

A large, hand-drawn orange circle with a textured, brushstroke-like appearance, centered on the page. The circle is composed of multiple overlapping strokes, giving it a dynamic and artistic feel.

# **STUDY OF PARTICLE IN A BOX**

# PROJECT REPORT

submitted in partial of requirement for the  
award of the degree of

## BACHELOR OF SCIENCE IN PHYSICS

(Model - II Computer Application)

OF



Mahatma Gandhi University, Kottayam



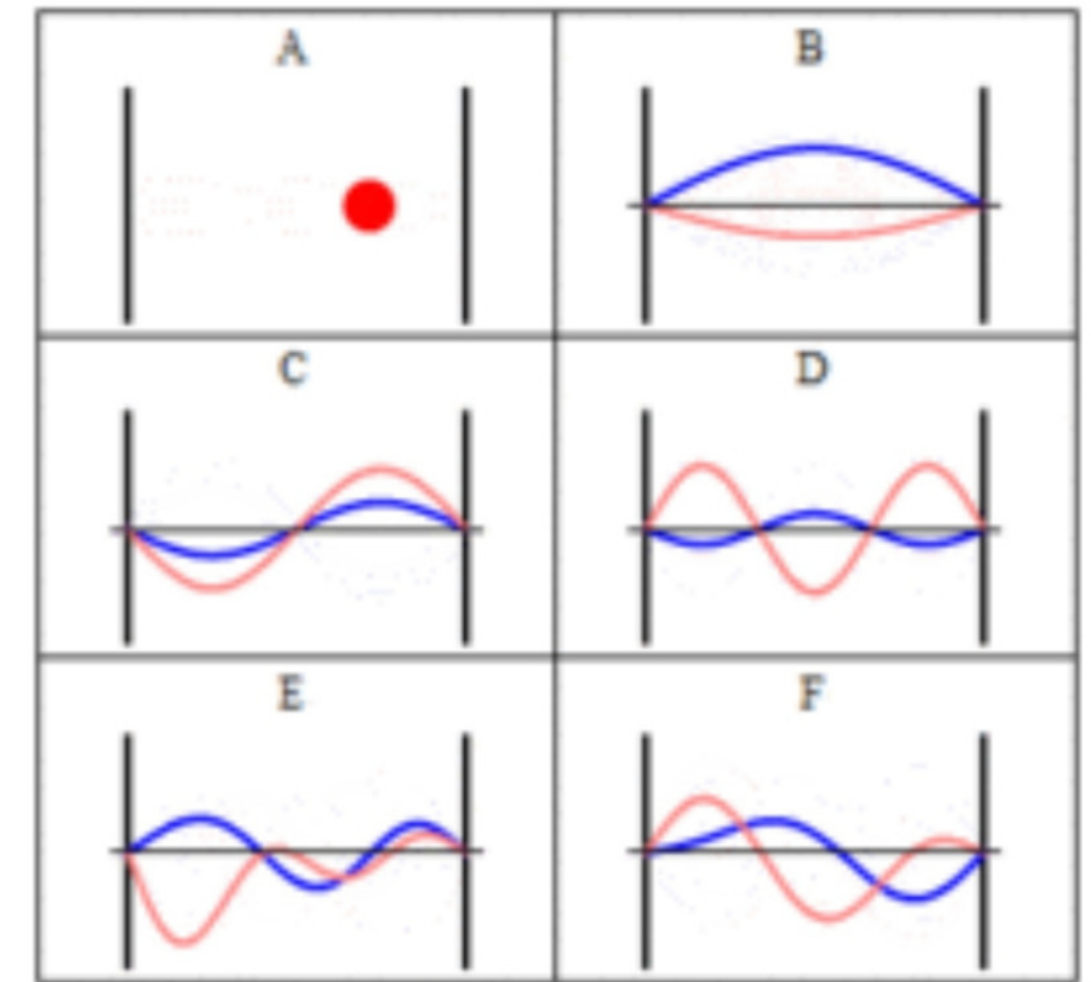
Bharata Mata College, Thrikkakkara

By  
Tony John  
Class no:19V1122  
Register no:190021042107

# Introduction

## DEFINITION

The particle in a box model (also known as the infinite potential well or the infinite square well) describes a particle free to move in a small space surrounded by impenetrable barriers. The particle in a box model is one of the very few problems in quantum mechanics which can be solved analytically, without approximations. Due to its simplicity, the model allows insight into quantum effects without the need for complicated mathematics. It serves as a simple illustration of how energy quantizations (energy levels), which are found in more complicated quantum systems such as atoms and molecules, come about. It is one of the first quantum mechanics problems taught in undergraduate physics courses, and it is commonly used as an approximation for more complicated quantum systems.



Some trajectories of a particle in a box according to Newton's laws of classical mechanics (A), and according to the Schrödinger equation of quantum mechanics (B–F). In (B–F), the horizontal axis is position, and the vertical axis is the real part (blue) and imaginary part (red) of the wavefunction. The states (B,C,D) are energy eigenstates, but (E,F) are not.

