CO22: Solutions of Schrödinger's Equation by Numerical Integration

Calum Holker
Somerville College

1 Abstract

A numerical solution to the time-independent Schrödinger equation in one dimension is found using Numerov's method. The use case of simple harmonic motion is used to test the results of the numerical solutions against the easy to find analytical solutions. The results showed that the solution this method produces has no significant variation from the analytical solution for small values of $\hat{x}$. It is theorised that the reason for huge inaccuracies at larger values of $\hat{x}$ is due to the use of a Taylor expansion to find the second value of ψ, expanded around $\hat{x} = 0$. It is also shown that the further the energy eigenvalue from the exact value the larger the error, and hence a method of find the exact value from a close guess is formulated.

2 Introduction

The Schrödinger equation is the most important equation in quantum mechanics, describing the wavefunction for a potential. This means that it is incredibly useful to have solutions for this, as the wavefunction can be used to solve many problems in physics. In one dimension the time-independent Schrödinger equation we are looking for solutions of can be written as:

$$\frac{d^2\psi}{dx^2} + \frac{2m}{\hbar^2}[E - V(x)]\psi = 0 \tag{1}$$

Solutions to this equation cannot be found analytically for many $V(x)$, however (if solutions exist), they can be found numerically for all potentials. In this case we will look at a method of numerically solving this equation for a potential of a simple harmonic oscillator $V(x) = \frac{1}{2}kx^2$. Here k is the spring constant of the oscillator. This potential can easily be solved analytically, hence we can use it to test the accuracy of the numerical method by comparing it's approximate solutions to the exact ones. If our numerical method is accurate, it can then be applied to potentials that cannot be solved analytically, hence proving to be a very useful tool.

To simplify the problem, it is useful if we define new non-dimensional variables [1]:

$$\hat{x} = (\frac{mk}{\hbar^2})^{\frac{1}{4}}x \quad \textit{and} \quad \hat{E} = \frac{2}{\hbar}\sqrt{\frac{m}{k}}E \tag{2}$$

This mean that we now have the equation:

$$\frac{d^2\psi}{dx^2} = [\hat{V}(\hat{x}) - \hat{E}]\psi \tag{3}$$

If we solve this analytically we obtain solutions in the form below, where $H_n(\hat{x})$ are the Hermite polynomials. The energy eigenvalues for this potential are also easily found [1]:

$$\psi = H_n(\hat{x})e^{-}\frac{\hat{x}^2}{2} \tag{4}$$

$$\hat{E_n} = 2n + 1 \tag{5}$$

To solve this numerically we use Numerov's Method, which is detailed in the following section. We can then solve equation 3 for a range of eigenvalues between the limits of $\hat{x} = 0$ and an upper bound $\hat{x_1}$. Comparing the plots of the exact analytical solution and the numerical solutions obtained will allow us to examine the accuracy of our method, and how precisely we need to know the energy eigenvalue to obtain an accurate numerical solution.

3 Method

The Numerov method can be applied to find solutions of linear ordinary differential equations that contain no first derivative term [2]. For an equation in the form below, we obtain the solution below [3] for solutions with a spacing of δ. f_j and ψ_j are the values of f and ψ at $\hat{x} = j\delta$.

$$\frac{d^2\psi}{dx^2} = f(\hat{x})\psi \tag{6}$$

$$(1 - \frac{\delta^2}{12}f_{j+1})\psi_{j+1} = (2 - \frac{5}{6}\delta^2 f_j)\psi_j - (1 + \frac{\delta^2}{12}f_{j-1})\psi_{j-1} + O(\delta^6) \tag{7}$$

This equation can be used to find solutions given two initial values ψ_{j-1} and ψ_j. For solving the Schrödinger equation we can define boundary conditions at $\hat{x} = 0$. Our form of $f(\hat{x}$ is $\hat{V}(\hat{x}) - \hat{E}$. $\hat{V}(\hat{x})$ is symmetric so solutions will fall into even and odd categories [1]. We then have the following boundary conditions to apply at $\hat{x} = 0$.

$$\begin{aligned} even: \quad & \psi(0) = 1 \quad & \frac{d\psi}{d\hat{x}} = 0 \\ odd: \quad & \psi(0) = 0 \quad & \frac{d\psi}{d\hat{x}} = 1 \end{aligned} \tag{8}$$

