used and so may cause issues with memory. In this case it may be possible to optimize the bounds of our region using the potential function itself, but I have found such a step unnecessary at this point considering the use cases of the project.

Although mass can be easily adjusted, I will solely consider the case of an electron in this project. This allows equal comparison of the influence of the potentials to be made throughout the report.

3.3 Normalisation

In order to make comparisons between wavefunctions, they must be normalised. This process requires the integral of the square of the wavefunction to be found within the region

considered and the wavefunction to be divided by the square of this quantity (Field (2022)). I will make use of the scipy.integrate module, trapz to numerically integrate the numerical wavefunctions using the trapezoidal rule. However, more accurate methods exist and so they are possible consideration for improving the accuracy of the solutions.

3.4 Plotting

I will use matplotlib to plot the results of my computation. I have omitted the styling of the plots for brevity in this report but if you would like to see all of the plots made in the process of this project please check this GitHub repository: https://github.com/Birr0/wavefunction/blob/main/schrodigner.ipynb.

Results

4.1 Infinite square well

The infinite square well potential can be described by the following function (Code Ref: 1)

$$V(x) = \begin{cases} \infty, \text{if } x \leq 0 \\ 0, \text{if } 0 < x < L \\ \infty, \text{if } x \geq L \end{cases} \tag{4.1}$$

In my case the width of the well was determined to be $L = 1.0063\mathring{A}$.

4.1.1 Eigenenergy solutions

Using the Euler method gave the following eigenenergy solutions to the equation

n	E_n (eV)	E_{exact} (eV)	ΔE (eV)	t (s)
1	37.22577643630038	37.22577643604761	2.5277×10^{-10}	64.5
2	148.90310574816596	148.90310574419044	39.7552×10^{-10}	60.3
3	335.03198794427198	335.03198792442851	198.4347×10^{-10}	58.8
4	595.61242303897143	595.61242297676176	622.0966×10^{-10}	57.5

Table 4.1: Numerical solutions (E_n) using the Euler method and analytical solutions (E_{exact}) for the first 4 eigenenergies of infinite potential well of width $1.0063\mathring{A}$

The Runge-Kutta solutions to the first four eigenenergies are shown in the table below

n	E_n (eV)	E_{exact} (eV)	ΔE (eV)	t (s)
1	37.22577643605169	37.22577643604761	4.08×10^{-12}	154.6
2	148.90310574432900	148.90310574419044	13.856×10^{-10}	150.0
3	335.03198792466100	335.03198792442851	23.249×10^{-10}	141.1
4	595.61242297701199	595.61242297676176	25.022×10^{-10}	138.4

Table 4.2: Numerical solutions (E_n) using the Runge-Kutta method and analytical solutions (E_{exact}) for the first 4 eigenenergies of infinite potential well of width $1.0063\mathring{A}$

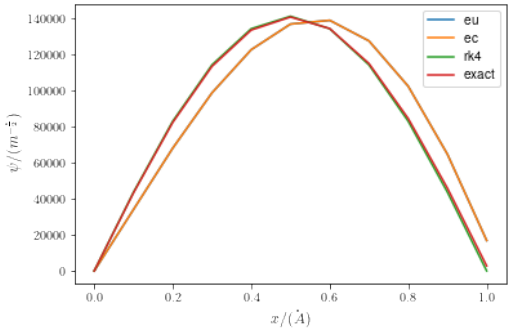
Using the methods provided by the Wavefunction class it is easy to compute the eigenenergy for any value of n such that $n \in \mathbb{Z}$. For example, (Code Ref: 5)

```
E = Wavefunction().eigen_energy_solver(n = n, V = ipw,
    x_min = x_min, x_max = x_max)
```

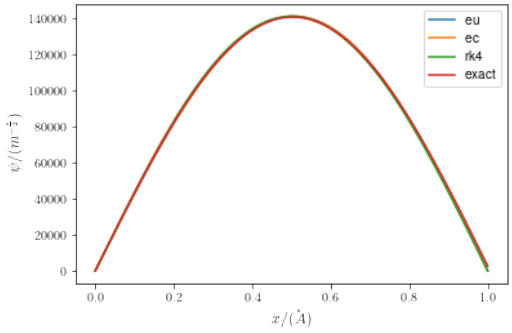
4.1.2 The Wavefunctions - exact and numerical solutions

The code can be extended to search for the first n eigenfunctions and values

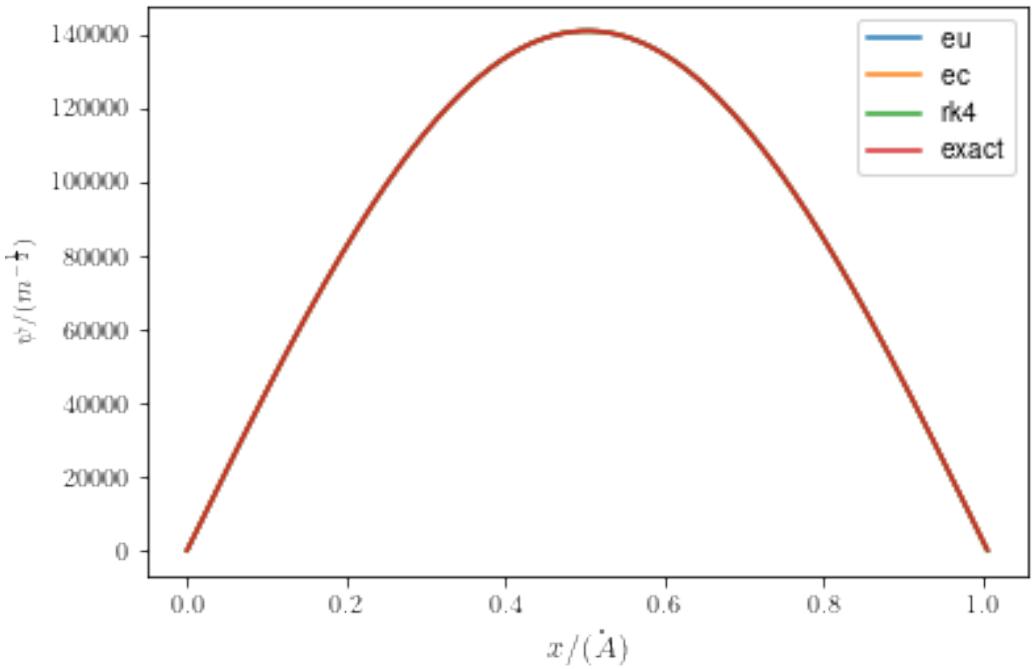
```
steps, psi, E = Wavefunction().plot_eigenfunction(n = n, V = ipw,
    x_min = x_min, x_max = x_max)
```



