STUDY OF PARTICLE IN A BOX

PROJECT REPORT

submitted In partial fulfillment of requirements for the award
of the degree of
BACHELOR OF SCIENCE IN PHYSICS
(Model - II Computer application)

Under the guidance

DEPARTMENT OF PHYSICS
BHARATA MATA COLLEGE THRIKKAKARA, KOCHI-21

MAHATMA GANDHI UNIVERSITY, KOTTAYAM

By
TONY JOHN
CLASS NO : 19V1122
REGISTER NO:190021042107
2019-2022 BATCH

BHARATA MATA COLLEGE, THRIKKAKARA, KOCHI-21

DEPARTMENT OF PHYSICS

CERTIFICATE

Certificate that this document titled "STUDY OF PARTICLE IN A BOX" is a bona-fide of the project report presented by TONY JOHN (university reg.no:190021042107) of sixth semester BSc Physics, model II Computer application submitted In partial fulfillment of the requirements for the award of the degree of Bachelor Of Science In Physics (Model - II Computer Application) of the Mahatma Gandhi university, Kottayam during the academic year 2019-2022.

Project Guide
Dr Riju K Thomas

Head of the department
Dr Anu Philip

Internal Examiner

External Examiner

STUDY OF PARTICLE IN A BOX

Project submitted to the
MAHATMA GANDHI UNIVERSITY,
KOTTAYAM
In partial fulfillment of requirements for the award of the degree of
BACHELOR OF SCIENCE IN PHYSICS
(Model - II Computer application),
BY

TONY JOHN
REG.NO:190021042107
DEPARTMENT OF PHYSICS
BHARATA MATA COLLEGE,
THRIKKAKARA
(2019-2022)

DECLARATION

I hereby declare that this project report entitled "STUDY OF PARTICLE IN A BOX" is based on the original work carried out by me under the supervision of Dr. ANU PHILIP in the department of Physics, Bharata Mata College, Thrikkakkara.

PLACE: THRIKKAKARA
DATE:

ACKNOWLEDGEMENT

The success and final outcome of this project report required a lot of guidance andassistance from many people and I am extremely fortunate to have got this all along the completion of my work. Whatever I have done is only due to such guidance and assistance and I would not forget to thank them.

First of all, I would like to thank God almighty for his divine grace and blessings throughout the course of this work.

I heartily thank our principal Dr. Shiny Palatty , & our Head Of the Department Dr.Anu philip for their guidance and suggestions during this project period.

I owe my profound gratitude to our project guide .Dr. Riju K Thomas who took keen interest in the project work and guided us all along, till the completion of our project by providing all the necessary information for developing a good system and I would like to extend my sincere gratitude to all my faculties for their support and guidance for the completion of my work.

I would also like to extend my sincere thanks to all my friends and family for their
whole hearted support and encouragement.

Tony John

INDEX

INTRODUCTION

DEFINITION

In quantum mechanics, 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 model is mainly used as a hypothetical example to illustrate the differences between classical and quantum systems. In classical systems, for example, a particle trapped inside a large box can move at any speed within the box and it is no more likely to be found at one position than another. However, when the well becomes very narrow (on the scale of a few nanometers), quantum effects become important. The particle may only occupy certain positive energy levels. Likewise, it can never have zero energy, meaning that the particle can never "sit still". Additionally, it is more likely to be found at certain positions than at others, depending on its energy level. The particle may never be detected at certain positions, known as spatial nodes.

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.

DIAGRAM

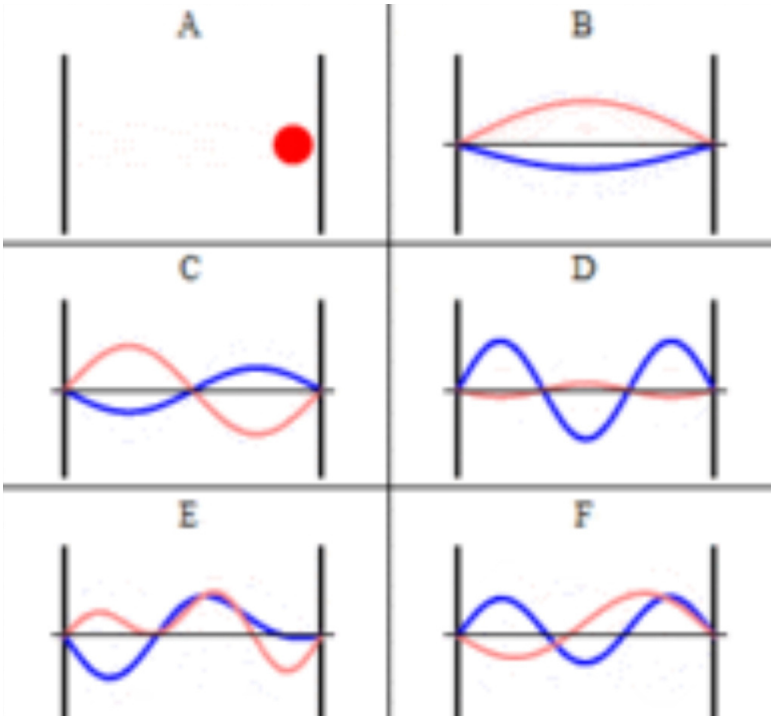

Fig 1

Figure 1

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.