The boundary conditions only provide one value, so the second value needed to start the calculation is found using a Taylor expansion around $\hat{x} = 0$ [1]. Prime here means differentiation with respect to $\hat{x}$:

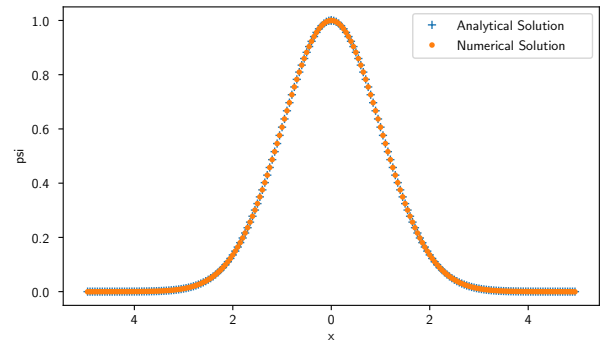
$$\psi(\delta) = \psi(0) + \psi'(0)\delta + \frac{1}{2}\psi''(0)\delta^2 + \frac{1}{6}\psi'''(0)\delta^3 + \frac{1}{24}\psi''''(0)\delta^4 + ... \tag{9}$$

We can then apply the boundary conditions and the fact that original equation implies $\psi'' = f\psi$ and the product rule to get two separate equations for the even and odd cases:

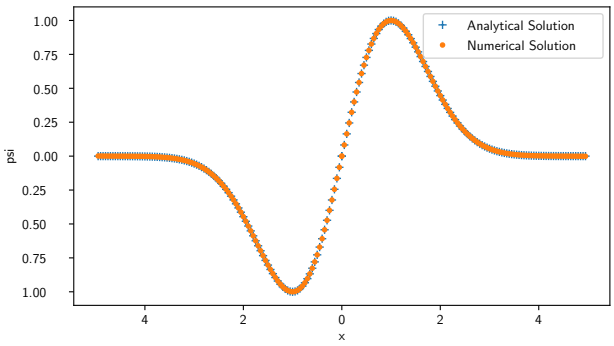
$$\begin{aligned} even: \quad \psi(\delta) &= \psi(0) + \frac{\delta^2}{2}f(0)\psi(0) + \frac{\delta^4}{24}\Big(f''(0)\psi(0) + 2f'(0)\psi'(0) + f(0)^2\psi(0)\Big) + ... \\ odd: \quad \psi(\delta) &= \delta\psi'(0) + \frac{\delta^3}{6}\big(f(0)\psi'(0) + f'(0)\psi(0)\big) + ... \end{aligned} \tag{10}$$

We note that for our equation $f'(0) = 0$ and the parity of the case depends directly on the parity of n. Finally as the equation is homogeneous our solution may be scaled by an arbitrary factor, so both the analytical and numerical solutions are scaled between 1 and -1 [1].

