Solving the Schrodinger's Equation using Finite Difference Methods

Magbutay, John Anfernee G.

PHYS 3178 Technical Report
Department of Physics, University of San Carlos,
Nasipit, Talamban, 6300 Cebu City, Philippines
magbutay.anfernee@gmail.com

Abstract

In this article, a purely numerical solution of the Time-Independent Schrodinger Equation was found using a finite difference methods. In this method the function $\Psi(x)$ was turned into a vector, and the operator of the differential equation into a matrix. This returned a matrix eigenequation, which was then diagonalized to obtain the solution. Illustrative cases include: energy levels of an infinite square well and the bound states of a square well in one-dimension. The results show that Finite Difference Methods successfully solved the Time-Independent Schrodinger Equation, returning the energy levels after five iterations and with the values averaged, with ± 0.0977 accuracy.

1 Introduction

The Time-Independent Schrodinger Equation is a linear ordinary differential equation that described the wavefunction or state function of a quantum-mechanical system. The solution of the Schrodinger Equation yields quantized energy levels as well as wavefunctions of a given quantum system. The Schrodinger Equation is interesting for computational physics because it can be used to model the behavior of elementary particles, and atoms without the need of computing manually. And when combined with the superposition principle, the Schrodinger equation accounts for the formation of chemical bonds, and hence can be used to model molecular systems and periodic systems such as crystalline materials.
In this report, finite difference methods is utilized for numerical solutions to the one-dimensional infinite square well problem [?]. The different finite difference methods are used to approximate the second derivative in the one-dimensional Schrodinger's Equation and linearize the problem. By doing so, the Infinite Square Well problem is converted to an Eigenvalue problem, which can then provide the Energy levels of the quantum system in question.

2 Theoretical Framework

2.1 Forward and Backward First-Order Differential

It is first necessary to develop how to take the derivative of our function. Going back to the introduction of calculus, it is noted that the derivative is defined as:

$$\tfrac{d}{dx}f(x) = \lim_{\Delta x \to 0} \tfrac{f(x+\Delta x)-f(x)}{\Delta x} \approx \tfrac{f(x+h)-f(x)}{h} + \mathrm{O}(h) \tag{1}$$

The most direct method for numerical differentiation [4] starts by expanding a function in a Taylor series to obtain its value at a small step h away:

$$y(t+h) = y(t) + h\frac{dy(t)}{dt} + \frac{h^2}{2!}\frac{d^2y(t)}{dt^2} + \frac{h^3}{3!}\frac{d^3y(t)}{dt^3} + ..., \tag{2}$$

$$\to \frac{y(t+h)-y(t)}{h} = \frac{dy(t)}{dt} + \frac{h}{2!}\frac{d^2y(t)}{dt^2} + \frac{h^2}{3!}\frac{dy^3(t)}{dt^3} + ... \tag{3}$$

Ignoring the h^2 terms in eq. (3), the forward-difference derivative algorithm is obtained for the derivative eq.(2) for $y'(t)$. When the finite difference is taken, the limits are not taken all the way down to 0, but stop at $\Delta x = h$. Note that for this equation, it is evaluated at the point just after x. If $\Delta x \to 0$, then this does not matter, but if you do a finite difference, you can also do:

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} \approx \frac{f(x) - f(x-h)}{h} + \mathrm{O}(h) \tag{4}$$

which is known as the backward difference. The central difference can also be computed with much higher accuracy, but one cannot use steps of $\frac{1}{2}\Delta x$, since that does not exist in the computational space of this report. The central difference is then a combination of the previous two:

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x - \Delta x)}{2\Delta x} \approx \frac{f(x+h) - f(x-h)}{2h} + \mathrm{O}(h^2) \tag{5}$$

2.2 Forward and Backward Second-Order Differential

Extending from Chapter 2.1 for the second-order differential, if the backward differential of the result of the forward differential is taken:

$$\frac{d^2}{dx^2}f(x) = \lim_{\Delta x \to 0} \frac{f'(x) - f'(x - \Delta x)}{\Delta x} \tag{6}$$

$$= \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x) - (f(x) - f(x-\Delta x))}{\Delta x^2} \tag{7}$$

$$\frac{d^2}{dx^2}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x))}{\Delta x^2} \approx \frac{f(x+h) - 2f(x) + f(x-h))}{h^2} \tag{8}$$

An implementation of the FDM, that will be used in the report, on the function $\sin(x)$ is shown in Figure 1.

