PHY 312: Numerical Methods and Programming

Wavefunction of nucleons inside a **spherical nucleus**

Soumyajit Samal (19129)
Satyajeet Moharana (19118)
Nitin Kumar Tripathy (19086)
Sitakanta Behera (19123)

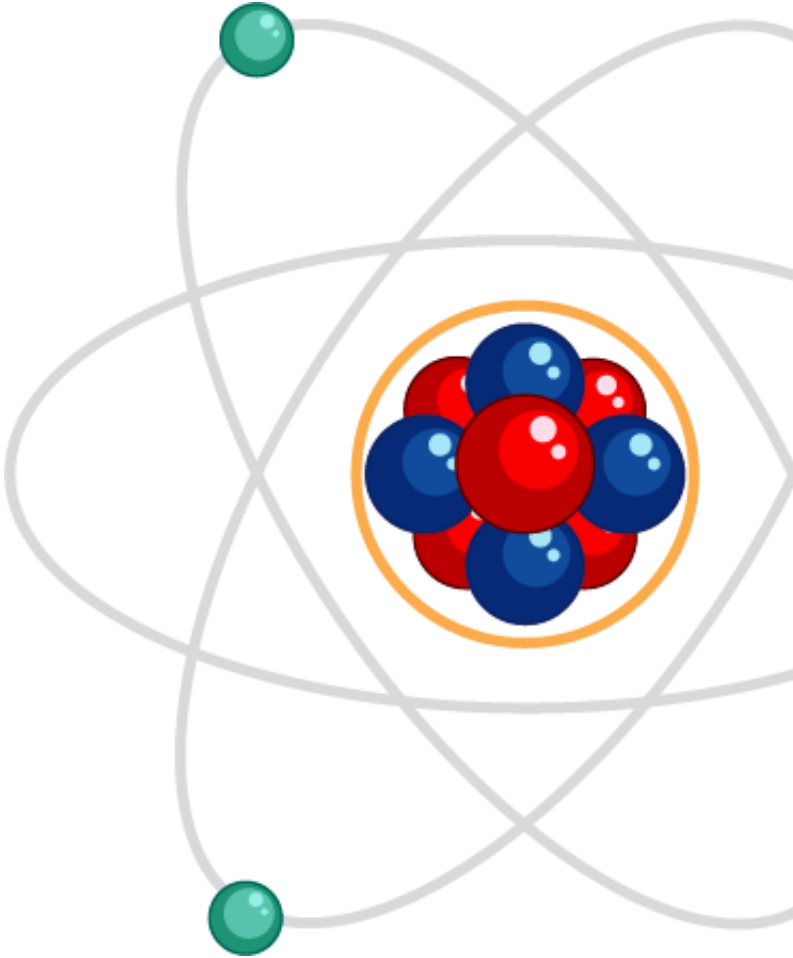
Submitted for the partial fulfilment of midsem sem 8 under the kind supervision of Prof. Achanta Venugopal

CONTENTS

i. Infinite cartesian well in 1D
ii. Infinite cartesian well in 3D
iii. Infinite spherical well in 3D
iv. Newton Raphson Method
v. Discussion of Code and Output

INTRODUCTION

The behaviour of nucleons in a nucleus differs from that of classical elements. Instead, the wave behaviour of the nucleons dictates the properties of the nucleus, and analysing this behaviour necessitates the use of mathematical methods of quantum mechanics. Here we have tried to write the solution of the schrodinger equation for a spherical nuclear potential using various numerical methods that we have learned.