(a) $\Delta X = 0.1L$

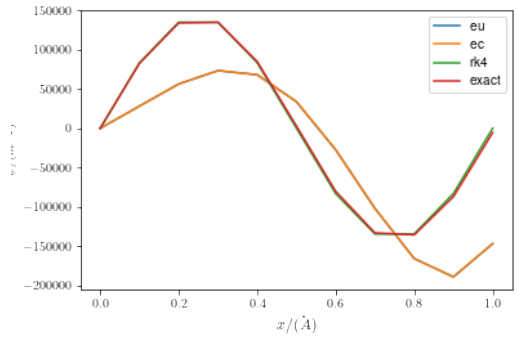


(b) $\Delta X = 0.01L$

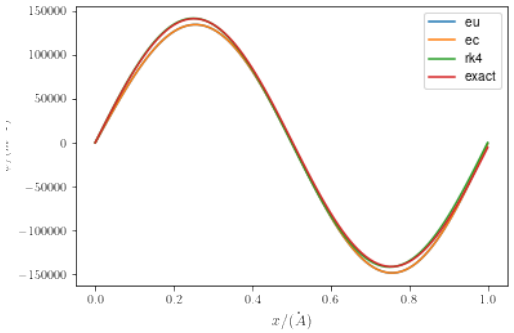


(c) $\Delta X = 0.001L$

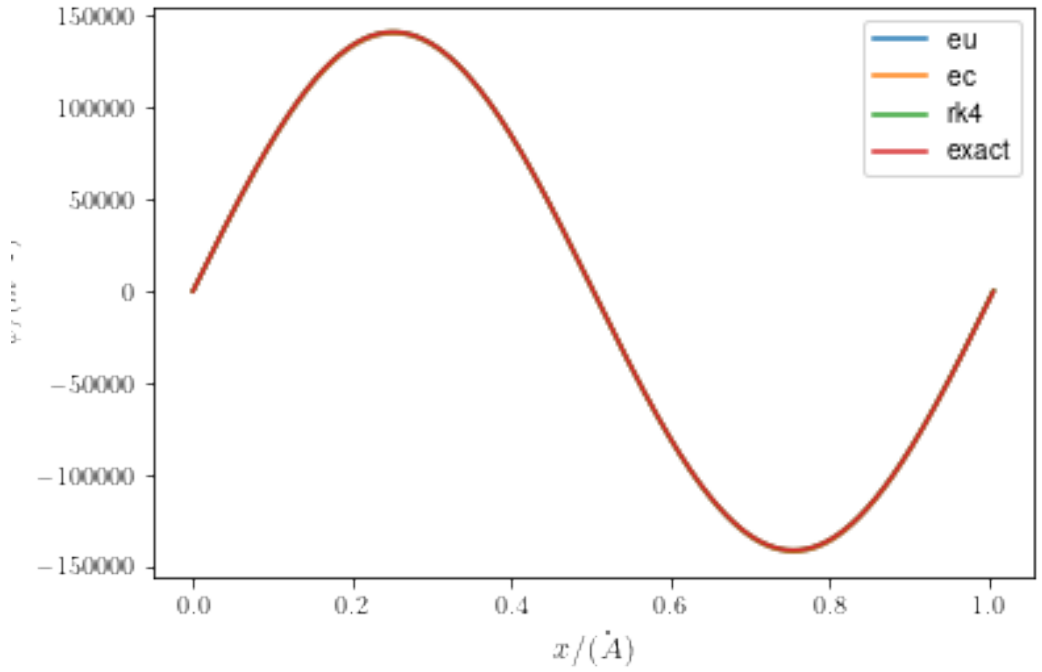
Figure 4.1: Solutions for n = 1 across spatial steps. The Runge-Kutta agrees well with exact solutions throughout however the Euler method requires a smaller spatial step to converge to the exact solution.



(a) $\Delta X = 0.1L$

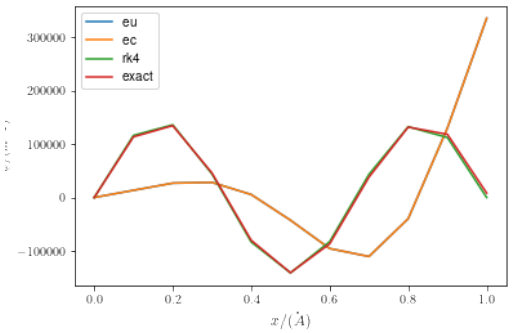


(b) $\Delta X = 0.01L$

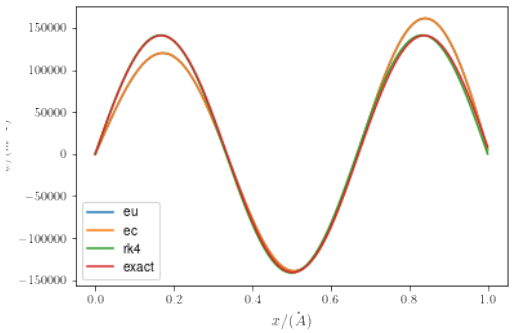


(c) $\Delta X = 0.001L$

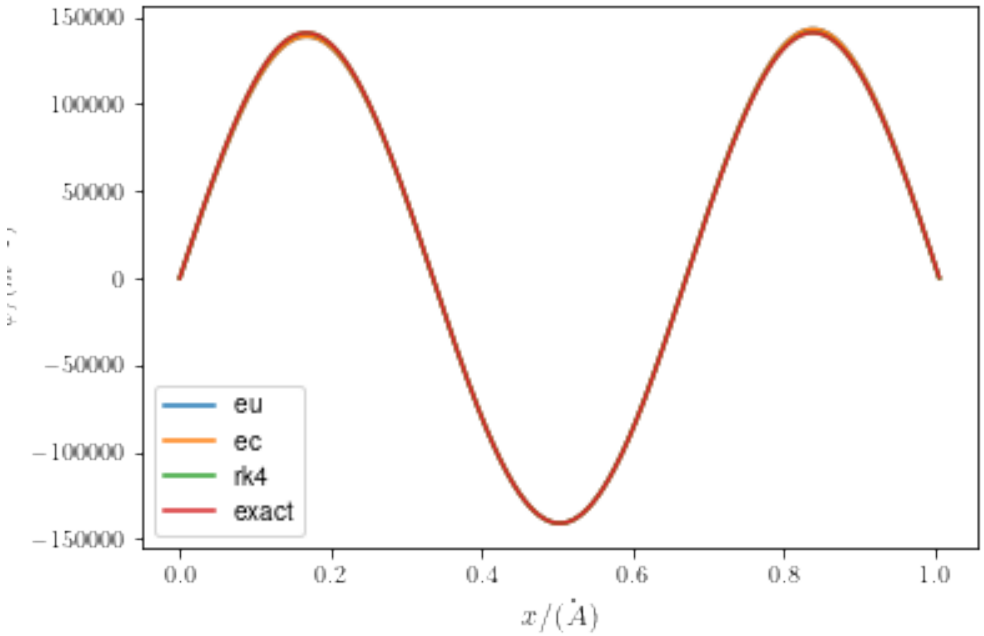
Figure 4.2: Solutions for n = 2 across spatial steps.