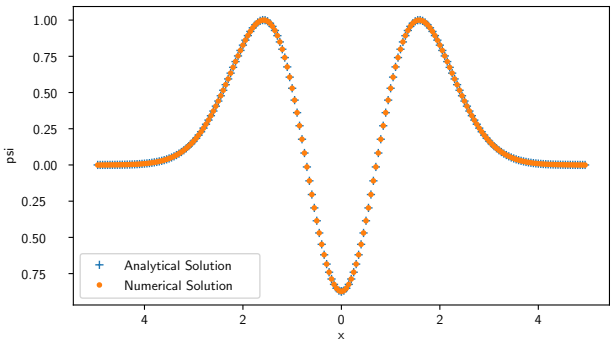
4 Results



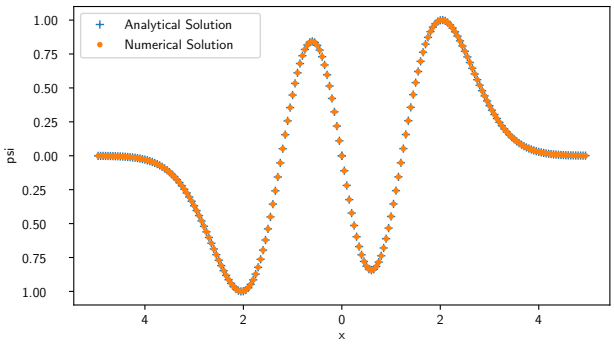
(a) $n = 0$



(b) $n = 1$

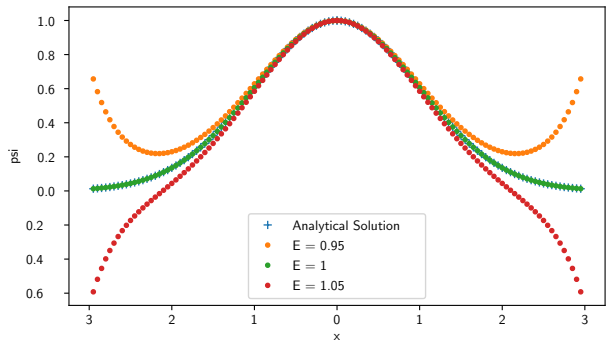


(c) $n = 2$

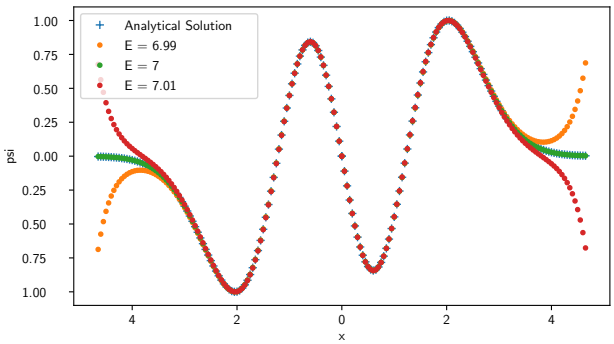


(d) $n = 3$

Figure 1: Numerical vs analytical solutions to Schrödinger's equation for energy level n



(a) $n = 0$



(b) $n = 3$

Figure 2: Dependence on solution on the energy eigenvalue $\hat{E}$ given

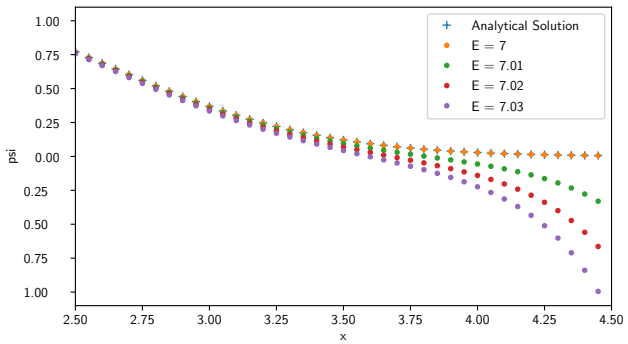
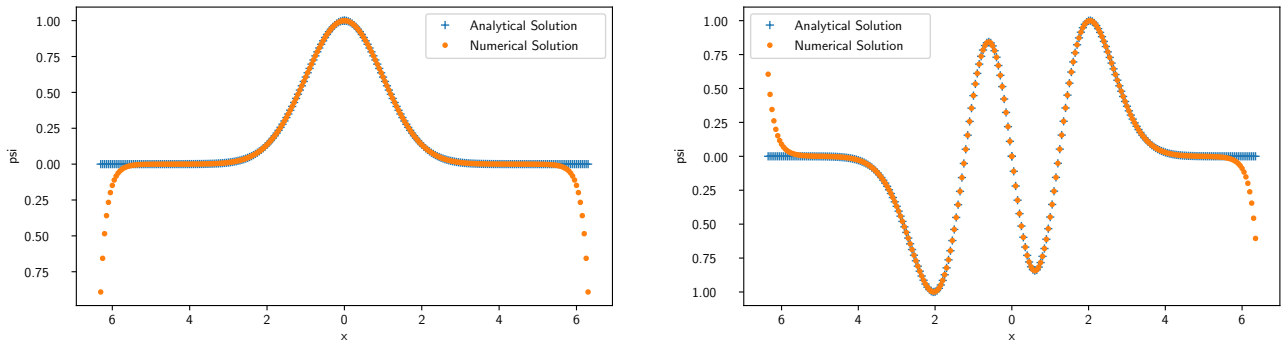