# **BENEFIT OF PYTHON PROGRAM**

1. Easy to read , Learn and Write.
- 2.Improved Productivity.
- 3.Interpreted Language.
- 4.Dynamically Typed.
- 5.Free and open source.
- 6.Vast Libraries Support.
- 7.Protability.

## **PARTICLE IN A BOX EXPERIMENTAL MODEL USING PYTHON**

Particle in a box model only applies to single electron atoms (Hydrogen) and it is used as a baseline to estimate electronic energies for other multi-electron atoms. Following functions are used to describe electronic energy, potential energy, and Hamiltonian function for particle in a box.

## Total Electronic Energy

$$E = \hbar\omega = \frac{\hbar^2 k^2}{2m}$$

Where

$$\hbar = h/2\pi$$

$h$  = Planck's constant

$$\omega = (k/2\mu)^{1/2}$$

$$\mu = (m_1 + m_2) / (m_1 m_2)$$

$k$  is the spring constant

Where  $m$  is the atomic mass of each atom

## Potential Energy

The potential energy of particle  $V(x)$  is infinite on both sides of box, while  $V(x)$  is a constant on the inside of the box. Since the particle cannot have an infinite amount of energy it cannot exist outside the box.

$$V(x) = \frac{1}{2} kx^2$$

Where

$V(x)$  is a potential energy function of position  $x$

$k$  is a spring constant

## Hamiltonian Function

Hamiltonian Function used to obtain wave function (Schrodinger Equation), combines Potential energy with kinetic (constant), so overall energy function is quadratic

$$i\hbar \frac{\partial}{\partial t} \psi(x, t) = -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \psi(x, t) + V(x) \psi(x, t)$$

