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, $E_1 = 0.5\,\text{eV}$, $E_2 = 0.87\,\text{eV}$, and $1.18\,\text{eV}$.

1. INTRODUCTION

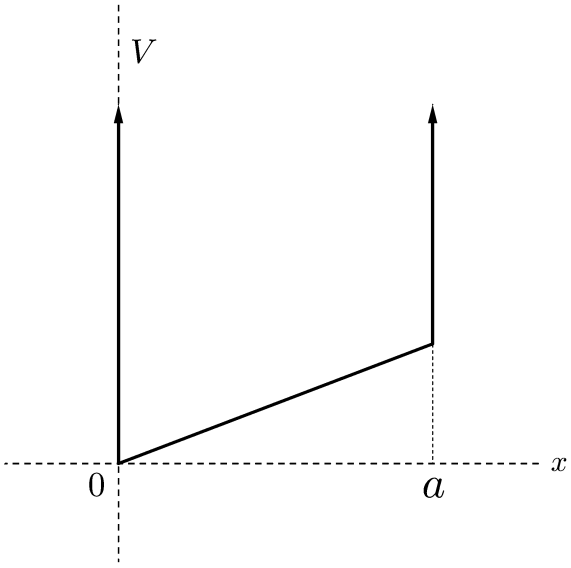

Figure 1. *Ramped Infinite Square Well Potential*

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

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

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

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

$$\hat{H}\psi_i = E_i\psi_i \tag{3}$$

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

$$\psi(x + \Delta x) = \psi(x) + (\Delta x)\psi'(x) + \frac{(\Delta x)^2}{2!}\boldsymbol{\psi''(x)} + \frac{(\Delta x)^3}{3!}\psi'''(x) + \cdots \tag{4}$$

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

By summing Eq. (4) and (5):

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

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

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

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

$$\boldsymbol{\psi''_i} = \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{(\Delta x)^2} \tag{6}$$

And the TISE becomes:

$$\boldsymbol{\psi_{i+1}} = \psi_i - \psi_{i-1} + g\left(E, x_i\right)(\Delta x)^2\psi_i \tag{7}$$

$$\text{where,} \quad g\left(E, x_i\right) = \frac{-2mc^2}{(\hbar c)^2}\left(E - V(x)\right) \tag{8}$$

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 (Griffiths & Schroeter 2018) to iterate ψ from $x = 0$ to $x = a$, keeping track of only three values:

Listing 1. Function Definition of ψ Endpoint Generator

```
    def g(E:float, x:float) -> float:
        """
        Helper function for 'generate_psi()'
        """
        return -((2 * mc2) / (hc)**2) * (E - b*x)


    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

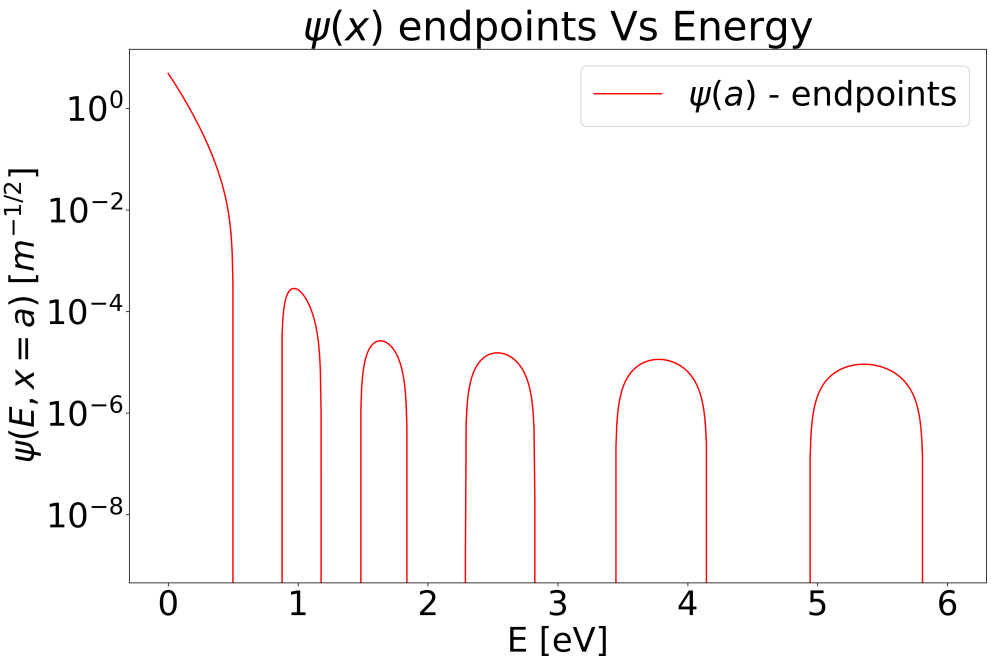

Figure 2. Log Scale Plot Showing Distribution of Energy Eigenvalues (Roots)

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_c, E_n = E_0, E_1, None
f_p, f_c = func(E_p), func(E_c)

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
    E_p, E_c = E_c, E_n
    f_p, f_c =  f_c, f_n
    iter += 1
return E_n
```