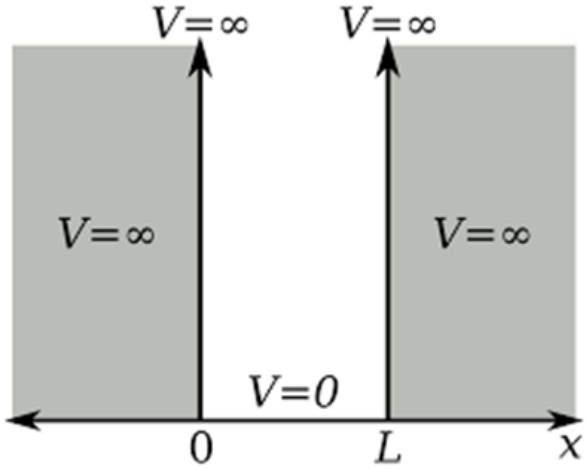


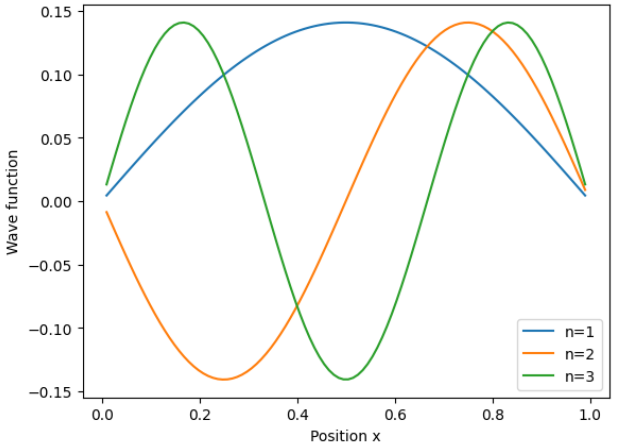
THE INFINITE CARTESIAN WELL

An infinite potential well is a model in quantum mechanics that describes a particle free to move in a small space surrounded by impenetrable barriers. It is also known as the particle in a box model or the infinite square well. The model is mainly used as a hypothetical example to illustrate the differences between classical and quantum systems. For the infinite well potential, the energy levels are proportional to n^2.

$$\frac{d^2\psi(x)}{dx^2} + k^2\psi(x) = 0,$$

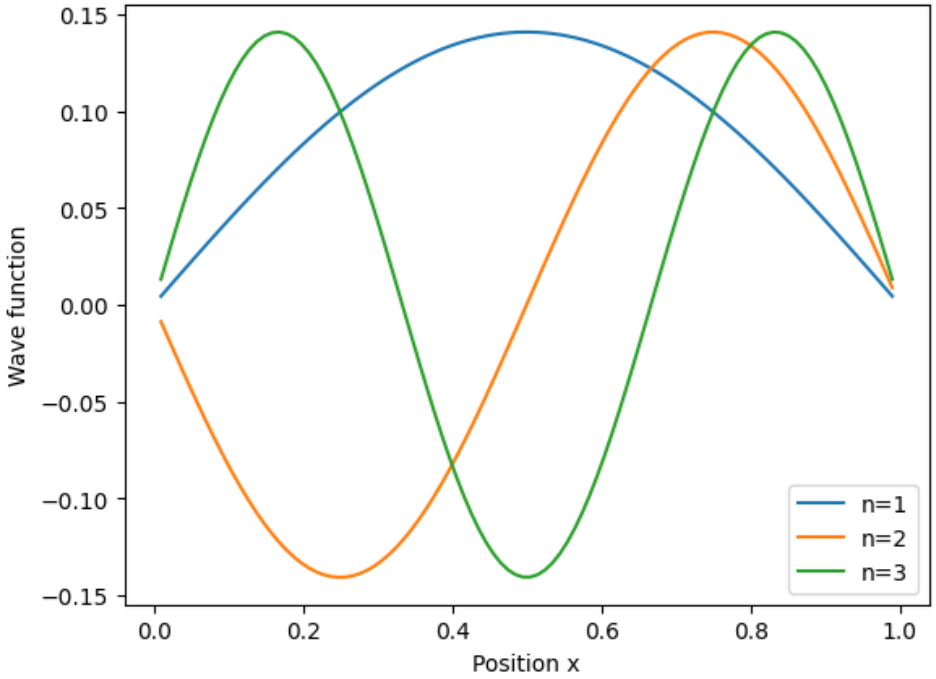
$$E_n = \frac{\hbar^2}{2m}k_n^2 = \frac{\hbar^2\pi^2}{2ma^2}n^2, \qquad (n = 1, 2, 3, \ldots).$$





