

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



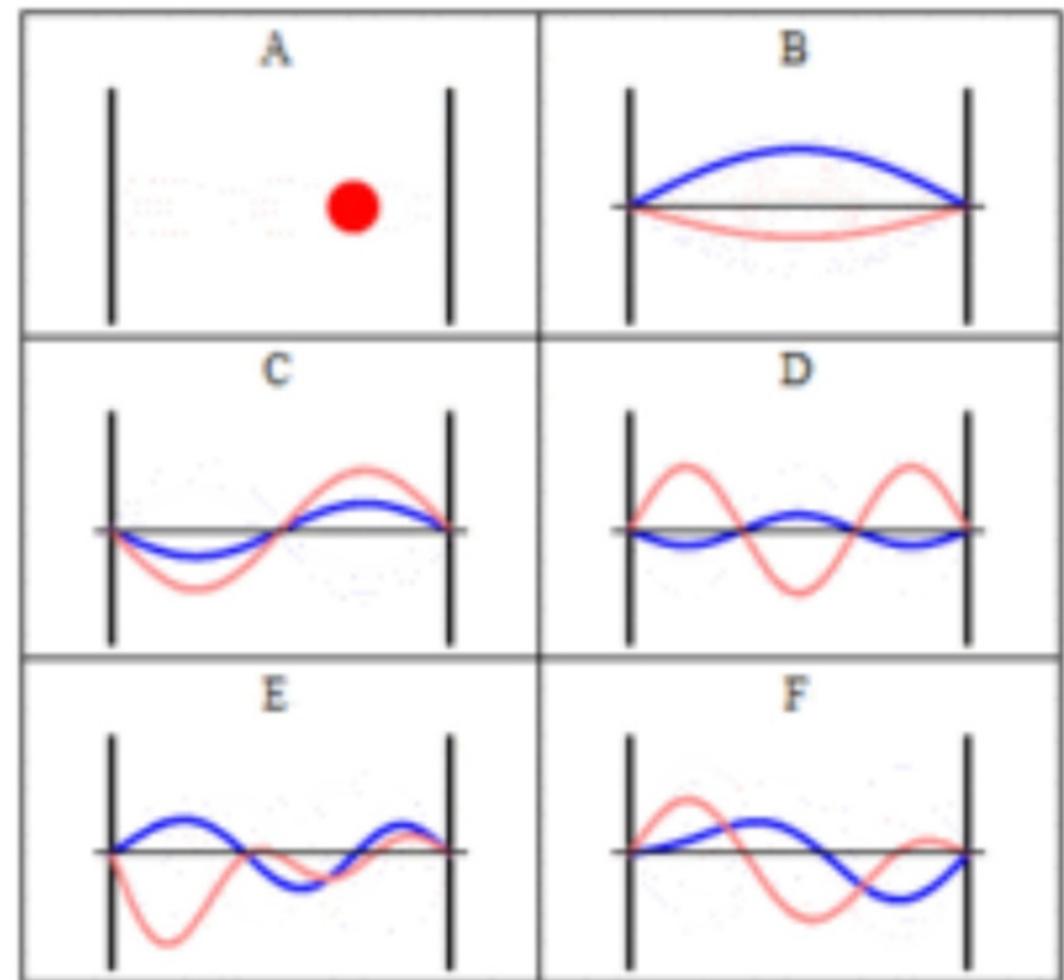
Bharata Mata College, Thrikkakkara

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

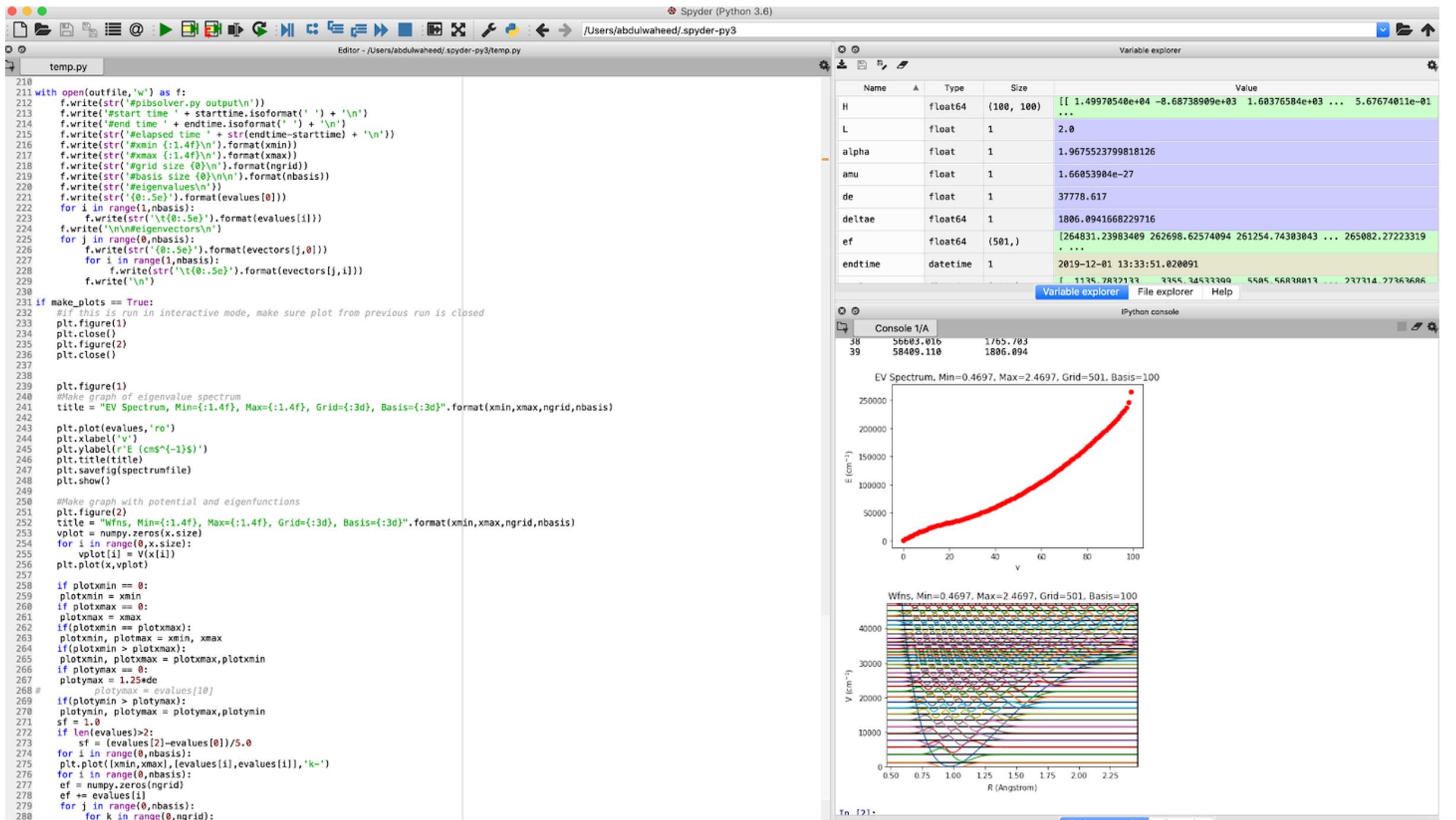
$$i\hbar \frac{\partial}{\partial t} \psi(x, t) = -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \psi(x, t) + V(x) \psi(x, t)$$

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



The screenshot shows the Spyder Python IDE interface. The code editor on the left contains a Python script named 'temp.py' with approximately 280 lines of code. The script includes sections for writing data to a file, setting up plots, and generating an eigenvalue spectrum and potential/eigenfunction plots. The variable explorer in the top right shows various variables like H, L, alpha, amu, de, deltae, ef, and endtime. Below the variable explorer is a Python console window showing command-line interactions. Two plots are displayed: 'EV Spectrum, Min=0.4697, Max=2.4697, Grid=501, Basis=100' showing a red curve, and 'Wfns, Min=0.4697, Max=2.4697, Grid=501, Basis=100' showing multiple colored lines representing wave functions.

```
210
211 with open(outfile,'w') as f:
212     f.write(str('#pibsolver.py output\n'))
213     f.write('#start time ' + starttime.isoformat(' ') + '\n')
214     f.write('#end time ' + endtime.isoformat(' ') + '\n')
215     f.write(str('#elapsed time ' + str(endtime-starttime) + '\n'))
216     f.write(str('#xmin {1.4f}\n').format(xmin))
217     f.write(str('#xmax {1.4f}\n').format(xmax))
218     f.write(str('#grid size {0}\n').format(grid))
219     f.write(str('#basis size {0}\n').format(nbasis))
220     f.write(str('#eigenvalues\n'))
221     f.write(str('{0:.5e}\n').format(values[0]))
222     for i in range(1,nbasis):