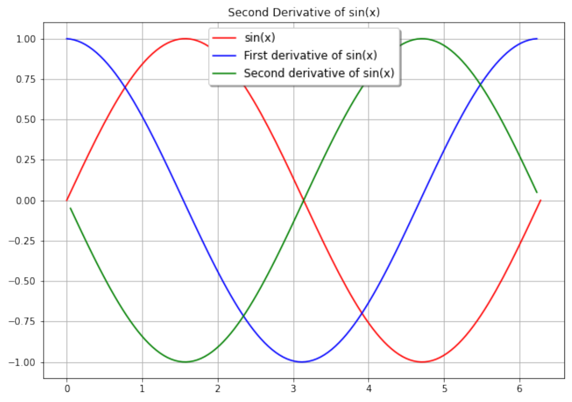


Figure 1: Implementation of Second-Order FDM on the function sin(x).

2.3 Time Independent Schrodinger Equation

As discussed in the introduction, the Time-Independent Schrodinger Equation is a linear partial differential equation that describes the wave function of a quantum-mechanical system [2]. The Schrodinger equation's Hamiltonian operator is essential in this report. In classical mechanics, the Hamiltonian is a scalar-valued function, whereas in quantum mechanics, it is an operator on a space of functions. The eigenvalues of $\hat{H}$ are the energy levels of the system. The Time-Independent Schrodinger equation for a one-dimensional quantum system is given as:

$$-\frac{\hbar^2}{2m}\frac{d^2\Psi}{dx^2} + V(x)\Psi(x) = E\Psi(x) \tag{9}$$

In the case of this report, the eigenvalue equation form of eq. (9) is desired. Therefore:

$$H\Psi(x) = E\Psi(x) \tag{10}$$

where, $H = -\frac{\hbar^2}{2m}\Delta + V(x)$ is the Hamiltonian

3 Methodology

3.1 Discretization of the continuous space

The process of discretization is simply turning the continuous space x, into a discrete number of steps, N, and the function $\psi(x)$ into an array of size N [1]. Thus, the N values x_i, which have a stepsize $h = \Delta x = x_{i+1} - x_i$. The choice of the size of the space, N, turns out to be important. Too large a number will slow down the computation and require too much computer memory, too small a number and the answers computed will not be sufficiently accurate. In this report, a small number N was introduced and then increased it until the accuracy is acceptable.

3.2 Matrix Representation

In order to find the energy levels of the system, it is first necessary to represent the equations in matrix form in order to evaluate the Hamiltonian. First, the vector $f(x) = [f_0, f_1, f_2, ..., f_{N-1}]$ is introduced as well as its derivative, $f'(x) = \left[f'_0, f'_1, f'_2, ..., f'_{N-1}\right]$. The forward difference [3] can then be written as:

$$f'_i = \frac{(f_{i+1} - f_i)}{h} \tag{11}$$

$$\begin{pmatrix} f'_0 \\ f'_1 \\ \vdots \\ f'_{N-1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} -1 & 1 & 0 & \\ 0 & -1 & 1 & \\ & & \ddots & \ddots \\ & & & -1 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} \tag{12}$$

It is noted that the last entry in the matrix will not be correct because there is no element for **N**. This is fixed by taking the backward derivative at the last point.

Extending this matrix equation for the second derivative:

$$\begin{pmatrix} f''_0 \\ f''_1 \\ f''_2 \vdots \\ f''_{N-1} \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} -2 & 1 & 0 & 0 & \\ 0 & -2 & 1 & 0 & \\ 0 & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & -2 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} \tag{13}$$

Take note however that at both ends of the array it would yield us inaccurate answers, this was adjusted by using the same elements as the row below (at the start) or the row above (at the end), thus giving us $f''_0 = f''_1$ and $f''_{N-1} = f''_{N-2}$.

3.3 Application of FDM on the Time-Independent Schrodinger Equation

Setting up the Schrodinger Equation as a matrix equation:

$$\hat{H} = \frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V \tag{14}$$

$$\hat{H}\Psi(x) = E\Psi(x) \tag{15}$$

The matrix for the potential is simply the values of the potential on the diagonal of the matrix: $V_{i=j} = V_i$

Writing out the matrix for **H**:

$$H = \frac{-\hbar^2}{2mh^2} \begin{pmatrix} -2 & 1 & 0 & 0 & & \\ 1 & -2 & 1 & 0 & & \\ 0 & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & \end{pmatrix} + \begin{pmatrix} V_0 & 0 & 0 & & \\ 0 & V_1 & 0 & & \\ 0 & 0 & V_2 & & \\ & & & \ddots & \\ & & & & V_{N-1} \end{pmatrix} \tag{16}$$

The simplest system to solve for is the infinite square well, for which $V = 0$. From known textbooks (refer to griffiths), it is readily recognizable that the results are alternating cos(x) and sin(x) functions, with energy levels:

$$E_i = \frac{n^2\pi^2\hbar^2}{2ma^2} \tag{17}$$

A subtlety must be discussed approaching this problem. The infinite square well from $\frac{-a}{2}$ to $\frac{a}{2}$ has $V = \infty$ at these points. A necessary implementation was done to avoid this, which was to limit the computational space from $\frac{-a}{2} + h$ to $\frac{a}{2} - h$, where h is the step size. This way the wavefunction is forced to zero at the end points. With this the parameters can now be set, and the Finite Difference Method can be implemented to the problem.