PYTHON

DEFINITION

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy:

a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective. of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Benefit of python Programs

1. Easy to Read, Learn and Write

Python is a high-level programming language that has English-like syntax. This makes it easier to read and understand the code.

Python is really easy to pick up and learn, that is why a lot of people recommend Python to beginners. You need less lines of code to perform the same task as compared to other major languages like C/C++ and Java.

2. Improved Productivity

Python is a very productive language. Due to the simplicity of Python, developers can focus on solving the problem. They don't need to spend too much time in understanding the syntax or behavior of the programming language. You write less code and get more things done.

3. Interpreted Language

Python is an interpreted language which means that Python directly executes the code line by line. In case of any error, it stops further execution and reports back the error which has occurred.

Python shows only one error even if the program has multiple errors. This makes debugging easier.

4. Dynamically Typed

Python doesn’t know the type of variable until we run the code. It automatically assigns the data type during execution. The programmer doesn’t need to worry about declaring variables and their data types.

5. Free and Open-Source

Python comes under the OSI approved open-source license. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behavior and use their version for development.

6. Vast Libraries Support

The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don’t have to depend on external libraries.

But even if you do, a Python package manager (pip) makes things easier to import other great packages from the Python package index (PyPi). It consists of over 200,000 packages.

7. Portability

In many languages like C/C++, you need to change your code to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere.

However, you should be careful not to include any system-dependent features.

Particle in a Box Experimental Model Using Python

MODEL

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
$\omega = (k/2\mu)^{(1/2)}$
$\mu = (m1 + m2) / (m1m2)$
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.

$$V(x) = ½ kx^2$$

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 -relativistice

Schrödinger time dependent equation general form for particle in a box

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

which are solutions for Schrodinger time dependent equation.

This function mainly resembles the quadratic function ($y = x^2$) but the function has an inflection point to the right of the vertex (minimum). This is due to the fact that there is an optimal range in which bonded atoms are stable (at the minimum). For example, if atoms are too close, then the nuclei and electrons of each atom will repel each other and destabilize the molecule. As they draw further away, the amount of energy in the molecule falls and the atoms become more stable. However, once they are too far, the energies begin to increase again because unpaired electrons (in unbounded atoms) have a spin and angular momentum, which gives them intrinsic energy. The inflection point results from the two atoms being too far to interact anymore, and the energies of the two atoms stop changing and become constant.

Python based Analysis

We'll use Spyder (figure 2) for analysis of particle in a box model.