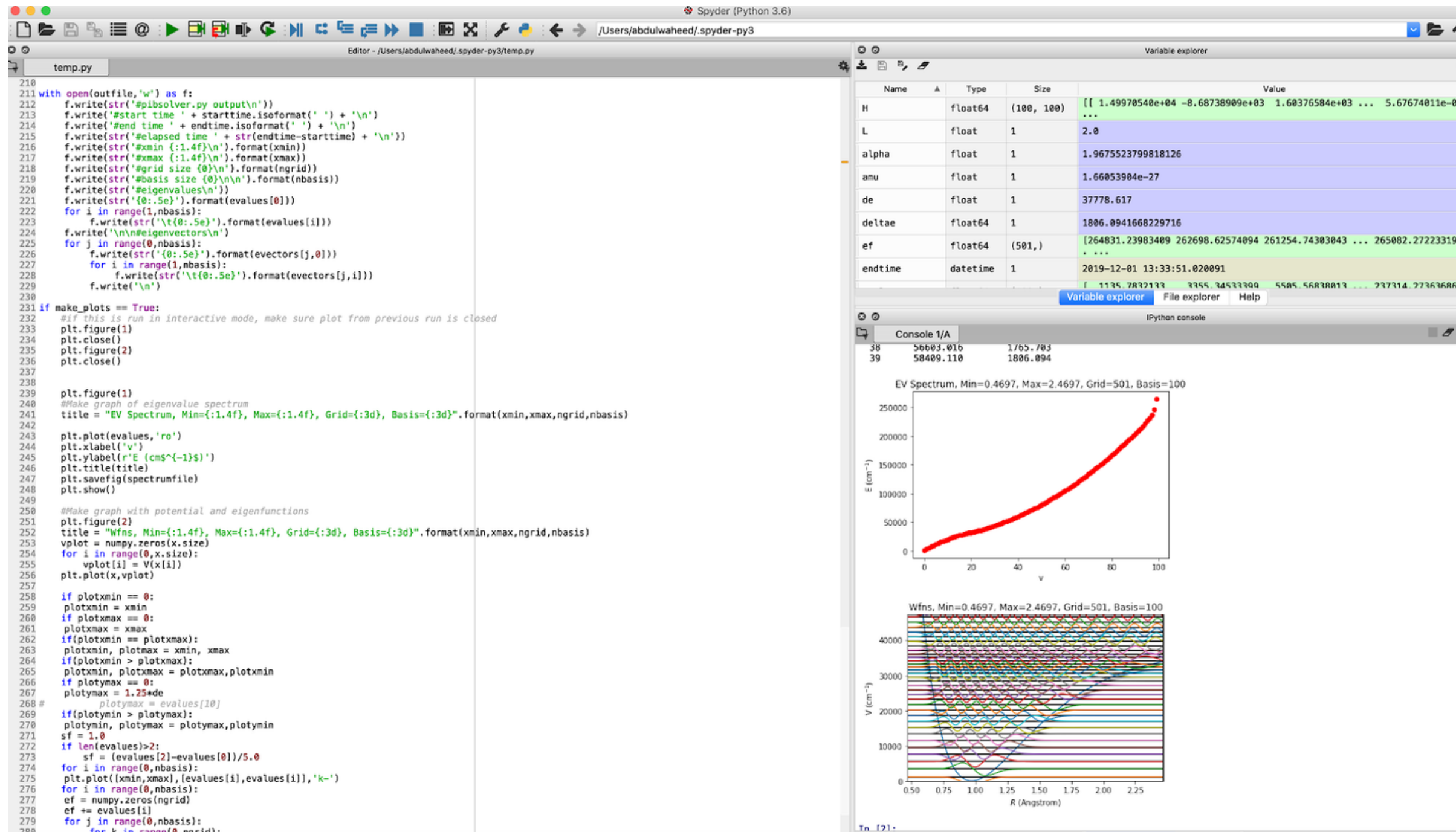
This equation is called the time dependent form of Schrödinger's equation for non-relativistic

## Schrödinger time dependent equation general form for particle in a box

$$\psi_n(x) = \begin{cases} \sqrt{\frac{2}{L}} \sin(k_n x) & \text{for } n \text{ even} \\ \sqrt{\frac{2}{L}} \cos(k_n x) & \text{for } n \text{ odd} \end{cases}$$



# PYTHON BASED ANALYSIS



Spyder is an open-source IDE for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open-source software.