4 Results and Discussion

4.1 Quantitative Analysis of the Energy Levels in the Infinite Square Well

With the parameters set, particularly with $N = 4096$, FDM was implemented on the infinite square well, with the first six energy levels obtained and plotted in Figure 2. The computation time was determined to be around 36.5 seconds, this is due to the relatively large number of N. Take note that as N gets larger, the results for the energy levels become more accurate. Setting it to 2048 yielded faster results with 4.6 seconds computation time but provided much less accurate results. After five iterations, the computational error for the first six energy levels was determined.

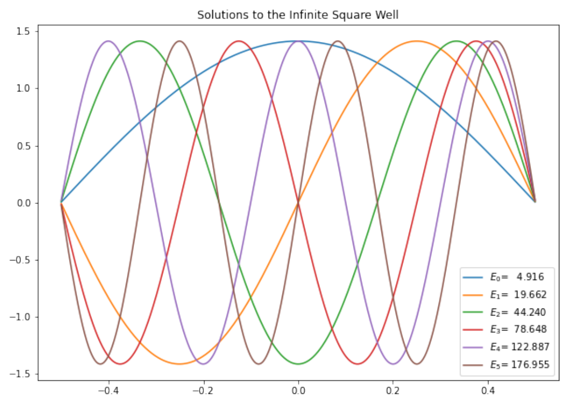
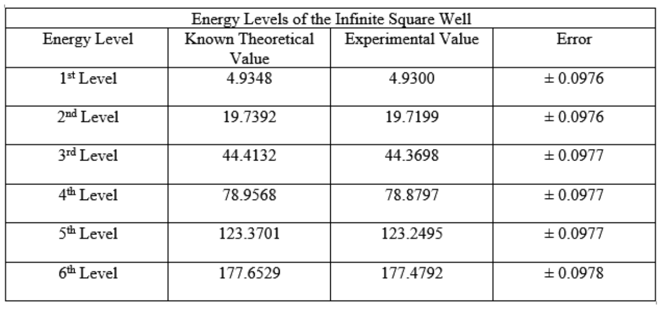


Figure 2: Energy level plot of the infinite square well obtained by through FDM.



Energy Levels of the Infinite Square Well			
Energy Level	Known Theoretical Value	Experimental Value	Error
1st Level	4.9348	4.9300	± 0.0976
2nd Level	19.7392	19.7199	± 0.0976
3rd Level	44.4132	44.3698	± 0.0977
4th Level	78.9568	78.8797	± 0.0977
5th Level	123.3701	123.2495	± 0.0977
6th Level	177.6529	177.4792	± 0.0978

Table 1. Energy level plot of the infinite square well obtained by through FDM.

The relative error between the known theoretical value of the energy levels in the infinite square well and the value obtained through FDM is shown in Table 1.

5 Conclusion and Recommendations

5.1 Conclusion

It has been shown that the Finite Difference Methods is a useful tool in solving an eigenvalue equation such as the Time-Independent Schrodinger Equation. The implementation of such methods on the Infinite Square Well problems showed satisfactory results and with a computation time of 36.5 seconds.

5.2 Recommendations

The code can also be applied to other fundamental one-dimensional models such as calculating the bound states of a finite well, a harmonic oscillator, and the hydrogen atom simply by changing the potential energy of the system.

References

[1] L. Edsberg. *Introduction to computation and modeling for differential equations.* John Wiley & Sons, 2015.

[2] D. J. Griffiths and D. F. Schroeter. *Introduction to quantum mechanics.* Cambridge University Press, 2018.

[3] A. Iserles. *A first course in the numerical analysis of differential equations.* Number 44. Cambridge university press, 2009.

[4] R. H. Landau, M. J. Pi, C. C. Bordeianu, et al. *Computational physics: Problem solving with Python.* John Wiley & Sons.