THE CODE

```
import numpy as np
import matplotlib.pyplot as plt

# Define parameters
L = 1.0  # length of box
m = 1.0  # mass of particle
N = 100  # number of grid points
hbar = 1.0  # Planck constant

# Define grid
dx = L/(N+1)
x = np.linspace(dx, L-dx, N)

# Construct Hamiltonian matrix
H = np.zeros((N,N))
for i in range(N):
    H[i,i] = hbar**2/(2*m*dx**2) + 0.0  # add potential energy term here
    if i > 0:
        H[i,i-1] = -hbar**2/(2*m*dx**2)
    if i < N-1:
        H[i,i+1] = -hbar**2/(2*m*dx**2)

# Solve eigenvalue problem
E, psi = np.linalg.eigh(H)

# Plot wave functions
for i in range(3):
    plt.plot(x, psi[:,i], label=f'n={i+1}')
plt.xlabel('Position x')
plt.ylabel('Wave function')
plt.legend()
plt.show()
```



Real World isn’t based on 1D elements.

INFINITE CARTESIAN WELL in 3D

Considering the nucleus an infinite cartesian well, the potential is defined as -

$$\begin{aligned} V(x, y, z) &= 0 \qquad 0 \le x \le a, \quad 0 \le y \le a, \quad 0 \le z \le a \\ &= \infty \qquad x < 0, \quad x > a, \quad y < 0, \quad y > a, \quad z < 0, \quad z > a \end{aligned} \tag{2.53}$$

The particle is thus confined to a cubical box of dimension a. Beyond the impenetrable walls of the well, ψ = 0 as before. Inside the well, the Schrodinger equation is,

$$-\frac{\hbar^2}{2m}\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} + \frac{\partial^2\psi}{\partial z^2}\right) = E\,\psi(x, y, z)$$

INFINITE CARTESIAN WELL in 3D (Contd.)

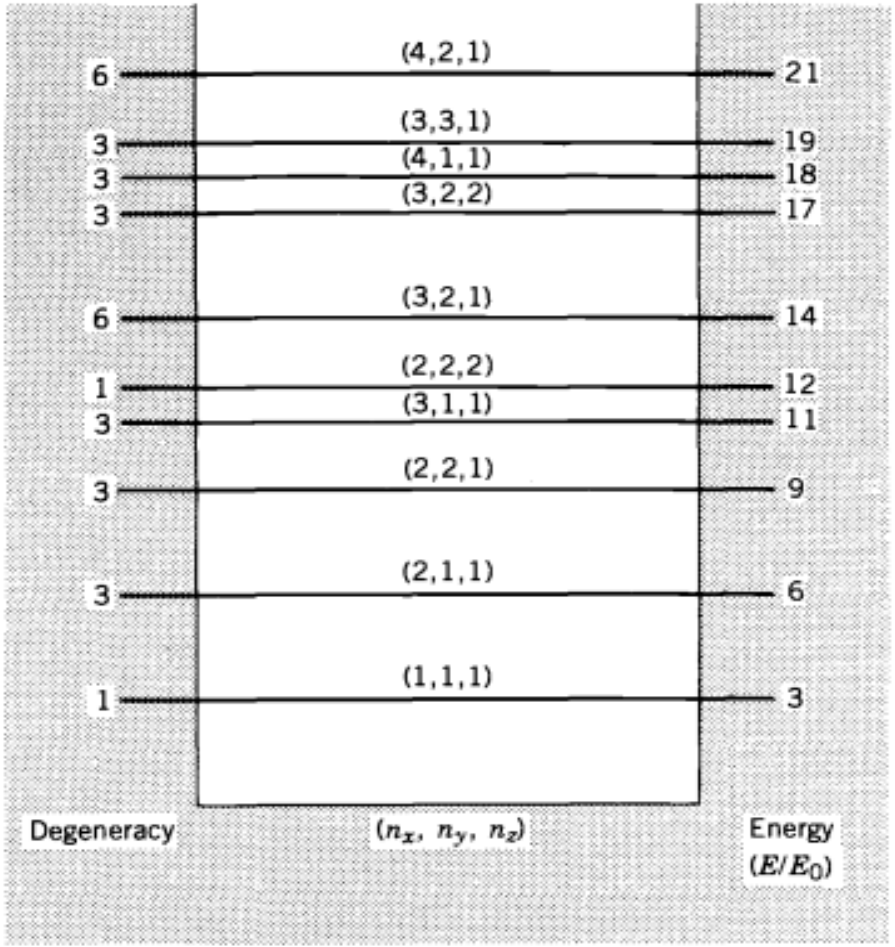
The usual procedure to solve these types of partial differential equations is to try to find a separable solution with ψ(x,y,z) = X(x).Y(y).Z(z). The result of this calculation gives the following solution.