From `secant()`, the following energies, presented in Table 2 were found convergent to within 1×10^{-13} eV of the values from previous iterations:

Table 2. Table of Energy Eigenvalues

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

TASK 2: EXPECTATION VALUES

$$\langle x \rangle_n = \int_0^a x \, |\psi_n(x)|^2 \, dx \tag{9}$$

$$\langle p \rangle_n = \int_0^a \psi_n^*(x) \left[-i\hbar \frac{\partial}{\partial x} \right] \psi_n(x) \, dx \tag{10}$$

To find the expectations, as per Eq.(9) and (10), the discrete wave function array must be normalised. This necessitates a numerical approximation of an integral, so the following scheme is employed:
Given a continuous function f, the area under the curve of $f(x)$ can be approximated as a sum of horizontally stacked trapezoids each with area T, according to the trapezoidal rule:

$$T = \Delta x \left(\frac{f(x_i) + f(x_{i+1})}{2} \right) \tag{11}$$

Thus, the total area under $f(x)$ can be approximated as the sum of the areas of the connected trapezoids each with 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 can be carried out by finding the reciprocal of the area under $|\psi_n(x)|^2$ and defining a normalisation constant A^2 which will scale ψ_n. The implementation is shown in Listing 5.

$$A_n^2 \int_0^a \psi_n^2(x) = 1 \tag{12}$$

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

So, using the normalised wave functions, Eq.(9) and (10) become:

$$\langle x \rangle_n \approx \Delta x \left(\sum_{i=1}^{N-1} \psi_n^2(x_i) \, x_i + \frac{\psi_n^2(x_0) \, x_0 + \psi_n^2(x_N) \, x_N}{2} \right) \tag{13}$$

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

The wave functions being real simplifies matters and also implies that $\langle p \rangle_n = 0$ (see Appendix A) which *is* expected for stationary states. Furthermore, the first derivative of the wave function can be found using a second order approximation in a similar fashion to Eq. 6. This, coupled with the simplification that the wave function should be zero at the endpoints, yields the following expressions for the first derivative and the momentum expectation:

$$\boldsymbol{\psi_n'(x_i)} \approx \frac{\psi_n(x_{i+1}) - \psi_n(x_{i-1}) - \cancelto{0}{\mathcal{O}(\Delta x)^3}}{2\Delta x}$$
$$\implies \langle p \rangle_n \approx i\hbar\, \Delta x \left[\sum_{i=1}^{N-1} \psi_n(x_i) \left(\frac{\psi_n(x_{i+1}) - \psi_n(x_{i-1})}{2\Delta x} \right) \right]$$

Or, more conveniently:

$$\langle p \rangle_n c = \frac{-2i\ \hbar c}{2} \left(\sum_{i=1}^{N-1} \psi_n(x_i) \left(\psi_n(x_{i+1}) - \psi_n(x_{i-1}) \right) \right)$$

These expressions were implemented in Listings 3 and 4 and their results are shown in Table 3. The momentum expectations are quoted as the square modulus of $\langle p \rangle_n c$ in units of eV for convenience.

Listing 3. Position Expectation Implementation

```
    def positionExpectation(psi:np.ndarray):
    """Calculates the position expectation
    of a normalised energy eigenstate
    """
    psiSquared = psi**2
    endpoints = (psiSquared[0] * x[0] + psiSquared[-1] * x[-1]) / 2
    return dx * (sum(psiSquared[1:-1] * x[1:-1]) + endpoints)
```

Listing 4. Momentum Expectation Implementation

```
    def momentumExpectation(psi:np.ndarray):
    """Calculates the modulus of the momentum expectation
    of a normalised energy eigenstate
    """
    result = -0.5 * (1j * hc) * sum(psi[1:-1] * (psi[:-2] - psi[2:]))
    return np.abs(result)
```

