PHYC40970- Advanced Laboratory II

The Deuteron

Nathan Power

19311361

Abstract

Here in this report, the properties of the deuteron are examined using graphical and numerical methods. The two particle time-independent Schrödinger equation for the deuteron, using the potential together with the application of the appropriate boundary conditions, leads to a transcendental equation that cannot be solved analytically. One aim of this project is to investigate the solution of this transcendental equation iteratively. We also investigate the bound state of the deuteron and the wave function of the deuteron. [1]

Introduction

A deuteron consists of a neutron and a proton. It is the simplest bound state of nucleons and therefore gives us an ideal system for studying the nucleon-nucleon interaction. There are no excited states of the deuteron, it is such a weakly bound system that the only “excited state” is the unbound system consisting of a free proton and a free neutron. The deuteron is the only stable bound system of two nucleons. Experimentally it is known that the deuteron binding energy is 2.2 MeV. The deuteron is therefore a very weak bound system when compared to a typical nucleon. In this study, we will solve for deuteron potential depth as one of its properties necessary to be known in order to understand the nucleon-nucleon force. [2]

The deuteron, composed of a proton and a neutron, is a stable particle. The stability of the deuteron is an important part of the story of the universe as in the Big Bang model it is presumed that in early stages there were equal numbers of neutrons and protons since the available energies were much higher than the energy required to convert a proton and electron to a neutron. When the temperature dropped to the point where neutrons could no longer be produced from protons, the decay of free neutrons began to diminish their population. Those which combined with protons to form deuterons were protected from further decay. This is fortunate for us because if all the neutrons had decayed, there would be no universe at all [3]
The nuclear force is an important force for the existence of life on earth. This force plays an important role in giving various properties to the nuclei. Understanding the nuclear force and its behaviour within the nuclear dimensions is important not only to explain the various observed properties of nuclei but also to predict many other properties in nuclei which are so far inaccessible. To study the properties of nuclear force, deuteron provides an ideal case as it is a loosely bound system, its binding energy is 2.2 MeV, which is much less than what is required for a strongly bound two-nucleon system.[4]

Theory

To simplify the analysis of the deuteron, we will assume that we can represent the nucleon-nucleon potential as a one-dimensional square well, as shown in Fig.1

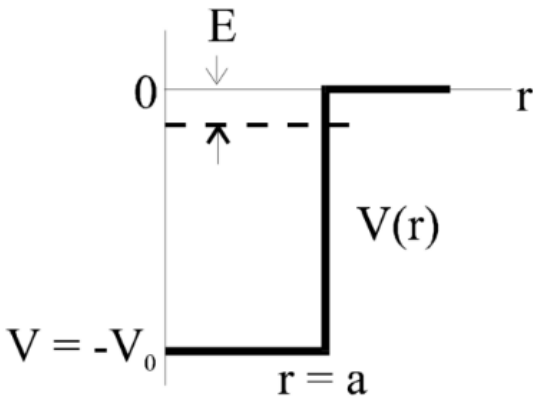


Fig.1 Square well potential representing the nucleon-nucleon potential of the deuteron [1]

$$V(r) = -V_0 \quad from\ r < a$$
$$V(r) = 0 \quad from\ r > a$$

This is of course an oversimplification, but it is sufficient for at least some qualitative conclusions. Here r represents the separation between the proton and the neutron, so R is in effect a measure of the diameter of the deuteron, the problem is also simplified by taking an imaginary single reduced mass encircling the centre of gravity of the deuteron. The reduced mass in the simplified problem is half the mass of the proton or neutron. This ignores the tiny difference in mass between the proton and neutron. If we are interested in bound states which are energy eigenstates that are normalizable. This energy, E, of the states must be negative. If E > 0, any solutions in the region r > a where the potential vanishes would be a plane wave, extending all the way to infinity. Such a solution would not be normalizable. The energy E is defined as being;[5]

$$-V_0 < E < 0$$

We have to examine how the equation looks in the various regions where the potential is constant and then use boundary conditions to match the solutions across the points where the potential is discontinuous. Let's examine the regions, where, for simplicity, we define the time-invariant, non-relativistic Schrödinger's equation by;[5]

$$\frac{-\hbar}{2m}\frac{d^2u}{dr^2} + V(r)\,u(r) = Eu(r)$$

Where $\hbar = 1.0545E-34\,J.s$ [8]

m $= 1.6E-27\,kg$

Which can be rearranged to give;

$$\frac{d^2\Psi}{dx^2} = -\frac{2m}{\hbar^2}(E - V(x))\Psi$$

Since the potential is piecewise continuous we must study the differential equation in the two regions:

Region of x < a :

$$\frac{d^2\Psi}{dx^2} = -\frac{2m}{\hbar^2}(E - (-V_0))\Psi = -\frac{2m}{\hbar^2}(V_0 - E))\Psi$$