$$\psi_{n_x n_y n_z}(x, y, z) = \sqrt{\left(\frac{2}{a}\right)^3} \sin\frac{n_x \pi x}{a} \sin\frac{n_y \pi y}{a} \sin\frac{n_z \pi z}{a}$$

$$E_{n_x n_y n_z} = \frac{\hbar^2 \pi^2}{2ma^2}\left(n_x^2 + n_y^2 + n_z^2\right)$$

Where n_x, n_y and n_z are independent integers greater than zero. The lowest state, the ground state has quantum numbers (n_x, n_y, n_z) = (1,1,1).

The first excited state has three possible states of quantum numbers: (1,1,2), (1,2,1) and (2,1,1). Each of these unique and autonomous states has a different wave function, which results in a different probability distribution and expectation values for the physical observables, but they all have the same energy. This is referred to as degeneracy, and the first excited state is threefold degenerate.



Energy levels of a particle confined to a three-dimensional cubical box. The energy is given in units of E = h2π2 / 2ma2.

INFINITE SPHERICAL WELL

If we work in spherical coordinates with a potential that depends only on r (not on Φ or θ) another new feature arises that will be important in our subsequent investigations of nuclear structure. When we search for separable solutions, of the form ψ(r,θ,Φ) = R(r)Θ(θ)Φ(φ), the central potential V(r) appears only in the radial part of the separation equation, and the angular parts can be solved directly. The differential equation for Φ(φ) is,

$$\frac{d^2\Phi}{d\phi^2} + m_\ell^2\,\Phi = 0$$

where m_ℓ^2 is the separation constant.

The solution is

$$\Phi_{m_\ell}(\phi) = \frac{1}{\sqrt{2\pi}}\, e^{im_\ell\phi}$$

where $m_\ell = 0, \pm 1, \pm 2, \ldots$. The equation for $\Theta(\theta)$ is

$$\frac{1}{\sin\theta}\frac{d}{d\theta}\left(\sin\theta\frac{d\Theta}{d\theta}\right) + \left[\ell(\ell+1) - \frac{m_\ell^2}{\sin^2\theta}\right]\Theta = 0$$

$$\Theta_{\ell m_\ell}(\theta) = \left[\frac{2\ell+1}{2}\frac{(\ell-m_\ell)!}{(\ell+m_\ell)!}\right]^{1/2} P_\ell^{m_\ell}(\theta)$$

where $P_\ell^{m_\ell}(\theta)$ is the associated Legendre polynomial

INFINITE SPHERICAL WELL

where $\ell = 0, 1, 2, 3, \ldots$ and $m_\ell = 0, \pm 1, \pm 2, \ldots, \pm \ell$. The solution $\Theta_{\ell m_\ell}(\theta)$ can be expressed as a polynomial of degree ℓ in $\sin\theta$ or $\cos\theta$. Together, and normalized, $\Phi_{m_\ell}(\phi)$ and $\Theta_{\ell m_\ell}(\theta)$ give the *spherical harmonics* $Y_{\ell m_\ell}(\theta, \phi)$, some examples of which are listed in Table 2.2. These functions give the angular part of the solution to the Schrödinger equation for *any central potential* $V(r)$. For example, it is these angular functions that give the spatial properties of atomic orbitals that are responsible for molecular bonds.

For each potential $V(r)$, all we need to do is to find a solution of the radial equation

$$-\frac{\hbar^2}{2m}\left(\frac{d^2R}{dr^2} + \frac{2}{r}\frac{dR}{dr}\right) + \left[V(r) + \frac{\ell(\ell+1)\hbar^2}{2mr^2}\right]R = E\,R \qquad (2.60)$$

Considering the case of infinite spherical well,

$$\begin{aligned} V(r) &= 0 \qquad r < a \\ &= \infty \qquad r > a \end{aligned}$$

BESSEL FUNCTION

The solutions of the differential equation of the form

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + \left(x^2 - \alpha^2\right) y = 0$$

Are collectively called Bessel Functions.

