Numerical Integration of the Schrodinger Equation

L. Nathan. L. Ngqwebo, NGQLIN011 [1]

[1] *University of Cape Town*
Woolsack Drive, Rondebosch, 7701
Cape Town, Western Cape, South Africa

ABSTRACT

A numerical integration approach to solving the time-independent Schrodinger Equation for an electron bound in a ramped infinite square well potential. The first three energy eigenstates are found with their respective energies.

1. INTRODUCTION

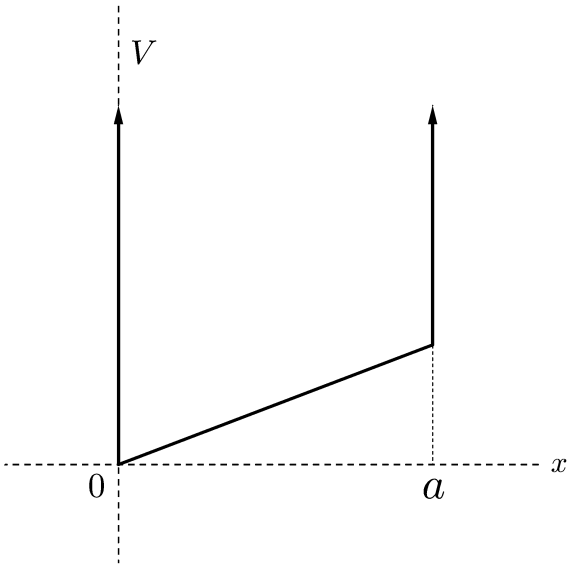

Figure 1. *Ramped Infinite Square Well Potential*

From the Time Independent Schrodinger Equation (TISE) it follows that:

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

$$\frac{d^2\psi}{dt^2} = \frac{-2m}{\hbar^2}\left(E - V(x)\right)\psi(x) \tag{2}$$

$$\tag{3}$$

The solutions of Eq. 2 are expected to be eigenfunctions of the Hamiltonian

$$\hat{H}\psi_i = E_i\psi_i \tag{4}$$

Solving this boundary value problem for the various energy eigenvalues requires that Eq. 2 be numerically integrated from $x = 0$ to $x = a$ and requiring that the $\psi(0) = \psi(a) = 0$. To numerically integrate the time-independent Schrodinger equation (TISE), a second order approximation of the second derivative is employed. Using the Taylor expansions of $\psi(x \pm \Delta x)$

$$\psi(x + \Delta x) = \psi(x) + (\Delta x)\psi'(x) + \frac{(\Delta x)^2}{2!}\boldsymbol{\psi''(x)} + \frac{(\Delta x)^3}{3!}\psi'''(x) + \cdots \tag{5}$$

$$\psi(x - \Delta x) = \psi(x) - (\Delta x)\psi'(x) + \frac{(\Delta x)^2}{2!}\boldsymbol{\psi''(x)} - \frac{(\Delta x)^3}{3!}\psi'''(x) + \cdots \tag{6}$$

By summing Eq. (5) and (6):

$$\psi(x + \Delta x) + \psi(x - \Delta x) = 2\psi(x) + \frac{2(\Delta x)^2}{2!}\boldsymbol{\psi''(x)} + \frac{2(\Delta x)^4}{4!}\psi''''(x) + \cdots$$

$$\psi(x + \Delta x) + \psi(x - \Delta x) - 2\psi(x) = (\Delta x)^2\boldsymbol{\psi''(x)} + \mathcal{O}((\Delta x)^4)$$

$$\boldsymbol{\psi''(x)} = \frac{\psi(x + \Delta x) + \psi(x - \Delta x) - 2\psi(x) - \mathcal{O}((\Delta x)^4)}{(\Delta x)^2}$$

