Finding Eigenenergies of Linear Half-Potential

(Numerical Integration of Time-Independent Schrodinger Equation)

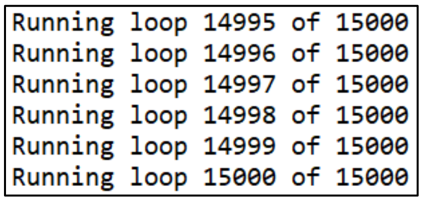
Jimmy Lilly

1 May 2020

PHYS 472

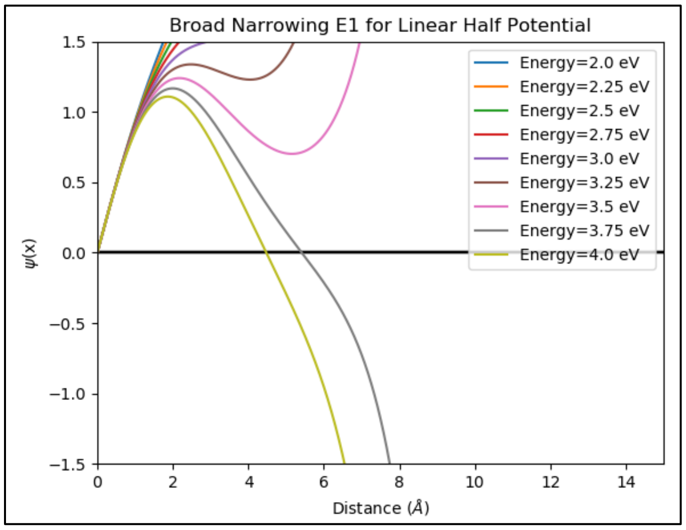
Prof. Shawn Jackson

1. Typical Output
    a. Command Line printouts
        i. Simply prints which loop is running to track code's progress

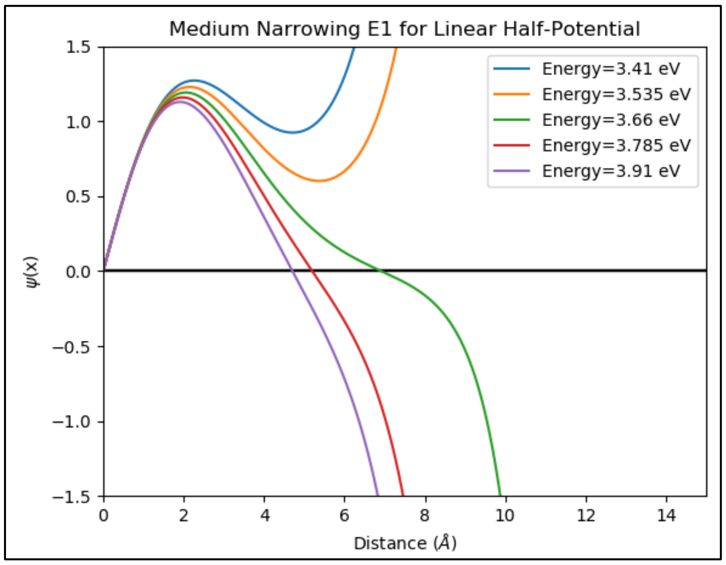


```
Running loop 14995 of 15000
Running loop 14996 of 15000
Running loop 14997 of 15000
Running loop 14998 of 15000
Running loop 14999 of 15000
Running loop 15000 of 15000
```