SPHERICAL BESSEL FUNCTION

The solution for V = 0 can be expressed in terms of oscillatory functions known as Spherical Bessel functions j(kr). To find the energy eigenvalues we proceed the same way as one dimensional problem and apply the continuity condition on ψ at r = a, which gives j(ka) = 0.

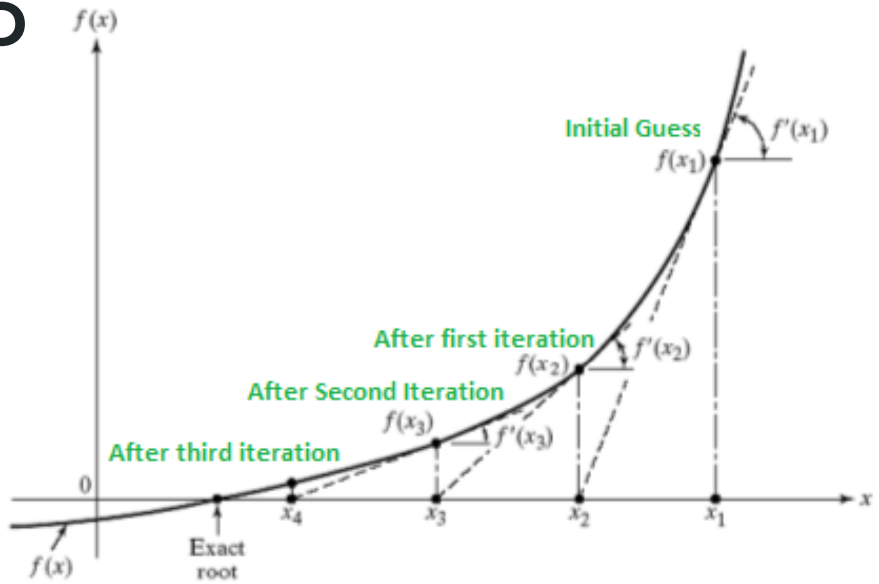
$$j_0(kr) = \frac{\sin kr}{kr}$$

$$j_1(kr) = \frac{\sin kr}{(kr)^2} - \frac{\cos kr}{kr}$$

$$j_2(kr) = \frac{3\sin kr}{(kr)^3} - \frac{3\cos kr}{(kr)^2} - \frac{\sin kr}{kr}$$