Figure 3: Increase in error as distance from exact energy eigenvalue increases



(a) $n = 0$ (b) $n = 3$

Figure 4: Difference in numerical solution for large $\hat{x}$

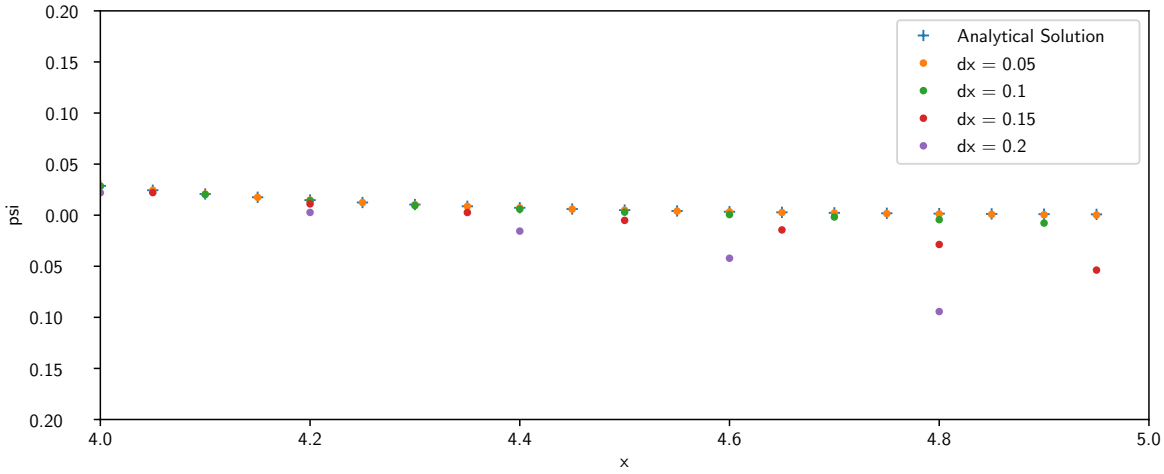


Figure 5: Increase in global error as the spacing increases

We first used Numerov's method to plot the graphs in figure 1 which are the numerical solutions for the lowest 4 energy levels, compared to the exact analytical solution for small $|\hat{x}| < 5$. In this range the numerical solution did not significantly differ from the analytical solution for all 4 energy levels. Then, looking at how the energy eigenvalue used affects the result in figure 2 and figure 3, it is clear to see that as the energy eigenvalue differs more from the exact value, the bigger the deviation from the exact solution will be. Note that for small $\hat{x}$ the solution remains accurate, however figure 3 also suggests that if the difference from the exact value is greater significant deviation from the exact solution begins at a smaller value of $\hat{x}$ and deviates at a faster rate.

Figure 4 looks at slightly larger values of $\hat{x}$ and it is clear that for larger values the numerical solution quickly becomes massive and the solution is unusable.

Finally figure 5 looks at how the spacing δ affects the magnitude of the error. It suggests that the larger the value of the spacing, the greater the deviation from the analytical solution.

5 Analysis

Figure 1 suggests that our solution using Numerov's method is accurate for small values of $\hat{x}$. There is no significant difference between the analytical and numerical solutions. It can therefore be deduced that the theory is correct and the Numerov method can be used for more complex problems that cannot be solved analytically.

Figure 2 and figure 3 show the importance of having an energy eigenvalue as close to the exact value as possible. In the case of the simple harmonic oscillator this is not a problem as we have derived the energy eigenvalues analytically. However in the case of more complex problems this may be an issue if they cannot be found exactly. In many problems a Hamiltonian can be described as a sum of two components, one that can be solved exactly and a perturbation. In these cases we may be able to find an energy eigenvalue that is close to the exact value, however not exact enough for the Numerov method to work. Due to this a method was devised which used gradient descent to find the exact energy eigenvalue from a close guess. This worked by taking the gradient at a large value of $\hat{x}$ to 0. The results of this method in the case of the simple harmonic oscillator are below:

n	**Exact Value**	**Initial Guess**	**Estimated Value**
0	1	0.8	1.0000000000000002
1	3	2.8	2.999999999999989
2	5	5.3	5.000000000000034
3	7	6.7	7.000000000000034
4	9	8.5	8.999999999999723

It is clear to see that this method of gradient descent works, and is useful if it is not possible to calculate an exact energy eigenvalue.

At large values of $\hat{x}$ it is clear that the analytical solution becomes unusable. This does not arise in the Numerov method. Rather this is due to the fact that our second value of ψ was found using a Taylor expansion taken only to 4th order. The error in the expansion increases the further you get from the value the expansion was calculated around [6]. As we expanded around $\hat{x} = 0$, the further from that point the greater the error. If the values of ψ at larger values of $\hat{x}$ are required, the Taylor expansion would need to include higher order terms past 4th order.