(a) $\Delta X = 0.1L$



(b) $\Delta X = 0.01L$



(c) $\Delta X = 0.001L$

Figure 4.3: Solutions for n = 3 across spatial steps.

4.1.3 Visualising the solutions to the infinite potential well

Combining the methodology expressed above produces figure 4.11. This graph allows all important components to be visualised. Firstly the potential region is shaded red where $V = \infty$ and white where $V = 0$. The grey dashed lines indicate the energy levels from $n = 1$ to $n = 4$. The wavefunctions have been imposed on top of their respective energy levels. These have been normalised and a scalar has been applied to fit them on the graph. The relative amplitudes should remain reflective of amplitudes of the eigenfunction solutions. Despite this, the wavefunction has units of $m^{-\frac{1}{2}}$ and so the amplitude should not be read off as an energy value.

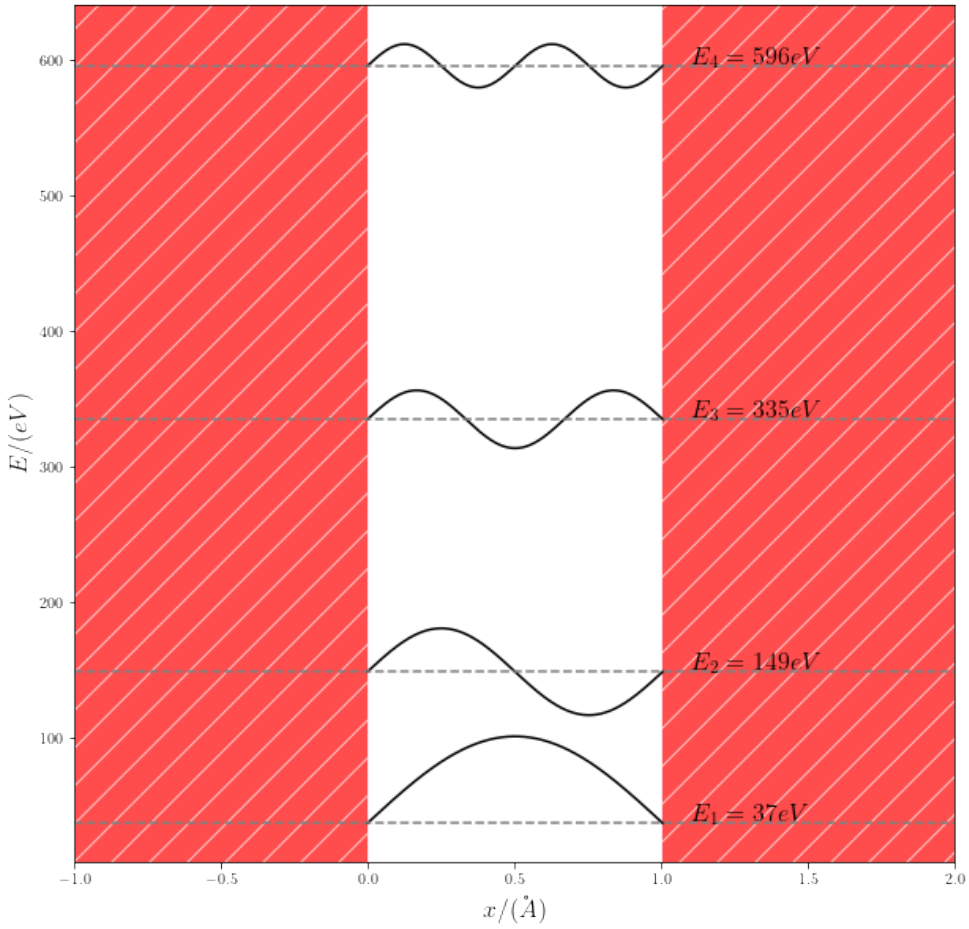


Figure 4.4: Infinite potential well for the first four wavefunctions. Note - a scaling factor has been applied to fit the wavefunctions on the graph.

4.2 Computational methodology revisited

In the previous section, I considered both the Euler method and the Runge-Kutta fourth order method. Both algorithms are implemented in the Wavefunction class (Code Ref: 5). The Euler-Cromer method was also considered during the project but it yield no significant difference to the Euler method, so I will only discuss the Euler method.
Each approach has certain advantages such that a trade off must be made when selecting one to use for the next parts of the project. From the comparison of table 4.1 and 4.2, it is clear that the Euler method is on average approximately 3 times faster than the Runge-Kutta. However, the Runge-Kutta is more accurate using the comparison of the error, the ΔE column of table 4.2. This accuracy is particularly true as the energy level increases, the Runge Kutta has a roughly constant error due it's the smaller global error where $\Delta E \propto \Delta x^3$. Whilst the Euler method error grows with n, it has a gloabal error $\Delta E \propto \Delta x$. Giordano and Nakanishi (2006)

The increased accuracy of the Runge-Kutta especially at smaller spatial steps made it the optimal choice for this project and therefore, all solutions from this point on make use of its method. Furthermore, figures 4.1, 4.2 and 4.3 suggest that for the purposes of plotting that a spatial step of $\Delta x = 1 \times 10^{-13}$ is more than sufficient to achieve an accurate plot. Unless mentioned specifically, I will use this as the spatial step when solving the Runge-Kutta equation for graphical analysis.

4.3 Harmonic potential

The harmonic potential can be shown as follows (Code Ref: 2)

$$V(x) = \frac{1}{2}kx^2 \tag{4.2}$$

where k is the spring constant and x is the displacement from the equilibrium position. The energy of a given level n in the oscillator is

$$E_n = (n + \frac{1}{2})\hbar\omega \tag{4.3}$$

4.3.1 Eigenenergies and Eigenfunctions

For this particular example, I set $k = 1 \times 10^4 kgs^{-2}$. The solutions to the eigenenergies for the Harmonic potential are shown in the following table.

n	E_n (eV)	E_{exact} (eV)	ΔE (eV) (s)
0	34.57139404601106	34.54812587289226	≈ 0.02
1	104.01363068611369	103.64437761867677	≈ 0.37
2	175.29130589985357	172.74062936446126	≈ 2.55

Table 4.3: Eigenenergy numerical (E_n) and analytical (E_{exact}) to the harmonic potential well of range $-L < x < L$.