# CODING

```
#!/usr/bin/python
#
# Solves the Schrodinger equation for a one-dimensional particle
# in an arbitrary potential using a finite basis of particle-
# in-a-box basis functions.
#
# Requires SciPy, NumPy, and matplotlib
#
# This script generates a text file containing the eigenvalues
# and eigenvectors, a pdf graph of the energy spectrum, and
# a pdf graph showing the potential, eigenvalues, and
# eigenfunctions.
#
#
#

import scipy as sp
import scipy.constants as spc
import scipy.integrate
import math
import numpy
from matplotlib import pyplot as plt
import datetime

#-----
#      VARIABLES      -
#-----

#equilibrium bond length in Angstrom
re = 0.96966

#effective mass in kg
amu = spc.physical_constants['atomic mass constant'][0]
mass = (1.*16./(1.+16.))*amu *2

#minimum x value for partice in a box calculation
xmin = -0.5+re

#maximum x value for particle in a box calculation
#there is no limit, but if xmax<xmin, the values will be swapped
#if xmax = xmin, then xmax will be set to xmin + 1
xmax = 1.5+re

#number of grid points at which to calculate integral
#must be an odd number. If an even number is given, 1 will be added.
#minimum number of grid points is 3
ngrid = 501

#number of PIB wavefunctions to include in basis set
#minimum is 1
```

```
nbasis = 100

#if you want to control the plot ranges, you can do so here
make_plots = True
plotymin = 0
plotymax = 0
plotxmin = 0
plotxmax = 0

#dissociation energy in cm-1
de = 37778.617

#force constant in N / m
fk = 774.7188418117737*0.75

#angular frequency in rad/s
omega = numpy.sqrt(fk/mass)

#output file for eigenvalues
outfile = "eigenvalues.txt"

#output PDF for energy spectrum
spectrumfile = "energy.pdf"

#output PDF for potential and eigenfunctions
potentialfile = "potential.pdf"

#-----
#  POTENTIAL DEFINITIONS  -
#-----

#definitions of potential functions
#particle in a box: no potential. Note: you need to manually set plotymin and
plotymax above for proper graphing
def box(x):
    return 0

#purely harmonic potential
#energy in cm^-1
prefactor = 0.5 * fk * 1e-20/(spc.h*spc.c*100.0)
def harmonic(x):
    return prefactor*(x-re)*(x-re)

#anharmonic potential
#def anharmonic(x):
#    return 0.5*(x**2.) + 0.05*(x**4.)

#Morse potential
#energy in cm^-1
```

```
#alpha in A^-1
alpha = math.sqrt(fk/2.0/(de*spc.h*spc.c*100.0))*1e-10
def morse(x):
    return de*(1.-numpy.exp(-alpha*(x-re)))**2.

#double well potential (minima at x = +/-3)
#def doublewell(x):
#    return x**4.-18.*x**2. + 81.

#potential function used for this calculation
def V(x):
    return morse(x)

#-----
#  BEGIN CALCULATION  -
#-----

#verify that inputs are sane
if xmax==xmin:
    xmax = xmin+1.
elif xmax<xmin:
    xmin,xmax = xmax,xmin

L = xmax - xmin

#function to compute normalized PIB wavefunction
tl = numpy.sqrt(2./L)
pixl = numpy.pi/L
def pib(x,n,L):
    return tl*math.sin(n*x*pixl)

ngrid = max(ngrid,3)

if not ngrid%2:
    ngrid+=1

nbasis = max(1,nbasis)

#get current time
starttime = datetime.datetime.now()

#create grid
x = numpy.linspace(xmin,xmax,ngrid)

#create Hamiltonian matrix; fill with zeros
H = numpy.zeros((nbasis,nbasis))

#split Hamiltonian into kinetic energy and potential energy terms
```



```

#H = T + V
#V is defined above, and will be evaluated numerically

#Compute kinetic energy
#The Kinetic energy operator is T = -hbar^2/2m d^2/dx^2
#in the PIB basis, T|phi(i)> = hbar^2/2m n^2pi^2/L^2 |phi(i)>
#Therefore <phi(k)|T|phi(i)> = hbar^2/2m n^2pi^2/L^2 delta_{ki}
#Kinetic energy is diagonal
kepf = spc.hbar*spc.hbar*math.pi**2./(2.*mass*(L*1e-10)*(L*1e-10)*spc.h*spc.c*100)
for i in range(0,nbasis):
    H[i,i] += kepf*(i+1.)*(i+1.)

#now, add in potential energy matrix elements <phi(j)|V|phi(i)>
#that is, multiply the two functions by the potential at each grid point and
integrate
for i in range(0,nbasis):
    for j in range(0,nbasis):
        if j >= i:
            y = numpy.zeros(ngrid)
            for k in range(0,ngrid):
                p = x[k]
                y[k] += pib(p-xmin,i+1.,L)*V(p)*pib(p-xmin,j+1.,L)
            H[i,j] += sp.integrate.simps(y,x)
        else:
            H[i,j] += H[j,i]

#Solve for eigenvalues and eigenvectors
evalues, evecors = sp.linalg.eigh(H)
evalues = evalues

#get ending time
endtime = datetime.datetime.now()

#-----
# GENERATE OUTPUT -
#-----

print("Results:")
print("-----")
print("  v  Energy (cm-1)  Delta E (cm-1) ")
print("-----")
for i in range(min(40,len(evalues))):
    if i>0:
        deltae = evalues[i] - evalues[i-1]
        print(" {:>3d}  {:>13.3f}  {:>14.3f} ".format(i,evalues[i],deltae))
    else:
        print(" {:>3d}  {:>13.3f}          ----- ".format(i,evalues[i]))

with open(outfile,'w') as f:
    f.write(str('#pibsolver.py output\n'))
    f.write('#start time ' + starttime.isoformat(' ') + '\n')
    f.write('#end time ' + endtime.isoformat(' ') + '\n')
    f.write(str('#elapsed time ' + str(endtime-starttime) + '\n'))
    f.write(str('#xmin {:1.4f}\n').format(xmin))
    f.write(str('#xmax {:1.4f}\n').format(xmax))
    f.write(str('#grid size {0}\n').format(ngrid))
    f.write(str('#basis size {0}\n\n').format(nbasis))