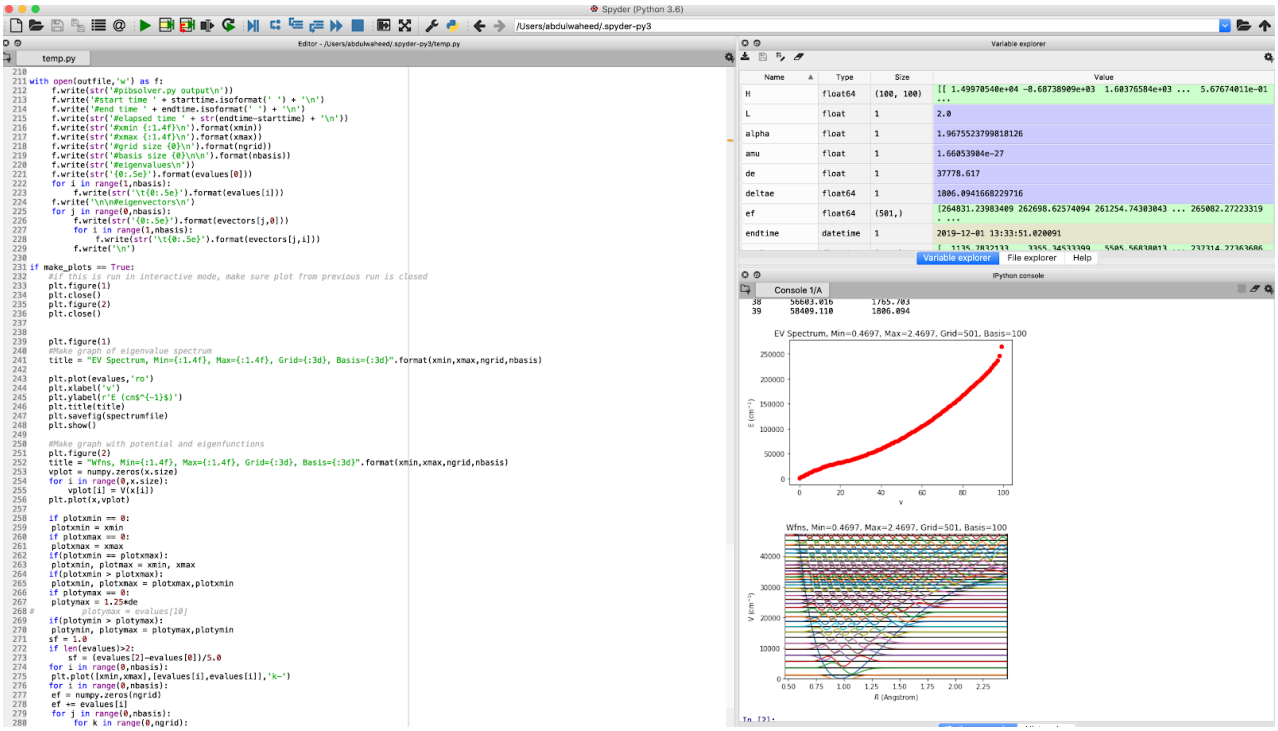

Fig 2

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 4). 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 3) is a function of atomic radius between the nuclei of bonded atoms. This function is the sum of all the individual harmonic functions.