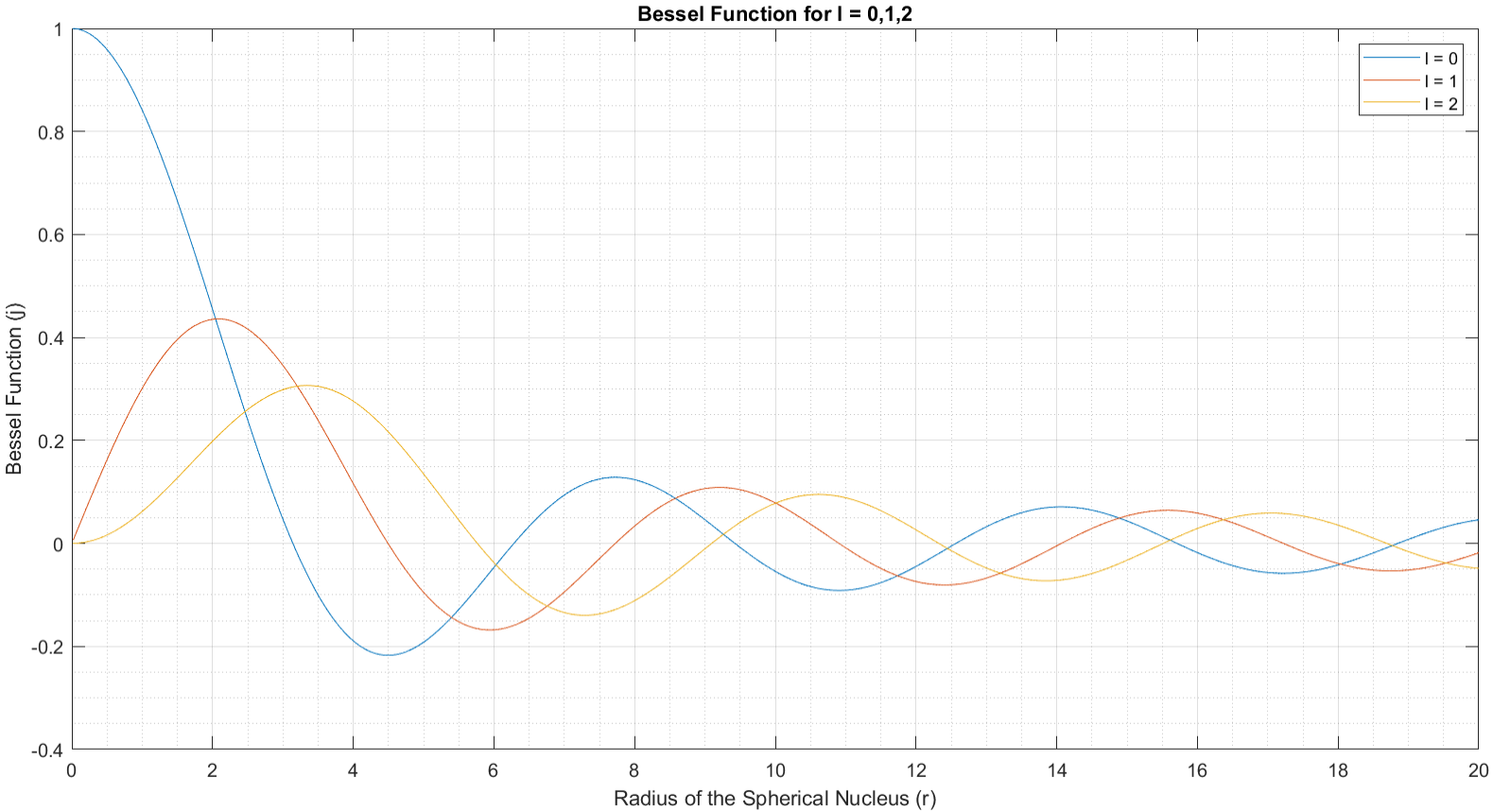
NEWTON RHAPSON METHOD

The Newton-Raphson method is an iterative root-finding algorithm that uses the derivative of a function to approximate its roots. The method starts with an initial guess for the root and then iteratively refines this guess using information from the function and its derivative.
At each iteration, the method uses the current guess x and the value of the function f(x) and its derivative f'(x) at that point to calculate a new guess x_new using the formula:



$$\text{Slope} = \frac{\Delta y}{\Delta x} = \frac{f(x_1)}{x_1 - x_2} = f'(x_1)$$

$$\Rightarrow\ x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$


Bessel Function for l = 0,1,2
Bessel Function (j)
Radius of the Spherical Nucleus (r)
l = 0
l = 1
l = 2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
2
4
6
8
10
12
14
16
18
20

DISADVANTAGES OF NEWTON RAPHSON METHOD

1. **Requires derivative information:** The Newton-Raphson method requires knowledge of the derivative of the function at each iteration. If the derivative is difficult or expensive to calculate, or if it is not available analytically, then the method may be impractical to use.
2. **Sensitive to initial guess:** The convergence of the Newton-Raphson method depends heavily on the choice of initial guess. If the initial guess is not close enough to the root, then the method may fail to converge or may converge to a different root than intended.
3. **Not guaranteed to converge:** Unlike some other root-finding algorithms, such as the bisection method, the Newton-Raphson method is not guaranteed to converge to a root. In some cases, the method may diverge or oscillate indefinitely without finding a solution.
4. **May converge slowly for certain functions:** For some functions, such as those with multiple roots or with roots that are close together, the Newton-Raphson method may converge slowly or may require many iterations to find a solution.

About the Code

The entire code has been written in MATLAB Ver 2023a.

Newton Raphson method is used to calculate the roots.