What is clear from this table is the reduction in accuracy compared to the infinite potential well. One possible source is the reliance of the potential energy function on x^2. As we compute the next value of x the error is compounded by the square. This would suggest that in order to achieve the same accuracy as the infinite potential well I would need to consider a higher order expansion of the Runge-Kutta method or another suitable method that decreases the local error of the numerical solution.

A particularly useful way of visualising the energy levels can be seen in figure 4.5. The equipartion of energy levels as described in equation 4.3 is evident from the physical spacing of the energy levels. This is in constrast to the infinite potential which has spacing proportional to n^2.

We get the entire picture of the harmonic well system by adding in the wavefunctions to figure 4.5. There are similarities to the infinite potential in the shapes of the wavefunctions, such as the number of nodes and anti-nodes. However, it is important to note that the ground state of the harmonic potential well is $n = 0$ where as it $n = 1$ for the infinite well. In addition, the shaded region is no longer forbidden such that the wavefunction can tunnel into the potential.

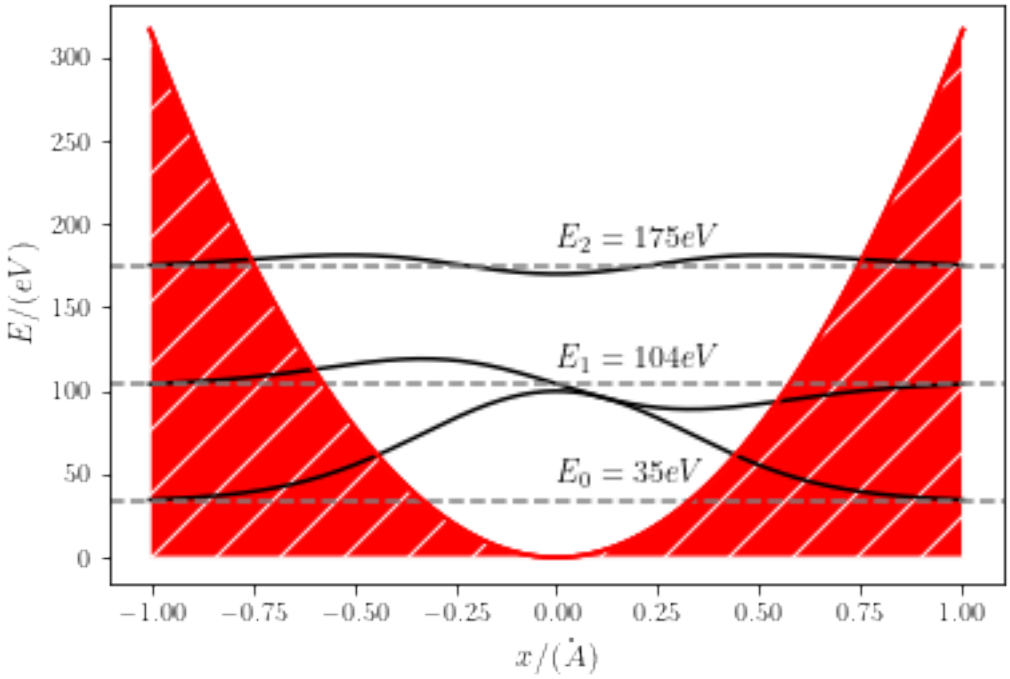


Figure 4.5: The harmonic potential well including the first three eigenenergies and eigenfunctions.

4.4 Morse Potential

The Morse potential can be described by the following function (Code Ref: 3)

$$V(x) = D_e(1 - e^{x-r_e})^2 \tag{4.4}$$

where D_e is the well depth, a is the "width" of the well, x is the displacement and r_e is the equilibrium bond distance.

In this section I will consider a specific well to illustrate with parameters such that; $D_e = 46.7eV$, $a = 1.85\mathring{A}$ and $r_e = 1.2\mathring{A}$.

4.4.1 Eigenenergies and Eigenfunctions

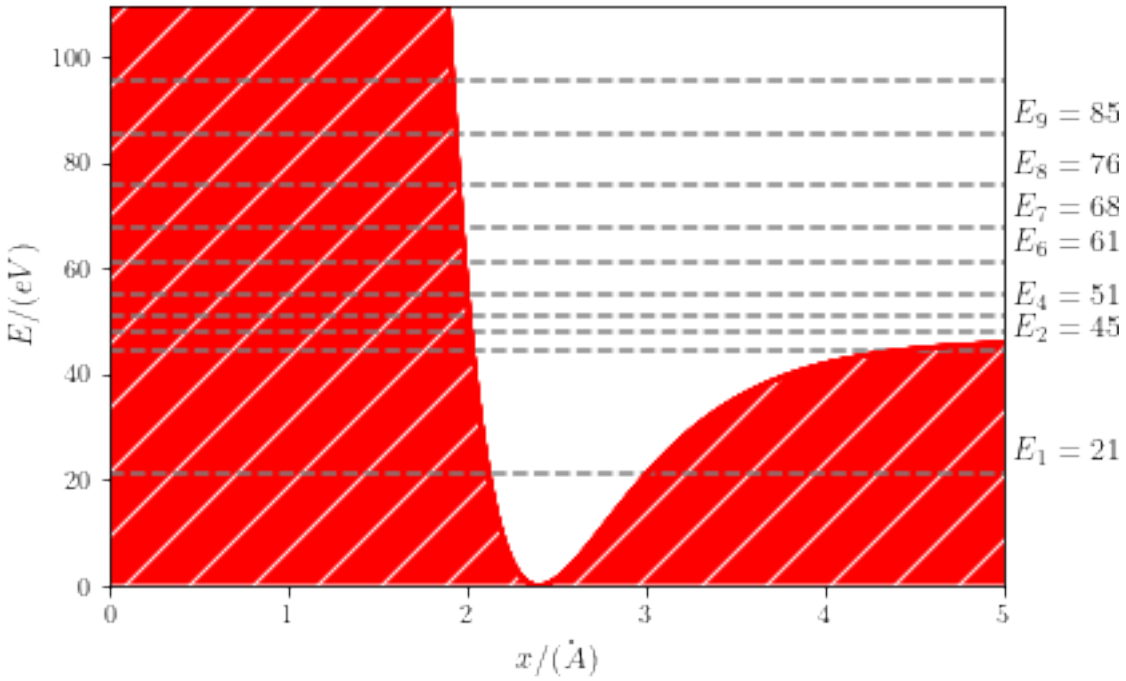


Figure 4.6: Eigenenergies for the Morse potential. Separation decreases as energy approaches the well depth, $D_e = 46.7eV$. After, separation increases in the same manner as the infinite potential well.