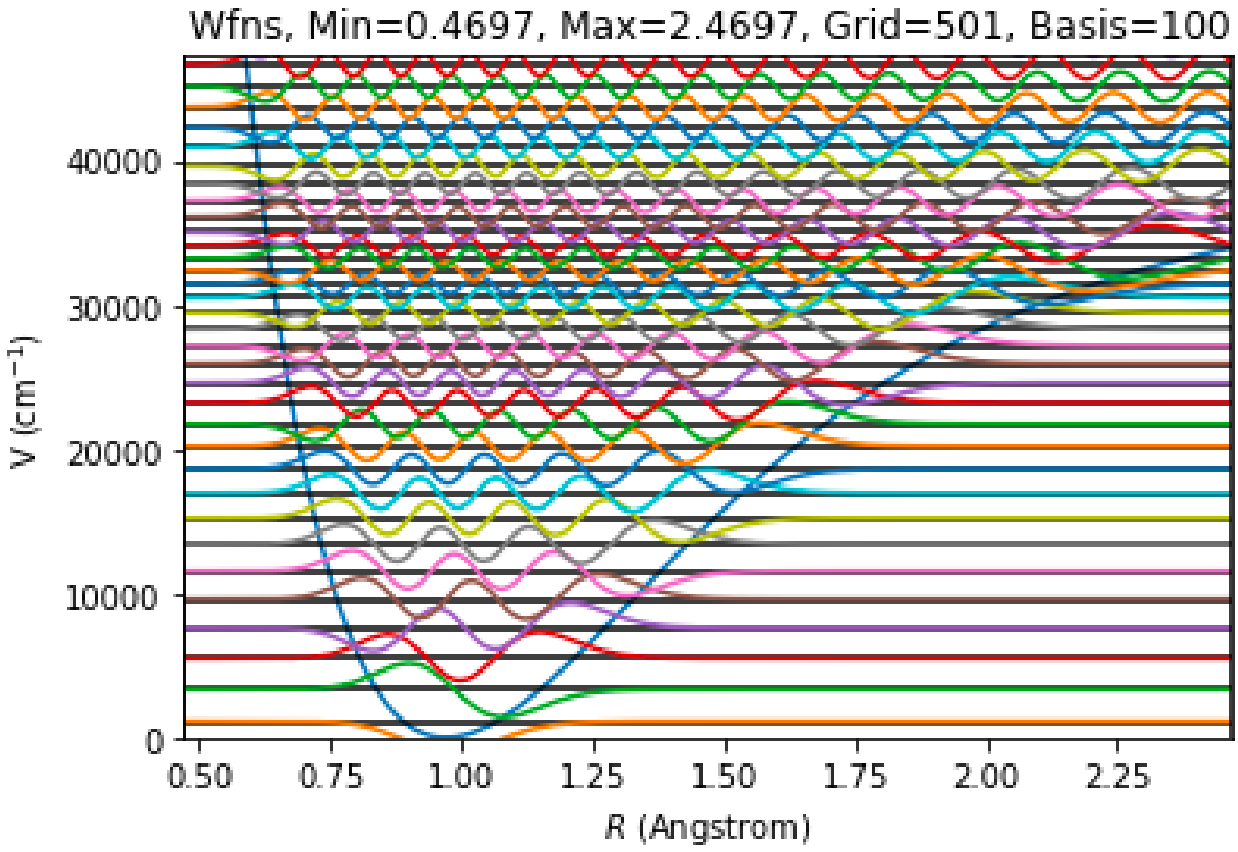

Fig 3

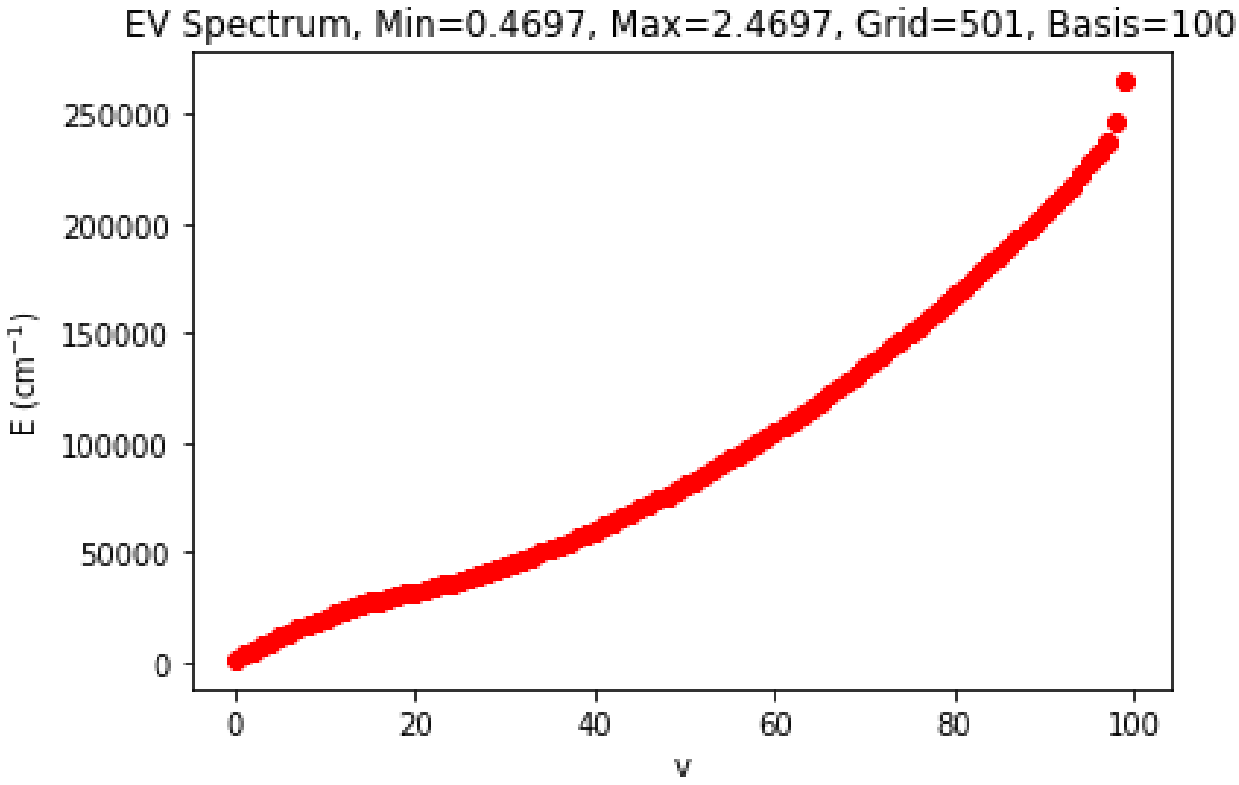

Fig 4

Overall, as energy levels increase, so does the energy transitions as shown in this graph (figure 4).

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 enery 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, evectors = 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(evectors[j,0]))
                for i in range(1,nbasis):
                        f.write(str('\t{0:.5e}').format(evectors[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] += evectors[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

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

CONCLUSION

Recently, I have been using python to model particle in a box. This model is used to approximate the energy of an electron as it transitions through various quantum states.I provided a simple description of particle in a box model and provided an easier python code to analyze and visualize it.

REFERENCE

- https://en.wikipedia.org/wiki/Particle_in_a_box
- https://estudogeral.sib.uc.pt/bitstream/10316/12349/1/Relativistic%20particle%20in%20a%20box.pdf
- https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Book%3A_Quantum_States_of_Atoms_and_Molecules_(Zielinksi_et_al)/04%3A_Electronic_Spectroscopy_of_Cyanine_Dyes/4.03%3A_The_Particle-in-a-Box_Model
- https://www.google.com/url?sa=t&source=web&rct=j&url=https://nptel.ac.in/content/storage2/courses/122101001/downloads/lec-2.pdf&ved=2ahUKEwjqp6jCmeT1AhXl4HMBHWqOA8wQFnoECDEQAQ&usg=AOvVaw0vnbZIWSnf2ms9FqO8x1F2