223         f.write(str('{0:.5e}\n').format(values[i]))
224     f.write(str('#eigenvectors\n'))
225     for j in range(0,nbasis):
226         f.write(str('{0:.5e}\n').format(evectors[j,0]))
227         for i in range(1,nbasis):
228             f.write(str('{0:.5e}\n').format(evectors[j,i]))
229     f.write('\n')
230
231 if make_plots == True:
232     #if this is run in interactive mode, make sure plot from previous run is closed
233     plt.figure(1)
234     plt.close()
235     plt.figure(2)
236     plt.close()
237
238
239 plt.figure(1)
240 #Make graph of eigenvalue spectrum
241 title = "EV Spectrum, Min={:1.4f}, Max={:1.4f}, Grid={:3d}, Basis={:3d}".format(xmin,xmax,ngrid,nbasis)
242
243 plt.plot(values,'ro')
244 plt.xlabel('v')
245 plt.ylabel('E (cm^-1)')
246 plt.title(title)
247 plt.savefig(spectrumfile)
248 plt.show()
249
250 #Make graph with potential and eigenfunctions
251 plt.figure(2)
252 title = "Wfns, Min={:1.4f}, Max={:1.4f}, Grid={:3d}, Basis={:3d}".format(xmin,xmax,ngrid,nbasis)
253 vplot = numpy.zeros(x.size)
254 for i in range(0,x.size):
255     vplot[i] = V(x[i])
256 plt.plot(x,vplot)
257
258 if plotxmin == 0:
259     plotxmin = xmin
260 if plotxmax == 0:
261     plotxmax = xmax
262 if(plotxmin == plotxmax):
263     plotxmin, plotmax = xmin, xmax
264 if(plotxmin > plotxmax):
265     plotxmin, plotxmax = plotxmax,plotxmin
266 if plotmax == 0:
267     plotmax = 1.25*de
268 #     plotymax = evals[10]
269 if(plotymax > plotymax):
270     plotymin, plotymax = plotymax,plotymin
271 sf = 1.0
272 if len(evals)>2:
273     sf = (evals[2]-evals[0])/5.0
274 for i in range(0,nbasis):
275     plt.plot([xmin,xmax],[evals[i],evals[i]],'k-')
276 for i in range(0,nbasis):
277     ef = numpy.zeros(ngrid)
278     ef += evals[i]
279     for j in range(0,nbasis):
280         for k in range(0,ngrid):
```