For small values of (Δx), the higher order terms are negligible. It can be expected that, barring any floating-point inaccuracies, this approximation will generally be an overestimate of $\psi(x)$. So, the second derivative approximation becomes:

$$\boldsymbol{\psi''(x)} = \frac{\psi(x + \Delta x) + \psi(x - \Delta x) - 2\psi(x)}{(\Delta x)^2}$$

or

$$\boldsymbol{\psi''_i} = \frac{\psi_{i+1} + \psi_{i-1} - 2\psi_i}{(\Delta x)^2} \tag{7}$$

And the TISE becomes:

$$\boldsymbol{\psi_{i+1}} = \psi_i - \psi_{i-1} + g\left(E, x_i\right)(\Delta x)^2\psi_i \tag{8}$$

$$g\left(E, x_i\right) = \frac{-2mc^2}{(\hbar c)^2}\left(E - V(x)\right) \tag{9}$$

Table 1. *Table of System Constants*

Quantity	Description	Value	Units
a	*well width*	3	[nm]
b	*ramp gradient*	0.5	[eVnm^{-1}]
mc^2	*electron's reduced mass*	0.511	[MeV]
$\hbar c$	*physical constant*	197.3	[eV·nm]

TASK 1: ENERGY EIGENVALUES

The main driver for the eigenvalue estimates was the function `generatePsiEndpoint()` which used the shooting method to iterate ψ from $x = 0$ to $x = a$, keeping track of only three values:

Listing 1. Function Definition of ψ Endpoint Generator

```
def generatePsiEndpoint(E_guess:float) -> float:
    psi_p = 0       # psi_previous: psi(x-1)
    psi_c = 1e-6    # psi_current: psi(x)

    # Numerical Integration (carrying only 3 values)
    for pos in x[2:]:
        psi_n = 2 * psi_c - psi_p + dx**2 * g(E_guess, pos) * psi_c
        psi_p, psi_c = psi_c, psi_n

    return psi_n
```

with this function, wave function values at the endpoint $\psi_E(a)$ could be plotted against various energy eigenvalue candidates to produce Figure 2 Which gives a good indication of the locations of the energy eigenvalues. A more fun exercise in locating eigenvalues is explored in Appendix B. Thereafter, finding the precise estimates was a matter of initialising the `secant()` method with two initial energies close to each of the first three energy eigenvalues, as shown in Listing 2

Listing 2. Secant Method Root-FInding Implementation

```
def secantDescent(func:callable, E_0:float, E_1:float) -> float:
    max_iter = 30   # Maximum number of iterations
    iter = 0
    threshold = 1e-13
    E_p = E_0
    E_c = E_1
    f_p = func(E_p)
    f_c = func(E_c)
    E_n = None

    while abs(E_p - E_c) >= threshold:
        if abs(E_p - E_c) <= threshold:
            print(f'Root␣Constrained␣After␣{iter}␣iterations')
            break
        E_n = (f_c * E_p - f_p * E_c) / (f_c - f_p)
        f_n = func(E_n)
        # Tuple assignment update of variables
```