TASK 3: PLOTTING WAVE FUNCTIONS

Normalisation of each wave function was performed according to the procedure in Listing 5, as detailed in Section 1

Table 3. Table of Position and Momentum Expectation Values

Quantity	Value	Units
$\langle x \rangle_1$	0.66	[nm]
$\langle x \rangle_2$	1.15	[nm]
$\langle x \rangle_3$	1.53	[nm]
$\lvert \langle p \rangle_1 c \rvert$	5.10×10^{-14}	[eV]
$\lvert \langle p \rangle_2 c \rvert$	1.09×10^{-14}	[eV]
$\lvert \langle p \rangle_3 c \rvert$	8.46×10^{-14}	[eV]

Note: Clearly, the momentum expectations, even before converting to their appropriate units, are vanishingly small.

Listing 5. Wave Function Normalisation Implementation

```
    def normalisePsi(psi:np.ndarray) -> np.ndarray:
    """Normalise |psi|^2 to find normalisation factors `A^2` and `A`
    for the square modulus of psi and psi, respectively
    """
    psiSquared = psi**2
    endpoints = (psiSquared[0] + psiSquared[-1]) / 2
    integralPsiSquared = dx * (np.sum(psiSquared[1: -1] + endpoints))
    A_squared = 1 / integralPsiSquared

    return np.sqrt(A_squared) * psi
```

Finally, Figure 3 presents the first three energy eigenstates. They have been scaled down to better appreciate the distribution of the energy levels in the ramped well. This, naturally, comes at the cost of normalisation. Thus, the normalised probability densities $|\psi_n(x)|^2$ and their corresponding position expectations are also presented in Figure 4.