The first two wavefunctions are shown in figure 4.7. The Morse shares the same number of nodes as the harmonic potential but as the figure shows the assymetric nature of the potential has "stretched" the wavefunctions. They now have much longer tails as $x \to \infty$ the potential goes to the well depth, D_e.

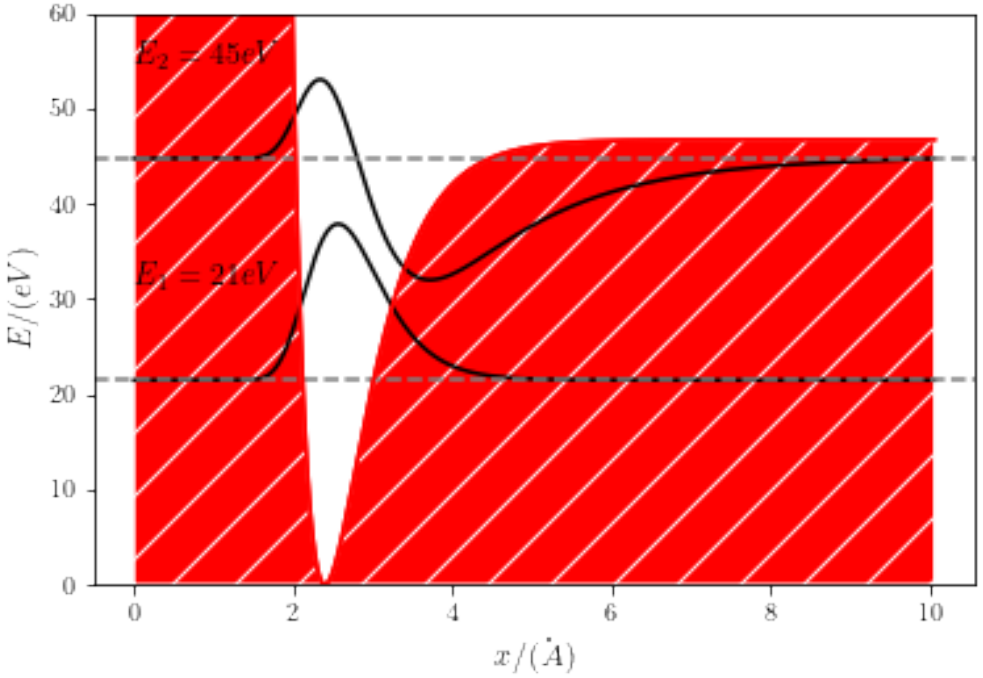


Figure 4.7

4.5 Finite square potential

$$V(x) = \begin{cases} V_0, \text{if } x \leq -L \\ 0, \text{if } -L < x < L \\ V_0, \text{if } x \geq L \end{cases}$$

(4.5)

In this section I will consider the potential defined as follows

$$V(x) = \begin{cases} 250eV, \text{if } x \leq -0.5\AA \\ 0, \text{if } -0.5\AA < x < 0.5\AA \\ 250eV, \text{if } x \geq 0.5\AA \end{cases}$$

(4.6)

which is initialised using

```
V = partial(square, bound_1 = -0.5e-10, Vb1 = 4e-17,
    bound_2 = 0.5e-10, Vb2 = 4e-17, Vs = 0)
```

4.5.1 Eigenenergies and Eigenfunctions

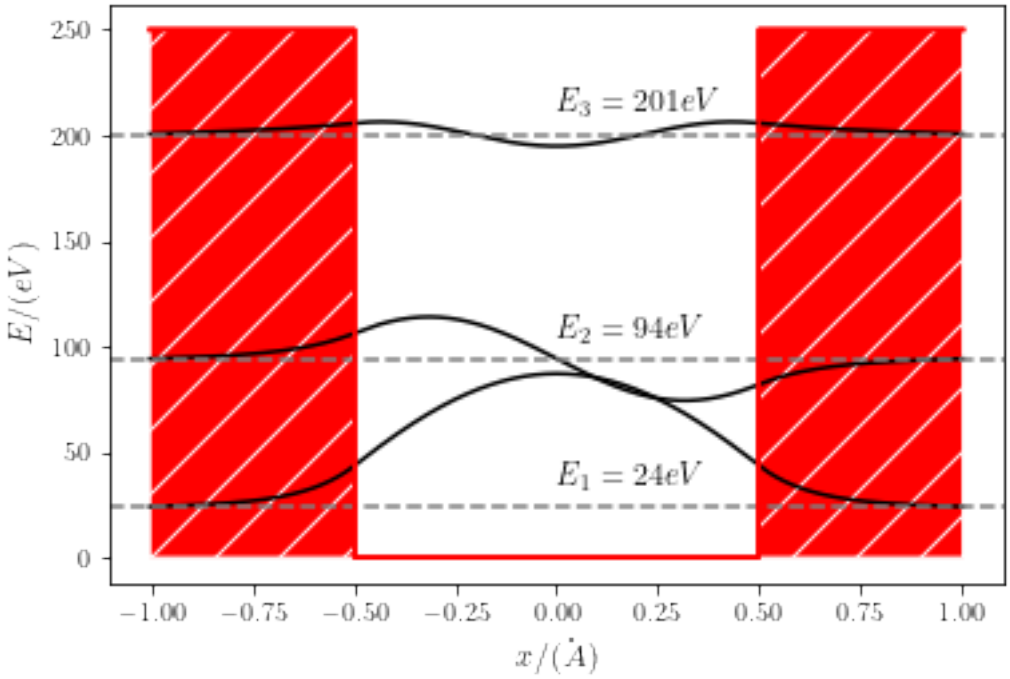


Figure 4.8: Finite potential well for the first three wavefunctions. Note - a scaling factor has been applied to fit the wavefunctions on the graph

Similarities exist with the infinite potential in the number of nodes and anti-nodes per energy level. However, the wavefunction can now tunnel into the walls of the potential, decaying exponentially inside the walls.