Now we must look into the errors of Numerov's method [3]. Firstly the trunctation error, the approximation of excluding terms of $O(\delta^6)$ mean that Numerov's method has a local truncation error of order δ^6. As the integration is between two definite values, the global error appears to be order δ^5, however it can be shown that the error accumulates in successive steps and the resulting global error at a fixed value of $\hat{x}$ is of order δ^4.

There are many advantages of the Numerov method. It has high ease in programming and low time complexity. One of the most important advantages is that it is very efficient (close to piecewise pertubation methods) for problems involving high values of energy, and as of such is highly relevant to solving the Schrödinger equation [7]. It also is particularly suited for the one dimensional Schrödinger equation as it takes advantage of the fact that there is no first order derivative term and linear in ψ [3].

There are a few alternatives to the Numerov method that can also be used. One method is the 4th order Runge-Kutta method (RK4) which has the same global error of $O(\delta^4)$ [4]. However it's time complexity is much greater as it would require writing Schrödinger's equation as two separate first order equations. Another method is to use the matrix Numerov algorithm [5]. This involves vectorisation of the Numerov method specifically for this problem. This method has the same errors as the classical Numerov method however has much lower time complexity. This is particularly true if the computer being used has a GPU which can be utilised to perform the large matrix calculations. This method is much more elegant and is a potential improvement on the Numerov method we have used.

6 Conclusion

A numerical solution to the time-independent Schrödinger equation in one dimension is found using Numerov's method. The use case of simple harmonic motion is used to test the results of the numerical solutions against the easy to find analytical solutions. The results showed that the solution this method produces has no significant variation from the analytical solution for small values of $\hat{x}$. It is theorised that the reason for huge inaccuracies at larger values of $\hat{x}$ is due to the use of a Taylor expansion to find the second value of ψ, expanded around $\hat{x} = 0$. It is also shown that the further the energy eigenvalue from the exact value the larger the error, and hence a method of find the exact value from a close guess is formulated. The method could be improved by implementing the matrix version of the Numerov method, which would reduce time complexity.

References

[1] EG, MK, NC *CO22 Solutions of Schrödinger's equation by numerical integration*, Oxford Physics, 2020[1].

[2] Drexel *The Numerov Method*, [2].

[3] Peter Young *Numerov method for integrating the one-dimensional Schrödinger equation*, 2009[3].

[4] I.H. Sloan *Errors in the Numerov and Runge-Kutta methods*, 1968[4].

[5] Danny Bennett *Numerical Solutions to the Time-Independent 1-D Schrodinger Equation*, 2015[5].

[6] Dartmouth *Error estimates in Taylor approximations*, [6].

[7] L.Gr Ixaru, M Rizea *Numerov method maximally adapted to the schrödinger equation*, [7].

[1] https://www-teaching.physics.ox.ac.uk/practical_course/scripts/srv/local/rscripts/trunk/Computing/CO22/CO22.pdf

[2] http://www.physics.drexel.edu/~steve/Courses/Comp_Phys/Physics-305/numerov.pdf

[3] http://physics.ucsc.edu/~peter/242/numerov.pdf

[4] https://doi.org/10.1016/0021-9991(68)90047-8

[5] https://www.maths.tcd.ie/~dbennett/js/schro.pdf

[6] https://math.dartmouth.edu/~m8s17/ErrorEstimates.pdf

[7] https://doi.org/10.1016/0021-9991(87)90139-2

Appendix Functions Source Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import derivative as der
from scipy.special import eval_hermite as herm

def solve_numerov(f, x, psi0, dpsi0, parity='even'):
    """ Solving d^2(psi)/dx^2 = f(x) psi from x0 to x1 with boundary condition psi(x0)=psi0 and d(psi(x0))/dx = dpsi0
    Input:  f - the function to be called on the rhs of the equation, receives x and returns the value of f(x)
            x - array of the integration, (upper bound x0, lower bound x1)
            psi0 - the value of \psi at x0
            dpsi0 - the value of \psi derivative at x0
            parity - determines even or odd solutions
    Output: psi - array of values of \psi with each element in the array corresponding to the same element in x
    """
    dx = x[1]-x[0]

    if parity == 'even':
        psidx = psi0 + 0.5*(dx)**2*(f(0)*psi0)+(1/24)*(dx**4)*(der(f, 0, dx=1e-6, n=2)*psi0+(2*der(f, 0, dx=1e-6, n=1)*dpsi0)+f(0)**2*psi0)
    elif parity == 'odd':
        psidx = dx*dpsi0 + (dx**3)*(1/6)*(f(0)*dpsi0+der(f, 0, dx=1e-6, n=1)*psi0)