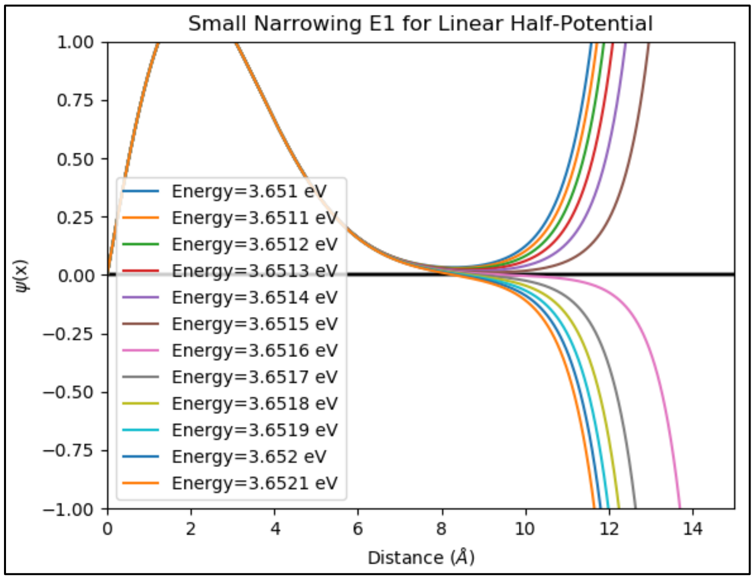
2. Typical Plots
    a. BigFishing plots wavefunctions,$\psi(x)$, for broad range of trial energies



    b. MediumFishing plots $\psi(x)$ for narrower range of trial energies



c. SmallFishing plots $\psi(x)$ for very small range of trial energies



3. Full Results

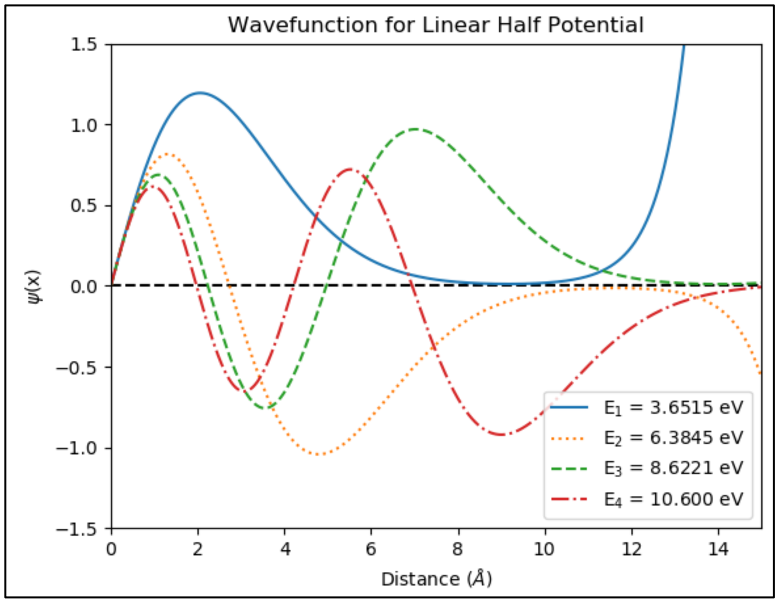
a. Values of important variables

i. $\beta = 0.26246\left(eV \cdot \text{Å}^2\right)^{-1}$ for $\frac{d^2\psi}{dx^2} = \beta(V(x) - E)\psi$