4.6 Step potential

The general equation to describe the step potential is (Code Ref: 4)

$$V(x) = \begin{cases} V_0, \text{if } x < x_1 \\ V_1, \text{if } x \geq x_1 \end{cases} \tag{4.7}$$

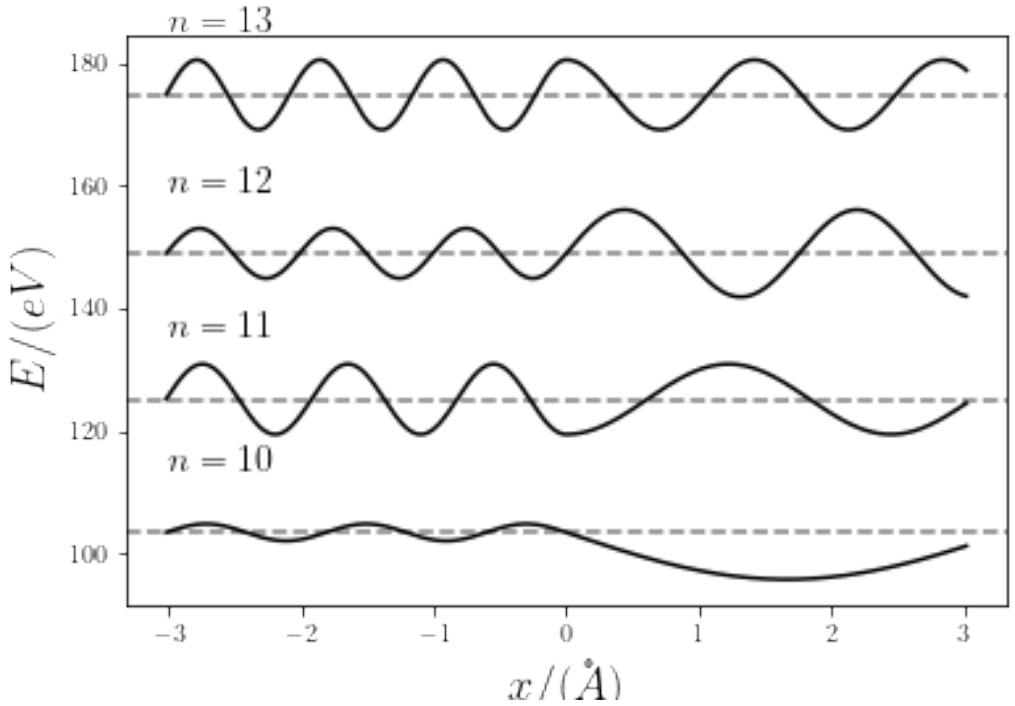
where x_1 is the position of the step and V_1 is the potential past the point x_1.

4.6.1 Step Up

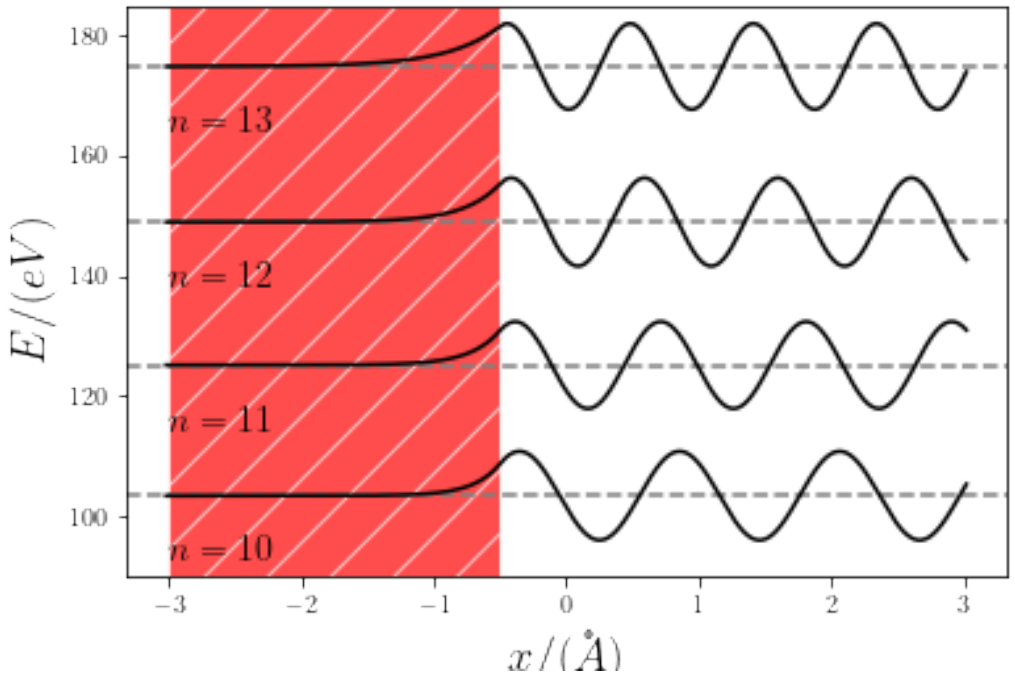
In the Step up potential, the wavefunction moves from a region of lower potential to a higher potential.

In figure 4.9a the wavefunction transitions from a higher potential where $x > 0$ to a lower potential where $x < 0$. Here the potential is less than the energy of the wavefunctions. The effect is visible in the wavelength of the wavefunction. As by De Broglie (2.2), the region of higher potential ($x > 0$) means that the kinetic energy of the electron is reduced and so the wavelength of the wavefunction is increased.

In figure 4.9b, the energy of the wavefunction is less than the potential so that the electron tunnels into the region $x < 0$. Here the wavefunction decays rapidly. For the less energetic levels, i.e $n = 10$ the wavefunction decays more rapidly compared to higher levels such as $n = 13$.



(a) Step Up in which for $x < 0$ $V = 0$ and for $x > 0$ $V = 100eV$. In this case $E_n > V$, therefore height of step is less than axis range in figure and so red shading is emitted.



(b) Step Up in which for $x < 0$,$V = 0$ and for $x > 0$,$V = 1.05E_{13}$. In this case $E_n < V$.

Figure 4.9: The two cases for the Step Up potential

4.6.2 Step down

The step down potential is analogous to the step up potential where $E > V_0$ however, the wavefunction passes in the opposite direction. We can see this behaviour again in figure 4.10 where at $x > -0.5\mathring{A}$ the wavelength decreases and so the potential of the electron must decrease in this region. Specifically, this potential decreases from $0eV$ to $-200eV$ across the step at $x = -0.5\mathring{A}$.