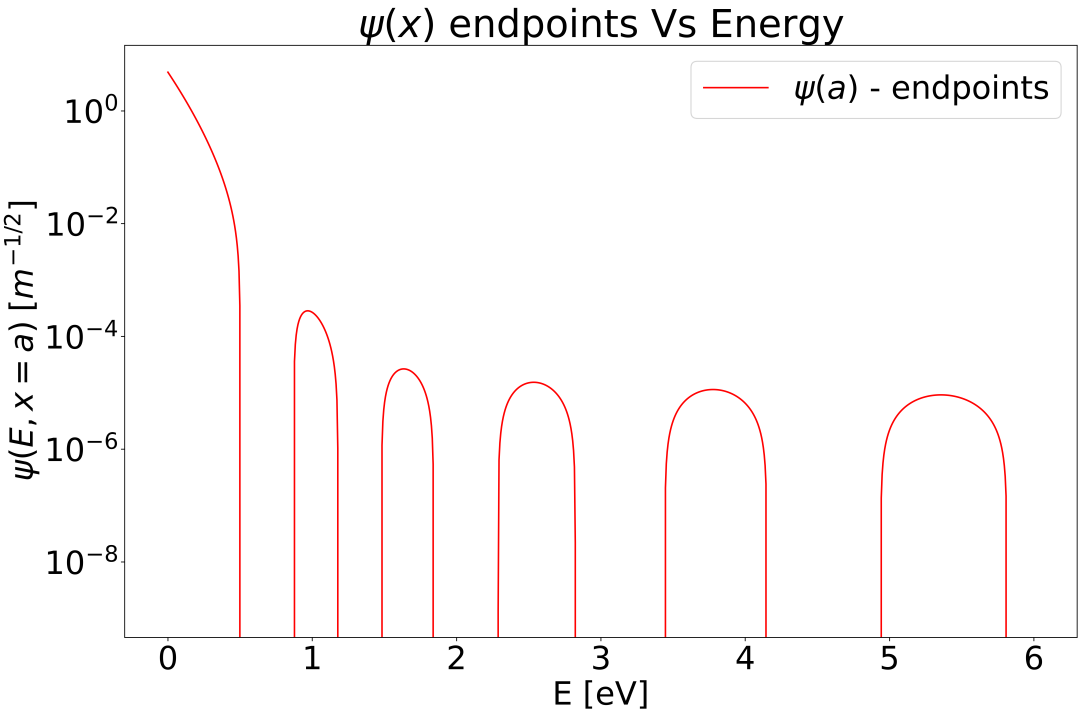

Figure 2. Log Scale Plot Showing Distribution of Energy Eigenvalues (Roots)

Table 2. Table of Energy Eigenvalues

Energy Level	**Energy [eV]**
E_1	0.500567553963311
E_2	0.8714876814569196
E_3	1.1781727259013322

```
        E_p, E_c, f_p, f_c = E_c, E_n, f_c, f_n
        iter += 1
        # print(f"Energy: {E_n}")
    return E_n
```

Therefrom, the following energies were found convergent to within 1×10^{-13} eV of the values from previous iterations:

TASK 2: EXPECTATION VALUES

$$\langle x \rangle_n = \int_0^a x \, |\psi_n(x)|^2 \, dx \tag{10}$$

$$\langle p \rangle_n = \int_0^a \psi_n^*(x) \left[-i\hbar \frac{\partial}{\partial x} \right] \psi_n(x) \, dx \tag{11}$$

To find the expectations, as per Eq.(10) and (11), the discrete wave function array must be normalised. Both these necessitate a numerical approximation of an integral,

so the following scheme is employed using the trapezoidal rule:
Given a continuous function f, the area under the curve of $f(x)$ can be approximated as a sum of trapezoids with area T, according to the trapezoidal rule:

$$T = \Delta x \left(\frac{f(x_i) + f(x_{i+1})}{2} \right) \tag{12}$$

Thus, the total area under $f(x)$ can be approximated as the sum of the area of connected trapezoids each of width Δx :

$$\begin{aligned} \sum_{i=0}^{N} T_i &= \Delta x \left(\frac{f(x_0) + f(x_1)}{2} + \frac{f(x_1) + f(x_2)}{2} + \cdots + \frac{f(x_{N-1}) + f(x_N)}{2} \right) \\ &= \Delta x \left(\frac{1}{2} f(x_0) + f(x_1) + \cdots + f(x_{N-1}) + \frac{1}{2} f(x_N) \right) \\ \therefore \int_{x_0}^{x_N} f(x)\, dx &\approx \Delta x \left(\sum_{i=1}^{N-1} f(x_i) + \frac{f(x_0) + f(x_N)}{2} \right) \end{aligned}$$