6 Appendix

6.1 Implementation of FDM on sin(x)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
# Define the number of points in our space, n

n = 128
a = 2*np.pi

# Define the x space from 0 to a with n-1 divisions

x = np.linspace(0, 2*np.pi, n)

# Storing the step size

h = x[1] - x[0] # Equal to 2*np.pi/(n-1)

# Compute for the function y = sin(x)

y = np.sin(x)

# Compute the matrix using np.diag(np.ones(n), 0) which will create
# a diagonal matrix of ones (nxn size)
# Multiply this by -1 to get -1 diagonal array as shown in the markdown above
# We then get a +1 off-diagonal array of ones, using np.diag(np.ones(n-1),1)
# Which needs to be n-1 for an nxn, since the off diagonal is one smaller
# Combining the two through addition and then normalize by 1/h

Md = 1./h*(np.diag( -1.*np.ones(n),0) + np.diag(np.ones(n-1),1))

# Compute the derivative of y into yp by matrix multiplication:

yp = Md.dot(y)

# Plot the results.

fig, ax = plt.subplots(figsize=(10,7))
plt.title('Derivative of sin(x)')
ax.plot(x,y, 'r', label = 'sin(x)')
ax.plot(x[:-1],yp[:-1], 'b', label = 'First derivative of sin(x)') # Don't plot last value, which is invalid
legend = ax.legend(loc = 'upper center', shadow = True, fontsize ='large')
plt.grid(True)
plt.savefig('first_derivative.png')
plt.show()

#For the second-order derivative

Mdd = 1./(h*h)*(np.diag(np.ones(n-1),-1) + np.diag( -2.*np.ones(n),0) + np.diag(np.ones(n-1),1))
print(Mdd)
ypp = Mdd.dot(y)

# Plot the results.

fig, ax = plt.subplots(figsize=(10,7))
plt.title('Second Derivative of sin(x)')
ax.plot(x,y, 'r', label = 'sin(x)')
ax.plot(x[:-1],yp[:-1], 'b', label = 'First derivative of sin(x)') # Don't plot last value, which is invalid
ax.plot(x[1:-1], ypp[1:-1] , 'g', label = 'Second derivative of sin(x)')
legend = ax.legend(loc = 'upper center', shadow = True, fontsize ='large')
plt.grid(True)
plt.savefig('second_derivative.png')
plt.show()
```

6.2 Implementation of FDM on the Infinite Square Well

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as scl
start = time.time()

iterations = 5
E_1 = []
E_2 = []
E_3 = []
E_4 = []
E_5 = []
for _ in range(iterations):

    # setting up our parameters

    hbar=1
    m=1
    N = 4096
    a = 1.0
    x = np.linspace(-a/2.,a/2.,N)
    #x = np.sort((2. * np.random.rand(N) - 1.) / 2.)
    # We want to store step size, this is the reliable way:
    h = x[1]-x[0] # Should be equal to 2*np.pi/(N-1)
    V = 0.*x

    #implementing the FDM

    Mdd = 1./(h*h)*(np.diag(np.ones(N-1),-1) -2* np.diag(np.ones(N),0) + np.diag(np.ones(N-1),1))
    H = -(hbar*hbar)/(2.0*m)*Mdd + np.diag(V)
    E,psiT = np.linalg.eigh(H) # This computes the eigen values and eigenvectors
    # obtaining the values for the first 5 energy levels to calculate the error
    E_1.append(E[0]) # first energy level
    E_2.append(E[1]) # second
    E_3.append(E[2]) # third
    E_4.append(E[3]) # fourth
    E_5.append(E[4]) # fifth
    psi = np.transpose(psiT)   # We take the transpose of psiT to the wavefunction vectors can accessed as psi
end = time.time()-start
print(f'Total time: {end} seconds')
plt.figure(figsize=(10,7))
for i in range(5):
    if psi[i][N-10] < 0:   # Flip the wavefunctions if it is negative at large x, so plots are more consistent.
        plt.plot(x,-psi[i]/np.sqrt(h),label="$E_{}$={:>8.3f}".format(i,E[i]))
    else:
        plt.plot(x,psi[i]/np.sqrt(h),label="$E_{}$={:>8.3f}".format(i,E[i]))
    plt.title("Solutions to the Infinite Square Well")
plt.legend()
plt.savefig("Infinite_Square_Well_WaveFunctions.png")
plt.show()
```

6.3 Error Analysis

```
#First we compute for the theoretical values of the energy levels in this problem

theoretical_val = []
for i in range(10):
    n = i+1
    theoretical_val.append(n*n*np.pi**2*hbar*hbar/(2*m*a*a))

# We then compare it with our experimental results

for i in range(10):
    n = i+1
    print("E[{}] = {:9.4f}, E_{} ={:9.4f}".format(n,E[i],n, theoretical_val[i]))

# Calculating the error between experimental and theoretical value

error = [abs(i-j)/i*100 for i,j in zip(theoretical_val, E)]

for j in range(10):
    print("E[{}] = {:9.4f} +- {:9.4f}".format(j+1, E[j], error[j]))

# A final test to show the accuracy of the calculation in the orthonormality of the states

for j in range(5):
    for i in range(5):
        print("{:16.9e}".format(np.sum(psi[j]*psi[i])))
```