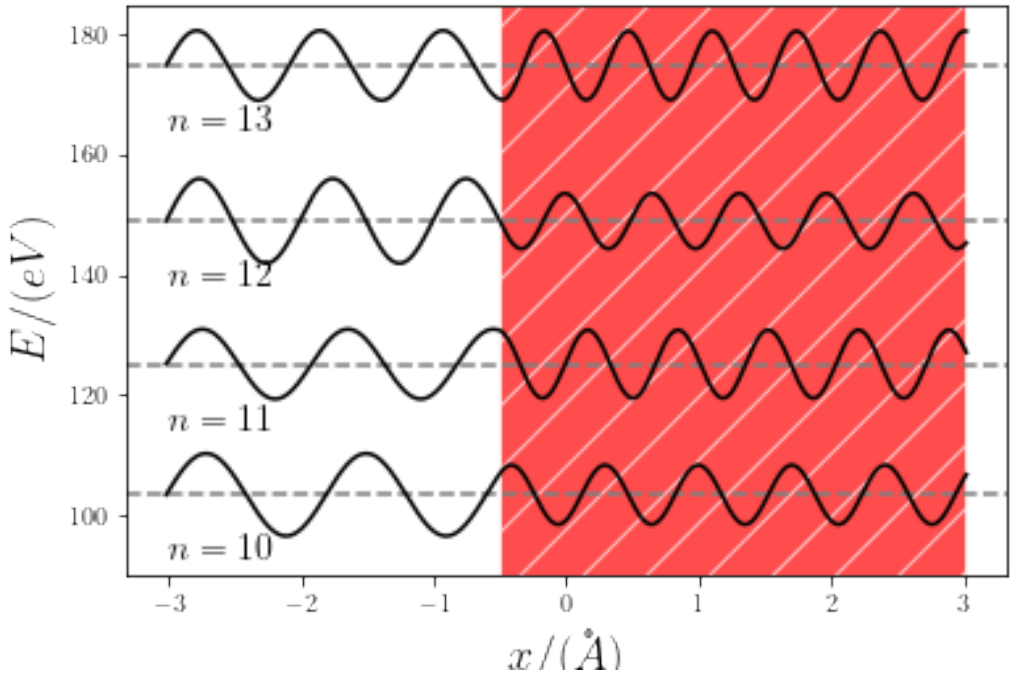


Figure 4.10: The stepdown potential

4.7 Barrier potentials

If a barrier exists in region x_1 to x_2 then potential is described by (Code Ref: 4)

$$V(x) = \begin{cases} V_0, \text{if } x \leq x_1 \\ V_1, \text{if } x_1 < x < x_2 \\ V_2, \text{if } x \geq x_2 \end{cases} \tag{4.8}$$

4.7.1 $E > V_0$

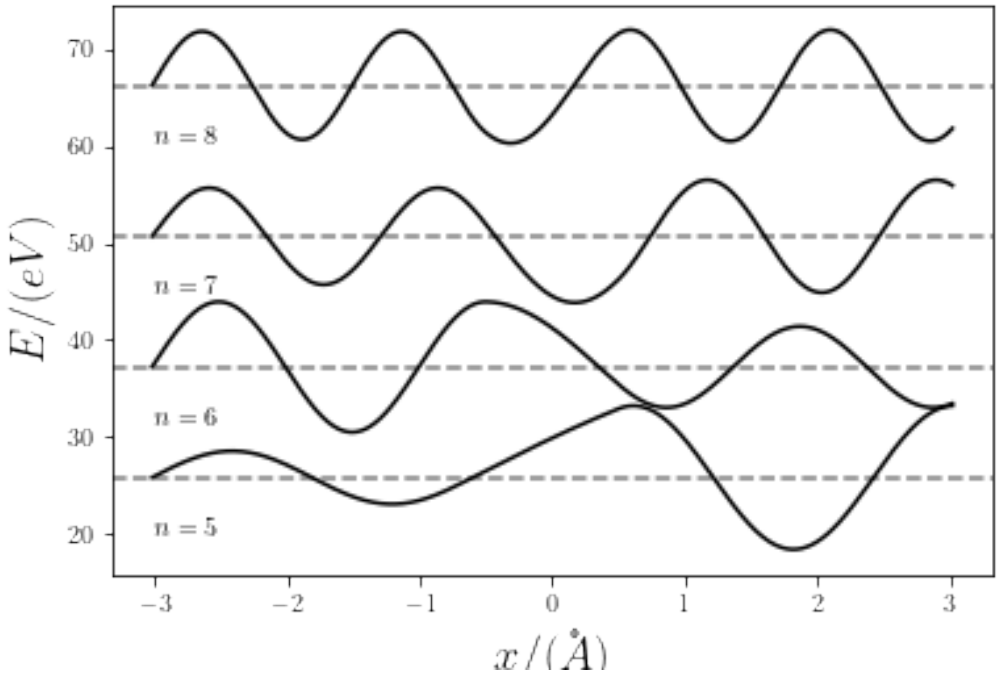


Figure 4.11: Barrier potential in the region $-0.5\mathring{A} < x < 0.5\mathring{A}$. Region is not shaded as barrier height is less than energy of the $n = 5$ eigenfunction.

It is evident, especially for the lower energy wavefunctions that the wavelength increases in the barrier region. This is explained as the increased potential energy slows the electron and so increases the wavefunction's wavelength (De Broglie's relation - 2.2).

4.7.2 $E < V_0$

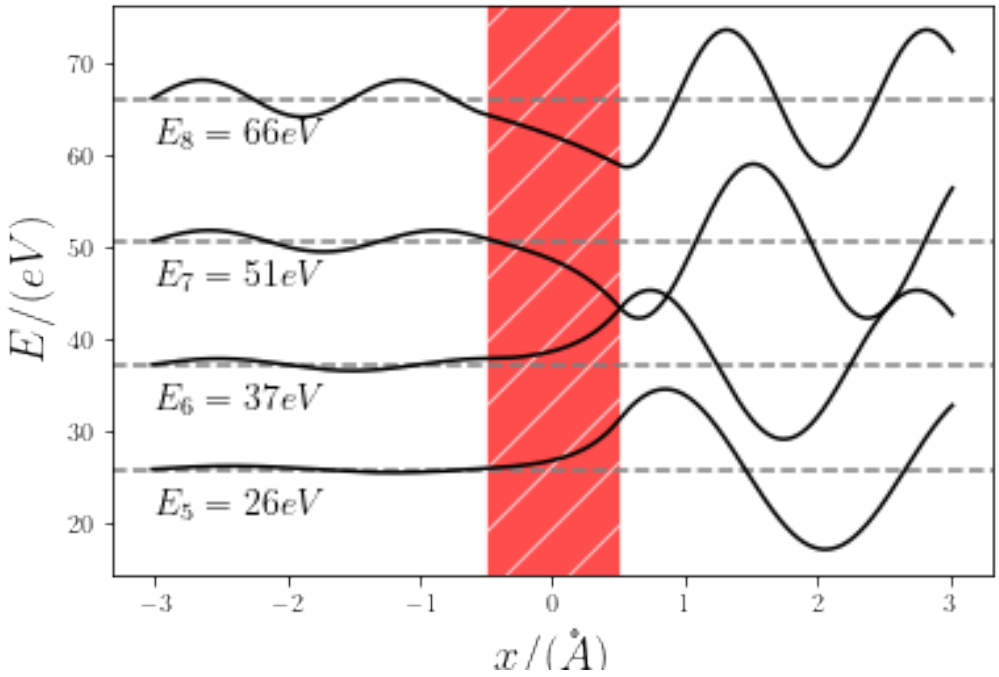


Figure 4.12: Barrier for the wavefunctions $n = 5$ to $n = 8$. The height of the well, shaded in red is $400eV$

In the barrier itself, the wavefunction decays rapidly. We can see despite the fact that the eigenenergies of the waves are less than the potential of the barrier, that there exists a wavefunction in the classically excluded region $x < -0.5\mathring{A}$. It also clear that the amplitudes are greater in the classically excluded region for the higher energy wavefunctions. The physical interpretation of this is that the probability that the particle will be found where $x < -0.5\mathring{A}$ is greater for the higher energy wavefunctions compared to the lower energy wavefunctions.