Thus, normalisation of the stationary states entails solving for an unknown scale factor A in front of them and setting the integral to 1:

$$A_n^2 \int_0^a \psi_n^2(x) = 1 \tag{13}$$

$$A_n^2 = 1 \div \Delta x \left(\sum_{i=1}^{N-1} \psi_n^2(x_i) + \frac{\psi_n^2(x_0) + \psi_n^2(x_N)}{2} \right)$$

and Eq.(10) and (11) become:

$$\langle x \rangle_n \approx A_n \Delta x \left(\sum_{i=1}^{N-1} \psi_n^2(x_i)\, x_i + \frac{\psi_n^2(x_0)\, x_0 + \psi_n^2(x_N)\, x_N}{2} \right) \tag{14}$$

$$\langle p \rangle_n \approx -A_n\, i\hbar\, \Delta x \left(\sum_{i=1}^{N-1} \psi_n(x_i) \psi_n'(x_i) + \frac{\psi_n(x_0)\psi_n'(x_0) \ + \psi_n(x_N)\psi_n'(x_N)}{2} \right) \tag{15}$$

Taking the wave functions and their normalisation constants A_n to be real simplifies matters and also implies that $\langle p \rangle_n = 0$ (see Appendix A) which is what is expected for stationary states.

TASK 3: PLOTTING WAVE FUNCTIONS

APPENDIX

A. MOMENTUM EXPECTATIONS

From Eq. 11 the momentum expectation is:

$$\langle p \rangle_n = -i\hbar \int_0^a \psi_n^* \frac{\partial \psi_n}{\partial x}\, dx$$

Integrating by parts to apply the derivative on ψ_n^* gives

$$= -i\hbar \cancelto{0}{\psi_n^* \psi_n \Big|_0^a} + i\hbar \int_0^a \frac{\partial \psi_n^*}{\partial x} \psi_n \, dx$$

Thus

$$\langle p \rangle_n = -i\hbar \int_0^a \psi_n^* \psi_n' \, dx = +i\hbar \int_0^a \psi_n'^* \psi_n \, dx$$

Since the stationary states ψ_n can be taken as real,

$$\begin{aligned} \langle p \rangle_n &= -i\hbar \int_0^a \psi_n \psi_n' \, dx = +i\hbar \int_0^a \psi_n \psi_n' \, dx \\ \implies \langle p \rangle_n &= 0 \end{aligned}$$

B. VISUAL ROOT FINDING

By populating an entire array with values produced by the `generatePsiEndpoint()` method, full wave functions can be produced for various energy values. Thus, finding valid wave functions and their corresponding energy eigenvalues entails plotting those wave functions and visually looking out for ones which behave as expected at both endpoints. Figure 3 presents 30 wave functions with energies between 0.5 eV and 1.18 eV: Granted, the visual inspection method is not as robust as checking the roots of the `generatePsiEndpoint()` method, but when the range of energies tightens around an eigenvalue, the plots become even more compelling, as shown in Figure 4

Software: Python (Van Rossum & Drake 2009), SymPy (Meurer et al. 2017)

REFERENCES

Meurer, A., Smith, C. P., Paprocki, M., et al. 2017, PeerJ Computer Science, 3, e103, doi: 10.7717/peerj-cs.103

Van Rossum, G., & Drake, F. L. 2009, Python 3 Reference Manual (Scotts Valley, CA: CreateSpace)