    psi = [psi0, psidx]

    for j in range(1, len(x)-1):
        A1 = 2+((5/6)*dx**2*f(j*dx))
        A2 = (1-((dx**2)/12)*f((j-1)*dx))
        A3 = (1-((dx**2)/12)*f((j+1)*dx))
        psi.append((A1*psi[j]-A2*psi[j-1])/A3)
    return psi

def solve_analytically(n, x, norm = True):
    """ Finds the analytical solution
    Input:  n - energy level
            dx - spacing
            x1 - upper bound for x
            norm - defines if psi is normalised or not, by default is True
    Output: psi - analytical solution
    """
    hermite = herm(n, x) # Obtain the relevant hermite polynomial
    exp = np.exp(-0.5*np.multiply(x, x))
    psi = hermite*exp
    if norm:
        return np.divide(psi, max(np.abs(psi))) # Normalise between 1 and -1
    else:
        return psi

def solve_numerically(n, x, E, norm = True):
    """ Finds the analytical solution
    Input:  n - energy level
            dx - spacing
            x1 - upper bound for x
            E - energy of oscillator
            norm - defines if psi is normalised or not, by default is True
    Output: psi - numerical solution
    """
    f = lambda y: y**2-E # Defines simple harmonic oscillator function
    if n%2 == 0:
        psi = solve_numerov(f, x, 1, 0)
    else:
        psi = solve_numerov(f, x, 0, 1, parity='odd')
    if norm:
        return np.divide(psi, max(np.abs(psi))) # Normalise between 1 and -1
    else:
        return psi

def find_oscillator_eigenvalue(n, x, E0=1, norm = True, step=0.001):
    """ Finding the eigenvalue for harmonic oscillator potential by iteration with starting value E0
    Input:  E0 - the initial guess of the eigenvalue
    Output: E - the eigenvalue near the initial guess of the system with harmonic oscillator potential
    Constraints: E0 is a positive real number
    """
    E = E0
    psi = solve_numerically(n, x, E, norm)
    dx = x[1]-x[0]
    slopeinit = (psi[-1]-psi[-2])/dx # Calculate the slope at large values of x (should be 0)
    if slopeinit > 0:
        slope = slopeinit
        while slope > 0:
            slope_before = slope
            E = E-step # If the slope is positive decrease the value of E
            psi = solve_numerically(n, x, E, norm)
            slope = (psi[-1]-psi[-2])/dx # Recalculate slope after step
            if abs(slope) > abs(slope_before): # If the slope becomes larger, have gone too far
                step = step*-1
    else:
        slope = slopeinit
        while slope < 0:
            slope_before = slope
            E = E+step # If the slope is positive increase the value of E
            psi = solve_numerically(n, x, E, norm)
            slope = (psi[-1]-psi[-2])/dx # Recalculate slope after step
            if abs(slope) > abs(slope_before): # If the slope becomes larger, have gone too far
                step = step * -1
    return E

def plot_num(n, x1, dx, E, scale = 1, marker = '.', label = 'Numerical_Solution'):
    """
    Plots numerical solutions
    """
    x_pos = np.arange(0, x1, dx) # positive values of x
    psi_pos = solve_numerically(n, x_pos, E)
```

```
    x_neg = x_pos * -1 # negative values of x
    if n % 2 == 0:
        psi_neg = solve_numerically(n, x_pos, E) # even solutions mean solution is symmetric
    else:
        psi_neg = solve_numerically(n, x_pos, E) * -1 # odd solutions mean solution is antisymmetric
    x = np.concatenate((x_pos, x_neg), axis=0) # combine even and odd solutions
    psi = np.concatenate((psi_pos, psi_neg), axis=0)
    plt.plot(x, psi*scale, marker, label=label)
    plt.legend()

def plot_an(n, x1, dx, scale = 1, marker = '+', label = 'Analytical_Solution'):
    """
    Plots analytical solutions
    """
    x_pos = np.arange(0, x1, dx)
    x_neg = x_pos * -1
    x = np.concatenate((x_pos, x_neg), axis=0)
    psi = solve_analytically(n, x)
    plt.plot(x, psi*scale, marker, label=label)
    plt.legend()
```