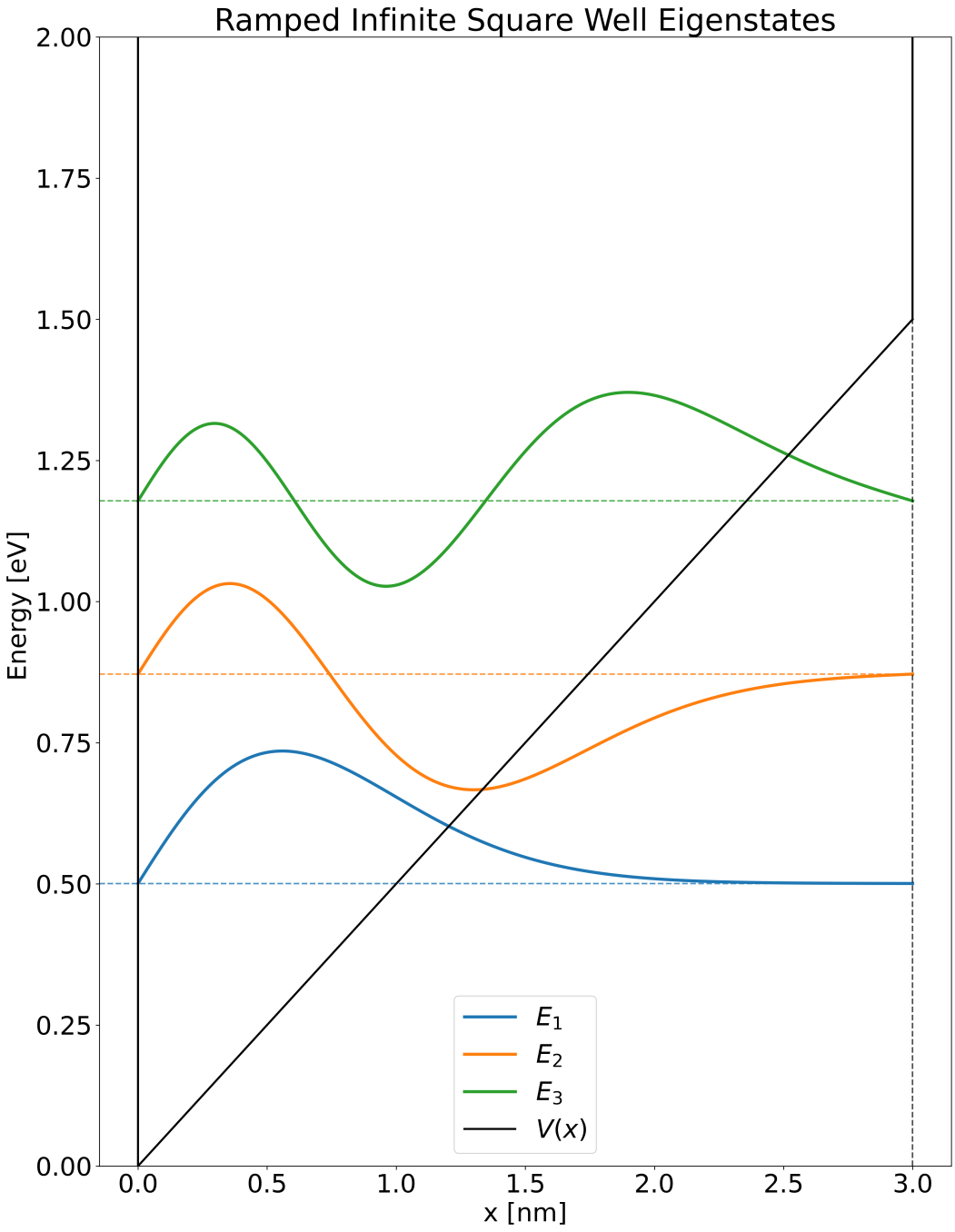

Figure 3. First Three Energy Eigenstates of Ramped ∞ Square Well

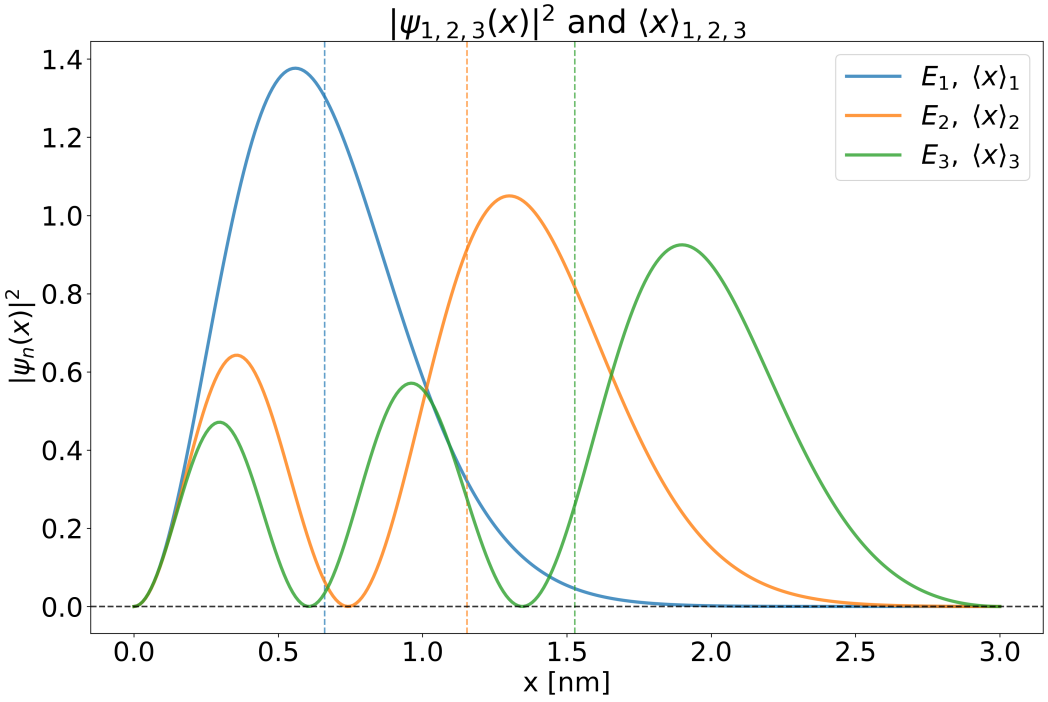

Figure 4. Wave Function Probability Density Plots with Position Expectations

APPENDIX

A. MOMENTUM EXPECTATIONS

From Eq. 10 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,

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

$$\implies \langle p\rangle_n = 0$$

This result is supported by the results quoted in Table 3.

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 looking out for ones which behave as expected at both endpoints. Figure 5 presents 30 wave functions with energies between 0.5 eV and 1.18 eV. Granted, the visual inspection method is not as