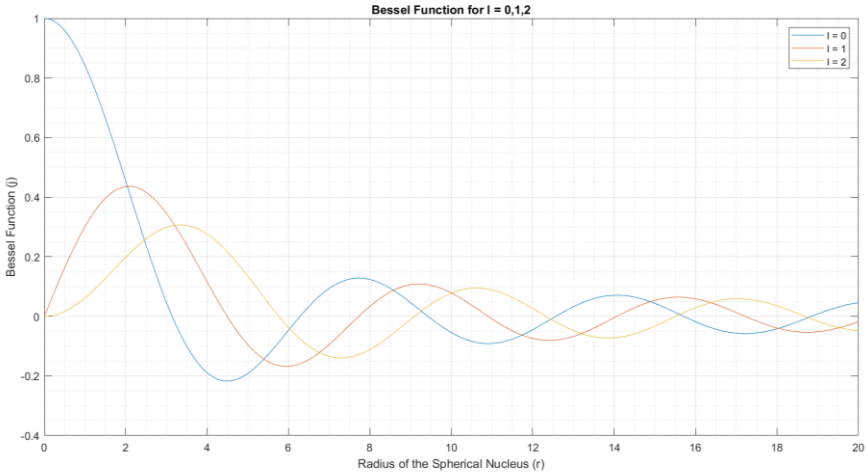
Intel i3 11th gen processor with 12GB RAM was used to run the codes.

CODE SNIPPETS

```
l=0;
% Define the function
f = @(r) sin(r)./r;

% Set the range of r values
r = linspace(0, 20, 1000);

% Calculate the Bessel function values
j = f(r);

% Plot the results
plot(r, j)
xlabel('Radius of the Spherical Nucleus
(r)')
ylabel('Bessel Function (j)')
title('Bessel Function for l = 0,1,2')
hold on
```

The code plots the Bessel functions. The given code is for only one function. For other 2 Bessel function the f value is replaced with other functions at the same time changing the value of l.

OUTPUT



CODE SNIPPETS

```
%using newton raphson method
%finding the solution for l=0
% Define the function and its derivative
f = @(r) sin(r)./r;
df = @(r) cos(r)./r - sin(r)./(r.^2);
% Set the tolerance
tol = 1e-6;
% Set the initial guesses
x0 = [3, 6, 9, 12, 15, 18];
% Initialize variables for the iteration
roots = zeros(size(x0));
errors = cell(size(x0));
% Perform the Newton-Raphson iteration for
each initial guess
for i = 1:length(x0)
x = x0(i);
error = inf;
errors{i} = [];
while error > tol
x_new = x - f(x)/df(x);
error = abs(x_new - x);
errors{i}(end+1) = error;
x = x_new;
end
```

```
roots(i) = x;
end
% Display the results
disp(['The roots of the function for l=0 are
approximately: ', num2str(roots)])
for i = 1:length(errors)
disp(['Errors for root ', num2str(i), ': ',
num2str(errors{i})])
end
```

OUTPUT

```
The roots of the function for l=0 are approximately: 3.14159
6.28319   9.42478   12.5664   15.708  18.8496

Errors for root 1: 0.13608   0.0055024    9.6155e-06    2.943e-11
Errors for root 2: 0.27754    0.0056353    5.0035e-06    3.9844e-12
Errors for root 3: 0.43067    0.0058971    3.7534e-06    1.4939e-12
Errors for root 4: 0.60386    0.037621    0.00012954    1.3346e-09
Errors for root 5: 0.80978    0.10284    0.0010178    6.5612e-08
Errors for root 6: 1.0697    0.22645    0.0062834    2.0132e-06
2.1316e-13
```

DESCRIPTION

Here we are getting 4 values for each root because it takes the algorithm 4 iterations to reach to the final result. The errors are noted at each step.

COMPUTATIONAL LIMITATIONS

- We could not implement for more assumed values of l=1 and l=2.
- Tolerance for l=2 is taken as 1 which is highly unreliable.
- The angular parts could not be simulated.

During the declaration of the project topic we intended to work on these topics as well but due to high computational demands we could not produce the results. All code was run on a personal computer.


Thank you!