```

```

f.write(str('#eigenvalues\n'))
f.write(str('{0:.5e}').format(evalues[0]))
for i in range(1,nbasis):
    f.write(str('\t{0:.5e}').format(evalues[i]))
f.write('\n\n#eigenvectors\n')
for j in range(0,nbasis):
    f.write(str('{0:.5e}').format(evecors[j,0]))
    for i in range(1,nbasis):
        f.write(str('\t{0:.5e}').format(evecors[j,i]))
    f.write('\n')

if make_plots == True:
    #if this is run in interactive mode, make sure plot from previous run is closed
    plt.figure(1)
    plt.close()
    plt.figure(2)
    plt.close()

    plt.figure(1)
    #Make graph of eigenvalue spectrum
    title = "EV Spectrum, Min={:1.4f}, Max={:1.4f}, Grid={:3d},
Basis={:3d}".format(xmin,xmax,ngrid,nbasis)

    plt.plot(evalues,'ro')
    plt.xlabel('v')
    plt.ylabel(r'E (cm$^{-1}$)')
    plt.title(title)
    plt.savefig(spectrumfile)
    plt.show()

    #Make graph with potential and eigenfunctions
    plt.figure(2)
    title = "Wfns, Min={:1.4f}, Max={:1.4f}, Grid={:3d},
Basis={:3d}".format(xmin,xmax,ngrid,nbasis)
    vplot = numpy.zeros(x.size)
    for i in range(0,x.size):
        vplot[i] = V(x[i])
    plt.plot(x,vplot)

    if plotxmin == 0:
        plotxmin = xmin

    if plotxmax == 0:
        plotxmax = xmax
    if(plotxmin == plotxmax):
        plotxmin, plotmax = xmin, xmax
    if(plotxmin > plotxmax):
        plotxmin, plotxmax = plotxmax,plotxmin
    if plotymax == 0:
        plotymax = 1.25*de
    # plotymax = evalues[10]
    if(plotymin > plotymax):
        plotymin, plotymax = plotymax,plotymin
    sf = 1.0
    if len(evalues)>2:
        sf = (evalues[2]-evalues[0])/5.0
    for i in range(0,nbasis):
        plt.plot([xmin,xmax],[evalues[i],evalues[i]],'k-')