Conclusion

By using the shooting method, I have solved the Schrödinger in various potentials and systems.

By comparing with the exact analytical solutions to the infinite potential well. I have demonstrated computation of the eigenenergy solutions to an accuracy of at least $10^{-10}eV$ (table 4.2). I illustrated the eigenfunction solutions in graphical form and shown that the methods in the wavefunction class can be used to determine the eigenenergy and eigenfunction for any value of n such that $n \in \mathbb{Z}$.

Using the foundations built from the infinite potential well I then showed how the eigenenergy and eigenfunction solver could be generalised for any potential function, V. I provided a list of the available potential functions in the appendix A. I then implemented these to illustrate their solutions.

For example, where possible I compared the numerical solutions to the exact solutions such as in the harmonic potential in which I discussed some of the limitations with the computational method. I displayed the solutions to the Morse potential in graphical format.

Lastly, I considered more complex systems including step potentials and barriers, illustrating their unique behaviours graphically.

References

DeBroglie, L. (1925), 'Recherches sur la theorie des quanta', *Ann. de Phys.* **10**.

Eisberg, R. and Resnick, R. (1985), *Quantum Physics*, John Wiley and Sons, London.

Field, T. (2022), 'Project guidelines'. Manual used for the project guidelines and methodology for starting project.

Giordano, N. and Nakanishi, H. (2006), *Computational Physics*, Pearson.

Code

```
def ipw(x):
    return 0
```

Listing 1: Infinite potential well - simply returns 0 given bounds.

```
def harmonic(k, x):
    return 0.5 * k * x**2
```

Listing 2: Harmonic potential

```
def morse(x, r_e, a, D_e):
    return D_e * ( 1 - math.exp(-a * (x - r_e) * 1e20) )**2
```

Listing 3: Morse potential

```
def square(bound_1, Vb1, bound_2, Vb2, x, Vs):
    if x <= bound_1:
        return Vb1

    elif x >= bound_2:
        return Vb2

    else:
        return Vs
```

Listing 4: Square potential

```
import numpy as np
from functools import partial

class Wavefunction():
    def __init__(self):
        self.l = (0.5 + 5063/10000) * 1e-10
        self.C = (-2 * m_e)/(h_bar**2)

    def euler(self, w, psi, E, V, dx):
        psi_e = psi + w*dx
        w_e = w + self.C*dx*(E - V)*psi

        return w_e, psi_e

    def euler_cromer(self, w, psi, E, V, dx):
        psi_ec = psi + w*dx
        w_ec = w + self.C*dx*(E - V)*psi
        return w_ec, psi_ec

    def runge_kutta(self, w, psi, E, V, dx):
        L = E - V # L is lagrangian

        apsi = w
        aw = self.C * L * psi

        bpsi = w + (dx/2)*aw
        bw = self.C* L *(psi + (dx/2)*apsi)

        cpsi = w + (dx/2)*bw
        cw = self.C* L *(psi + (dx/2)*bpsi)

        dpsi = w + dx*cw
        dw = self.C * L * (psi + dx*cpsi)

        w_rk = w + (1/6)*(aw + 2*bw + 2*cw + dw)*dx
        psi_rk = psi + (1/6)*(apsi + 2*bpsi + 2*cpsi + dpsi)*dx

        return w_rk, psi_rk

    def ipw_wavefunction(self, x, n):
        return math.sqrt(2/(self.l))*math.sin(n * math.pi * x/(self.l))
```

Listing 5

```
    def eigen_energy_solver(self, n, V, x_min, x_max, dx, method = "rk4"):
        if method not in ["rk4", "eu", "euc"]:
            raise Exception("Not valid method")

        steps = np.arange(x_min, x_max, dx)

        delta = 1

        e_max = 1e-16
        e_min = 1e-25

        converged = False
        counter = 0

        while not converged:
            E = (e_max + e_min) * 0.5

            w = np.zeros(len(steps))
            psi = np.zeros(len(steps))

            w[0] = 1
            psi[0] = 0

            for i, step in enumerate(steps[:-1]):
                if method == "rk4":
                    w[i + 1] , psi[i + 1] =  self.runge_kutta(w[i], psi[i],
                                                E, V(x = step), dx)
                elif method == "eu":
                    w[i + 1] , psi[i + 1] =  self.euler(w[i], psi[i],
                                                E, V(x = step), dx)
                else:
                    w[i + 1] , psi[i + 1] =  self.euler_cromer(w[i], psi[i],
                                                E, V(x = step), dx)
            roots = 0

            for i, _ in enumerate(psi[:-1]):
                if psi[i + 1] * psi[i] < 0: # number of crosses of x axis
                    roots += 1

                if roots >= n:
                    e_max = E
                    break

            if roots < n: # too low
                e_min = E

            delta = e_max - e_min
            avg = (e_max + e_min)/2

            #print(f"Emax = {e_max}, Emin = {e_min} , avg / delta = {delta/avg}")

            counter += 1
            if (delta / avg) < 1e-12: # add proper condition.
                converged = True

        return e_max
```

```
def plot_wavefunction(self, n, V, x_min, x_max, dx):
    steps = np.arange(x_min, x_max, dx)

    w = np.zeros(len(steps))
    psi = np.zeros(len(steps))

    if V == morse:
        E = self.eigen_energy_solver(n, isw, x_min, x_max, dx)
    E = self.eigen_energy_solver(n, V, x_min, x_max, dx)

    w[0] = 1
    psi[0] = 0

    # nice to add potential to this plot
    for i, step in enumerate(steps[:-1]):
        w[i + 1] , psi[i + 1] =  self.runge_kutta(w[i], psi[i], E,
                                    V(x = step), dx)
    #plt.plot(steps/1e-10, psi/1e-13)
    return steps, psi, E
```

Listing 6: Wavefunction class