ii. Using $V(x) = \alpha x$ for x>0 where $\alpha = 1.0\,\frac{eV}{\text{Å}}$

b. Full wavefunctions for first four eigenenergies

i. $E_1 = 3.6515\ eV; E_2 = 6.3845\ eV; E_3 = 8.6221; E_4 = 10.600\ eV$



```
"""
Author: Jimmy Lilly (www.github.com/jlilly364)

Program Objective: Numerical Integration of the Time-Independent Schrodinger
                   Equation for a 1D Linear Half-Potential V(x)=alpha*x
"""
####
# V(x)=alpha*x for x>0, infinite elsewhere
# E_3 should be ~8.6217 eV
####

# Import relevant modules
import matplotlib.pyplot as plt
import numpy as np

# Define functional form of time-independent potential: V(x)
def Potential(x,alpha,exponent):
    # Inputs:
    #    x = value to evaluate potential at
    #    alpha = multiplicative constant for potential
    #    exponent = potential's exponential dependence on x
    # Returns:
    #    value of potential at given x value

    function = alpha*(x**exponent)
    return function

# Potential function for testing 'Solver': V(x)=0
def Test(x):
    # Inputs:
    #    x = distance from left wall
    # Returns:
    #    0 for all x

    return 0*x

# Function to solve TISE
def Solver(stepsize,upper,energy):
    # Inputs:
    #     stepsize = size of deltaX for iterating loop (in angstroms)
    #     upper = upper limit for integration (in angstroms)
    #     energy = trial energy
    #     even = boolean for if wavefunction is even (True) or odd (False)
    # Returns:
    #     xrange = x values at which Psi was evaluated
    #     Psi_array = wavefunction evaluated for all values in xrange

    # Establish initial conditions
    Psi = 0.0
    dPsi = 1.0

    # 2m/hbar^2 in natural units
    beta = 0.26246 # in (ev*angstroms^2)^-1

    # Set up array to add Psi values too
    Psi_array = []
    # Establish list of x values
    xrange = np.arange(0,upper+stepsize,stepsize)

    # Calculate how many loops will run
    numLoops = int(upper/stepsize)

    # Loop to calculate important quantities
    for i in range(0,len(xrange)):

        # Tell user which loop is running
        print('Running loop {0} of {1}'.format(i,numLoops))

        # Calc. 2nd derivative of Psi at x0 (0)
        d2Psi = beta*(Potential(xrange[i],1.0,1) - energy)*Psi

        # What to add to wavefunction
        extra = ((dPsi*stepsize) + ((d2Psi*(stepsize**2))/2))

        # Calc. new value of dPsi: dPsi(x0+deltaX)
        dPsi += d2Psi*stepsize

        # Calc. new value of Psi: Psi(x0+deltaX) & add to array
        Psi += extra
        Psi_array.append(Psi)

    # Add initial value of Psi to array of Psi values
    np.insert(Psi_array,1,Psi)

    return xrange,Psi_array

# Function to find general locations of eigenenergies
def BigFishing(central,buffer=1.00,step=0.25):
    # Inputs:
    #    central = reference energy to check wavefn. around (in eV)
    #    buffer = region around cental energy to test (in eV)
    #    step = interval between test energies (in eV)
    # Returns:
    #    plot of wavefunction for trial energies

    # Make list of energies to run through
    energies = np.arange(central-buffer,central+buffer+step,step)
    x = [[] for i in range(len(energies))]
    y = [[] for i in range(len(energies))]

    # Set values to plot for each test energy
    for i in range(0,len(energies)):
        x[i],y[i] = Solver(.001,15.0,energies[i])

    # Plot wavefunctions for each test energy
    for i in range(len(x)):
        plt.plot(x[i],y[i],label='Energy={0} eV'.format(energies[i]))
        plt.legend()

        # Set axis limits and labels/title
        plt.xlim(0,15)
        plt.ylim(-1.5,1.5)
        plt.xlabel(r'Distance ($\AA$)')
        plt.ylabel(r'$\psi$(x)')
        plt.title('Small Narrowing for Linear Half-Potential')

        # Draw horizontal line at Psi = 0
        plt.hlines(0,0,15)

        # Save figure
        #plt.savefig('C:/Users/Jimmy/Physics-Programs/PHYS 472/BigFishing.png')

    plt.show()

# Function to find more precise eigenenergies
def MediumFishing(central,buffer=.25,step=0.125):
    # Inputs:
    #    central = reference energy to check wavefn. around (in eV)
    #    buffer = region around cental energy to test (in eV)
    #    step = interval between test energies (in eV)
    # Returns:
    #    plot of wavefunction for trial energies

    # Make list of energies to run through
    energies = np.arange(central-buffer,central+buffer+step,step)
    x = [[] for i in range(len(energies))]
    y = [[] for i in range(len(energies))]

    # Set values to plot for each test energy
    for i in range(0,len(energies)):
        x[i],y[i] = Solver(.001,15.0,energies[i])

    # Plot wavefunctions for each test energy
    for i in range(len(x)):
        plt.plot(x[i],y[i],label='Energy={0} eV'.format(np.around(energies[i],4)))
        plt.legend()

        # Set axis limits and labels/title
        plt.xlim(0,15)
        plt.ylim(-1.5,1.5)
        plt.xlabel(r'Distance ($\AA$)')
        plt.ylabel(r'$\psi$(x)')
        plt.title('Medium Narrowing for Linear Half-Potential')

        # Draw horizontal line at Psi = 0
        plt.hlines(0,0,15)

        # Save figure
        #plt.savefig('C:/Users/Jimmy/Physics-Programs/PHYS 472/MediumFishing.png')
    plt.show()

# Function to find most precise eigenenergies
def SmallFishing(central,buffer=.003,step=0.001):
    # Inputs:
    #    central = reference energy to check wavefn. around (in eV)
    #    buffer = region around cental energy to test (in eV)
    #    step = interval between test energies (in eV)
    # Returns:
    #    plot of wavefunction for trial energies

    # Make list of energies to run through
    energies = np.arange(central-buffer,central+buffer+step,step)
    x = [[] for i in range(len(energies))]
    y = [[] for i in range(len(energies))]

    # Set values to plot for each test energy
    for i in range(0,len(energies)):
        x[i],y[i] = Solver(.001,15.0,energies[i])

    # Plot wavefunctions for each test energy
    for i in range(len(x)):
        plt.plot(x[i],y[i],label='Energy={0} eV'.format(np.around(energies[i],6)))
        plt.legend()

        # Set axis limits and labels/title
        plt.xlim(0,15)
        plt.ylim(-1.5,1.5)
        plt.xlabel(r'Distance ($\AA$)')
        plt.ylabel(r'$\psi$(x)')
        plt.title('Small Narrowing for Linear Half-Potential')

        # Draw horizontal line at Psi = 0
        plt.hlines(0,0,15)

        # Save figure
        #plt.savefig('C:/Users/Jimmy/Physics-Programs/PHYS 472/SmallFishing.png')
    plt.show()

# Call fishing functions to narrow down precise eigenenergies
#BigFishing(2.0)
#MediumFishing(2.50)
#SmallFishing(2.453)

# Plot of final eigenenergy wavefunctions
def FinalPlot(save=False):
    # Inputs:
    #     save = boolean to save (True)/not save (False) plot (default = False)
    # Returns:
    #     plot with wavefunctions of four lowest eigenenergies

    # Define energy eigenvalues found using fishing functions
    E1 = 3.6515
    E2 = 6.3845
    E3 = 8.6221
    E4 = 10.600

    # Save x values and save y (Psi) values from Solver function
    x1,y1 = Solver(0.001,15.0,E1)
    x2,y2 = Solver(0.001,15.0,E2)
    x3,y3 = Solver(0.001,15.0,E3)
    x4,y4 = Solver(0.001,15.0,E4)

    # Plot wavefunctions for four lowest eigenenergies
    plt.plot(x1,y1,label=r'E$_1$ = {0} eV'.format(E1),linestyle='solid')
    plt.plot(x2,y2,label=r'E$_2$ = {0} eV'.format(E2),linestyle='dotted')
    plt.plot(x3,y3,label=r'E$_3$ = {0} eV'.format(E3),linestyle='dashed')
    plt.plot(x4,y4,label=r'E$_4$ = %.3f eV' % E4,linestyle='dashdot')
    plt.legend()

    # Set axis limits and labels/title
    plt.xlim(0,15)
    plt.ylim(-1.5,1.5)
    plt.xlabel(r'Distance ($\AA$)')
    plt.ylabel(r'$\psi$(x)')
    plt.title('Wavefunction for Linear Half Potential')

    # Draw horizontal line at Psi = 0
    plt.hlines(0,0,15,linestyle='dashed')

    # Save plot if user chooses to
    if save == True:
        plt.savefig('C:/Users/Jimmy/Physics-Programs/PHYS 472/Linear Half Potential Energies')
    plt.show()

# Generate plot with wavefunctions of four lowest eigenenergies
FinalPlot(True)
```