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 particle 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
t1 = numpy.sqrt(2./L)
pixl = numpy.pi/L
def pib(x,n,L):
    return t1*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)
        else:
            H[i,j] += H[j,i]

#Solve for eigenvalues and eigenvectors
evals, eigvecs = sp.linalg.eigh(H)
evals = evals

#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(evals))):
    if i>0:
        deltae = evals[i] - evals[i-1]
        print(" {:>3d} {:>13.3f} {:>14.3f} ".format(i,evals[i],deltae))
    else:
        print(" {:>3d} {:>13.3f} ----- ".format(i,evals[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').format(nbasis))

```

```

f.write(str('#eigenvalues\n'))
f.write(str('{0:.5e}').format(evals[0]))
for i in range(1,nbasis):
    f.write(str('\t{0:.5e}').format(evals[i]))
f.write('\n\n#eigenvectors\n')
for j in range(0,nbasis):
    f.write(str('{0:.5e}').format(evecs[j,0]))
    for i in range(1,nbasis):
        f.write(str('\t{0:.5e}').format(evecs[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(evals,'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 = evals[10]
    if(plotymin > plotymax):
        plotymin, plotymax = plotymax,plotymin
    sf = 1.0
    if len(evals)>2:
        sf = (evals[2]-evals[0])/5.0
    for i in range(0,nbasis):
        plt.plot([xmin,xmax],[evals[i],evals[i]],'k-')

```

```

for i in range(0,nbasis):
    ef = numpy.zeros(ngrid)
    ef += evals[i]
    for j in range(0,nbasis):
        for k in range(0,ngrid):
            ef[k] += evecs[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