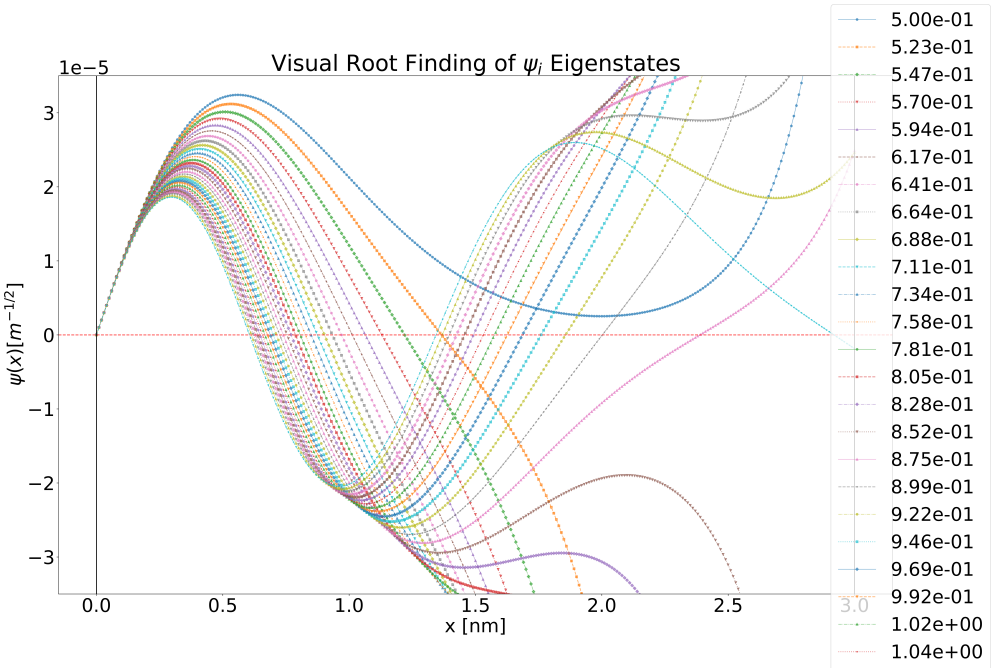

Figure 3. Plot of 30 Candidate Energy Eigenfunctions

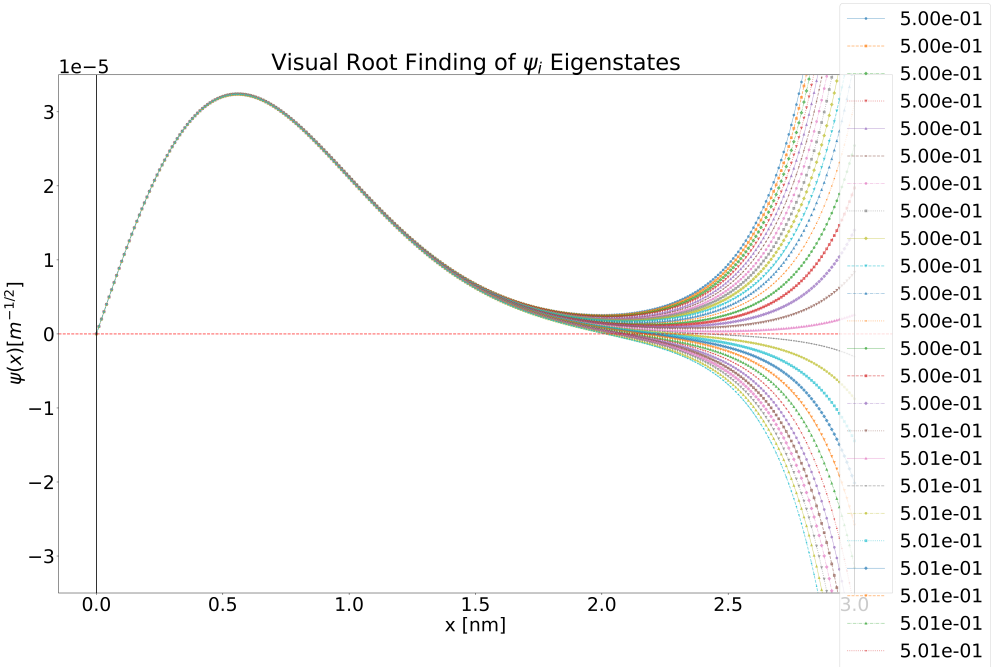

Figure 4. Plot of 30 Wave Functions Converging on an Energy Eigenstate