```

```

for i in range(0,nbasis):
    ef = numpy.zeros(ngrid)
    ef += evalues[i]
    for j in range(0,nbasis):
        for k in range(0,ngrid):
            ef[k] += evecors[j,i]*pib(x[k]-xmin,j+1,L)*sf

plt.plot(x,ef)

plt.plot([xmin,xmin],[plotymin,plotymax],'k-')
plt.plot([xmax,xmax],[plotymin,plotymax],'k-')
plt.axis([plotxmin,plotxmax,plotymin,plotymax])
plt.title(title)
plt.xlabel('$R$ (Angstrom)')
plt.ylabel(r'V (cm$^{-1}$)')
plt.savefig(potentialfile)
plt.show()

```



# OUTPUT

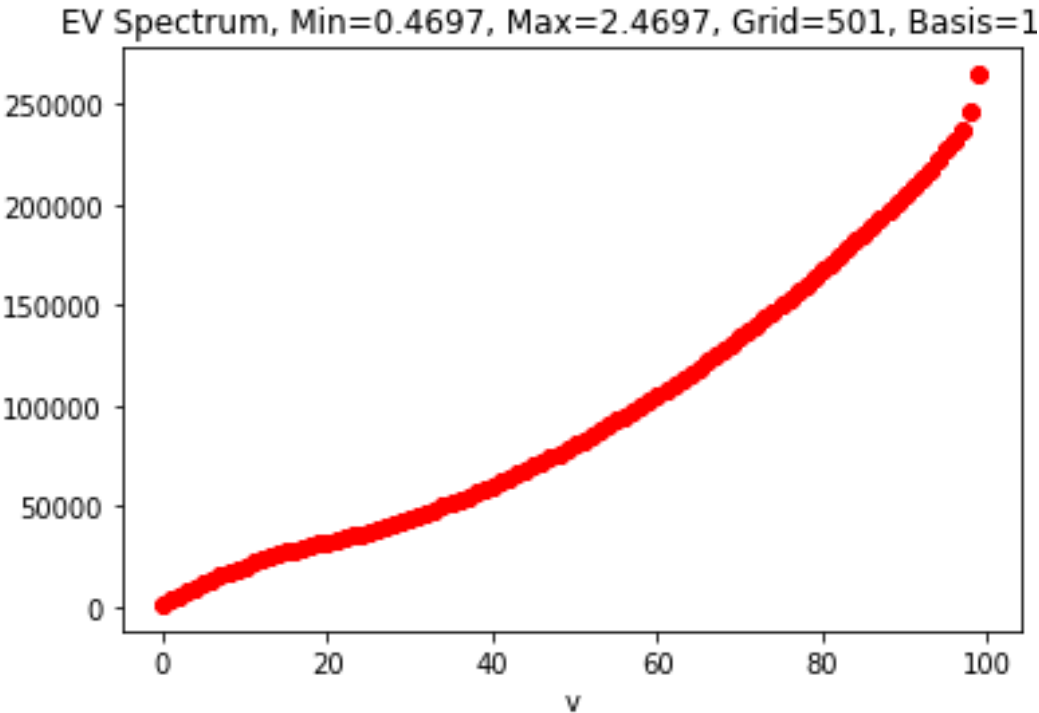


Figure 1

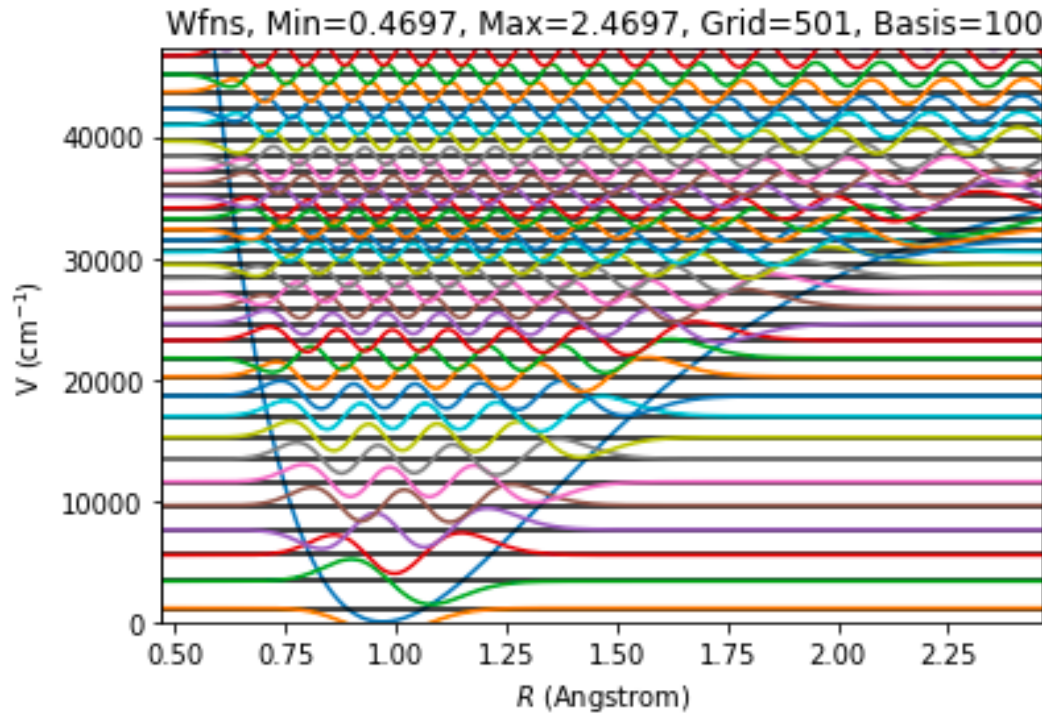


Figure 2

1	Results:		
2	-----		
3	v	Energy (cm-1)	Delta E (cm-1)
4	-----		
5	0	1135.783	-----
6	1	3355.345	2219.562
7	2	5505.568	2150.223
8	3	7586.452	2080.884
9	4	9597.997	2011.545
10	5	11540.203	1942.206
11	6	13413.070	1872.867
12	7	15216.597	1803.528
13	8	16950.786	1734.189
14	9	18615.635	1664.850
15	10	20211.146	1595.510
16	11	21737.317	1526.171
17	12	23194.150	1456.832
18	13	24581.643	1387.493
19	14	25899.797	1318.154
20	15	27148.612	1248.815
21	16	28328.088	1179.476
22	17	29438.225	1110.137
23	18	30479.036	1040.810
24	19	31450.802	971.766
25	20	32357.264	906.462
26	21	33221.864	864.600
27	22	34103.504	881.640
28	23	35055.285	951.781
29	24	36089.774	1034.489
30	25	37200.613	1110.839
31	26	38380.000	1179.386
32	27	39621.923	1241.923
33	28	40921.915	1299.992
34	29	42276.562	1354.647
35	30	43683.170	1406.608
36	31	45139.549	1456.379
37	32	46643.879	1504.330
38	33	48194.615	1550.736
39	34	49790.425	1595.810
40	35	51430.145	1639.720
41	36	53112.747	1682.602
42	37	54837.313	1724.565
43	38	56603.016	1765.703
44	39	58409.110	1806.094
45			

# CONCLUSION

The data table the graph show the energy interactions that takes place as electrons transition between energy levels, starting from the ground state (0 point)(figure 1). Each of the mini functions straddling the parabola are harmonic wave functions that depict the individual electronic wave functions in each harmonic. These functions are important because they show individual energy transitions, which occur between one energy level at a time. This is because electrons can not transition between multiple energy levels at once. The main parabola (figure 2) is a function of atomic radius between the nuclei of bonded atoms. This function is the sum of all the individual harmonic functions. Overall, as energy levels increase, so does the energy transitions as shown in this graph (figure 1).

Thank You