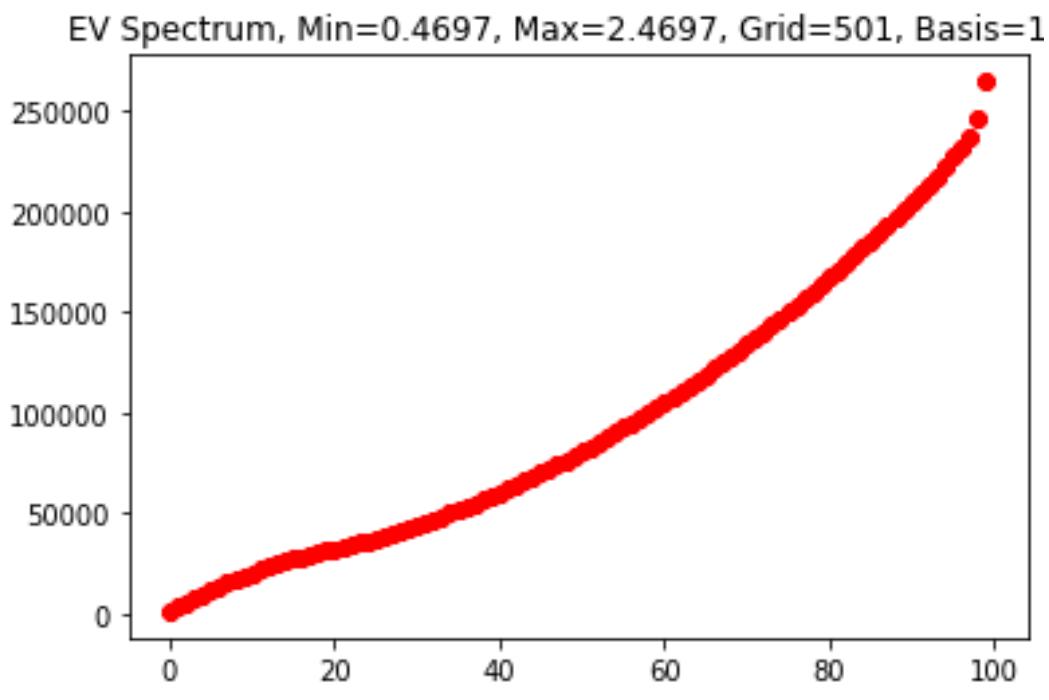


Figure 1

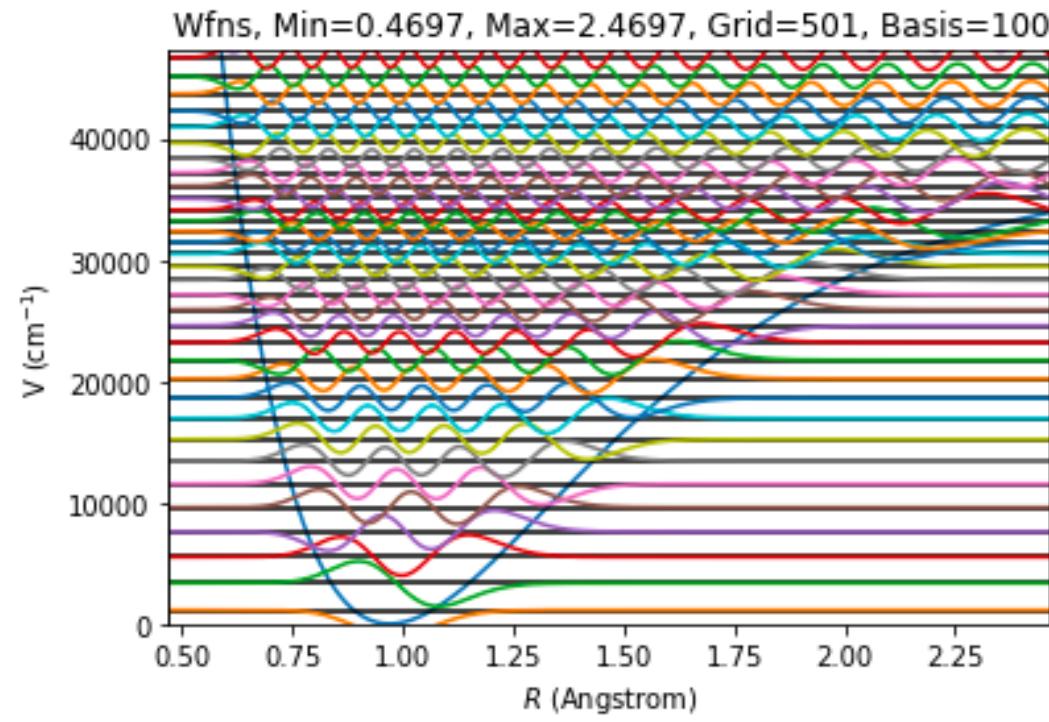


Figure 2

1 Results:		
2	3	4
v	Energy (cm ⁻¹)	Delta E (cm ⁻¹)
5	0	1135.783
6	1	3355.345
7	2	5505.568
8	3	7586.452
9	4	9597.997
10	5	11540.203
11	6	13413.070
12	7	15216.597
13	8	16950.786
14	9	18615.635
15	10	20211.146
16	11	21737.317
17	12	23194.150
18	13	24581.643
19	14	25899.797
20	15	27148.612
21	16	28328.088
22	17	29438.225
23	18	30479.036
24	19	31450.802
25	20	32357.264
26	21	33221.864
27	22	34103.504
28	23	35055.285
29	24	36089.774
30	25	37200.613
31	26	38380.000
32	27	39621.923
33	28	40921.915
34	29	42276.562
35	30	43683.170
36	31	45139.549
37	32	46643.879
38	33	48194.615
39	34	49790.425
40	35	51430.145
41	36	53112.747
42	37	54837.313
43	38	56603.016
44	39	58409.110
45		1806.094

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