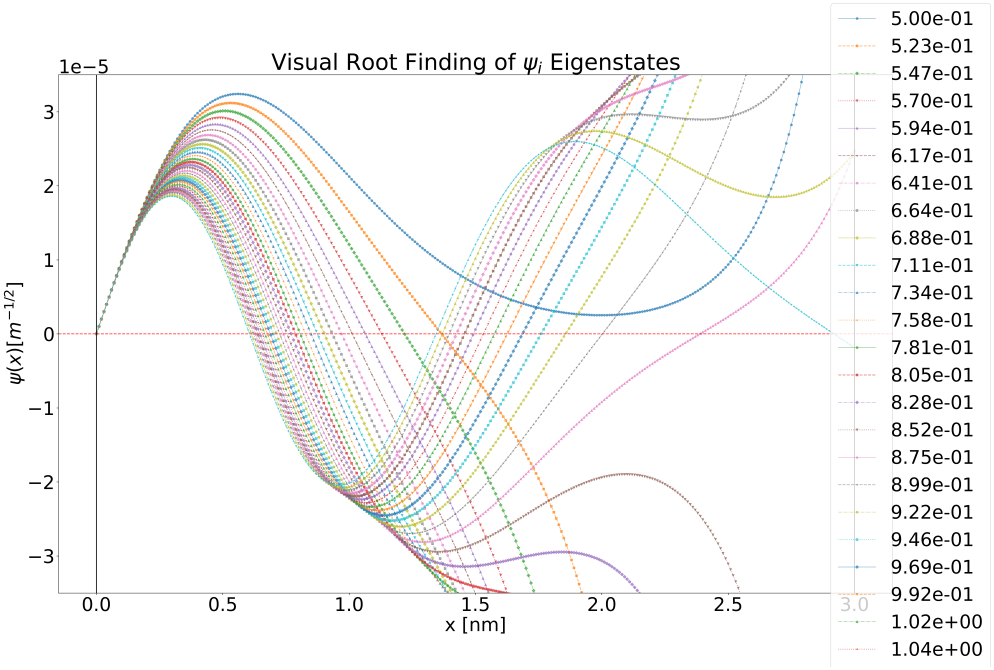

Figure 5. Plot of 30 Candidate Energy Eigenfunctions

robust as searching for roots of the `generatePsiEndpoint()` method, but when the range of possible energies tightens around an energy eigenvalue, the plots become even more visually compelling, as shown in Figure 6.

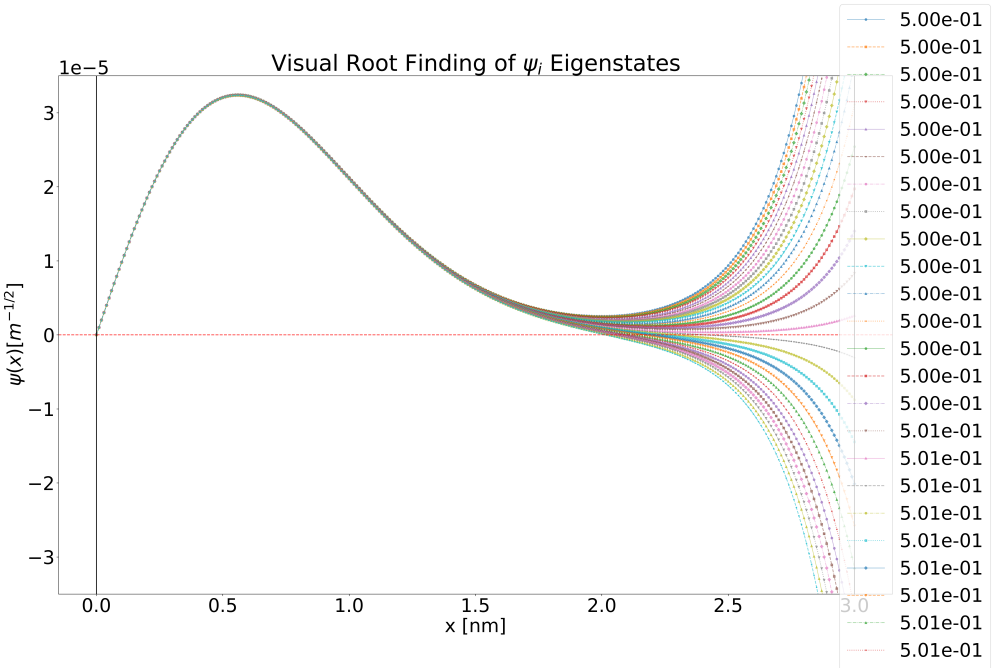

Figure 6. Plot of 30 Wave Functions Converging around an Energy Eigenstate

Software: Python (Van Rossum & Drake 2009), Matplotlib (Hunter 2007)

REFERENCES

Griffiths, D., & Schroeter, D. 2018, Introduction to Quantum Mechanics (Cambridge University Press), 93–94. https://books.google.co.za/books?id=82FjDwAAQBAJ

Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

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