$(V_0 - E))$ is a positive constant thus define a real k > 0 by;

$$k_1^2 = \frac{2m}{\hbar^2}(V_0 - E))$$

The differential equation to be solved now reads;

$$\frac{d^2\Psi}{dx^2} = -k_1^2\Psi$$

and the solutions can be written;[5]

$$u(r) = Asin(k_1 r) + Bcos(k_1 r)$$

Region of x > a :

$$\frac{d^2\Psi}{dx^2} = -\frac{2m}{\hbar^2}(E - (0))\Psi = \frac{2m}{\hbar^2}|E|\Psi$$

$$k_2^2 = \frac{2m}{\hbar^2}|E|$$

The differential equation to be solved for this region now reads;

$$\frac{d^2\Psi}{dx^2} = k_2^2\Psi$$

Which solutions are exponentials; $u(r) = Ce^{k_2 r} + De^{-k_2 r}$

To keep the wave function finite for r →∞ we must have C = 0, and to keep it finite for r → 0 we must have B = 0. Applying the continuity conditions on u and $\frac{d\,u}{dr}$ at r = R, we obtain;

$$k_1 r\, cot(k_1 r) = -k_2 r$$

This transcendental equation gives a relationship between V_0 and R and this report aims to examine the solutions to this equation iteratively. [5]

The first task within this report was to write a routine to give an initial estimate for the roots of the transcendental equation using graphical methods. This could be achieved cleanly through the use of unit-free constants. This way we would not have to worry about units and conversions in order to find our roots.[6] By examining our transcendental equation we could extract three new unit-free constants;

$$x = k_1 r$$
$$y = k_2 r$$
$$z_0^2 = x^2 + y^2 = \frac{2m}{\hbar^2} r^2 V_0$$

Clearly x is a proxy for k_1 and y is a proxy for k_2. Both depend on the energy of the bound state. The parameter z_0^2, unit-free, just depends on the data associated with the potential, the depth V_0 ,the width r and the mass m of the particle. A very deep or wide potential has very large z_0^2, while a very shallow or narrow potential has small z_0^2. As we will see the value of z_0^2 tells us how many bound states the square well has. These unit-free parameters will be very useful when finding the roots of our equation as we can rearrange it as follows;

$$-xcot(x) - y = 0$$

Or

$$-cot(x) = \frac{k_2 r}{x}$$

Plotting these functions reveal possible values for x and y which in turn can be used to find values for V_0 needed in the next part of the report.

The next section of the report is was required to write an implementation of the Newton-Raphson method to determine V_0 iteratively from an initial estimate. This can be done by using our root equation from the previous section and implementing it into our Newton Raphson function. The Newton-Raphson method is an iterative method used to approximate the roots or zeros of a function. The Newton-Raphson method is one of the most popular methods for calculating roots due to its simplicity and speed. The first step of the method takes an initial guess, x_1 and uses the function, $f(x)$, and function derivative, $g(x)$, to calculate a next guess. Then, this guess is used in a similar fashion to calculate the next guess, and so on, until a tolerance or iteration limit is met. The first step would be;

$$x_1 = x_0 - \frac{f(x_0)}{g(x_0)}$$

Followed then by;

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

Which repeats until the required tolerance is met.

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{g(x_{i-1})}$$

The initial guess is very important. Depending on the problem, if you start with a terrible guess, it can make your convergence take longer or not converge at all. However, with an accurate estimated initial guess, the Newton-Raphson algorithm will converge on a solution relatively quickly within a few iterations.[7] The initial guess is also important if you have a function with multiple roots, like our transcendental equation, so the stringency of our initial guess is of utmost importance. Depending on the initial guess, we can converge on any of these roots. These problems can be mitigated by using an educated initial guess.

Next, we investigate the possible bound states of the deuteron. How many bound states exist for the deuteron? The simplest way to determine the condition for the existence of the bound states is to examine our transcendental equation when , $-xcot(x) = y = 0$. When $y = 0$ we have $xcot(x) = 0$ and therefore $x = \frac{\pi}{2}, \frac{3\pi}{2}, \frac{5\pi}{2}$ .....
By using a circle of radius z_0 we can determine how many bound states are possible for various values of V_0 and R and therefore determine the allowed values of E.[6]

Having determined a value for V_0, and hence an idea of the strength of the nuclear force that is responsible for binding the deuteron, it is possible to determine the wave function for the deuteron.This enables properties of the system, such as the expectation value for the separation of the proton and the neutron and the probability that this separation will be greater than the dimension of the potential well, to be explored. [1]

Results and Discussion

Write a routine to give an initial estimate for the root(s) of the transcendental equation using graphical methods

The first task within this report was to write a routine to give an initial estimate for the root(s) of the transcendental equation using graphical methods. This could be achieved cleanly through the use of unit-free constants; x, y and z_0.

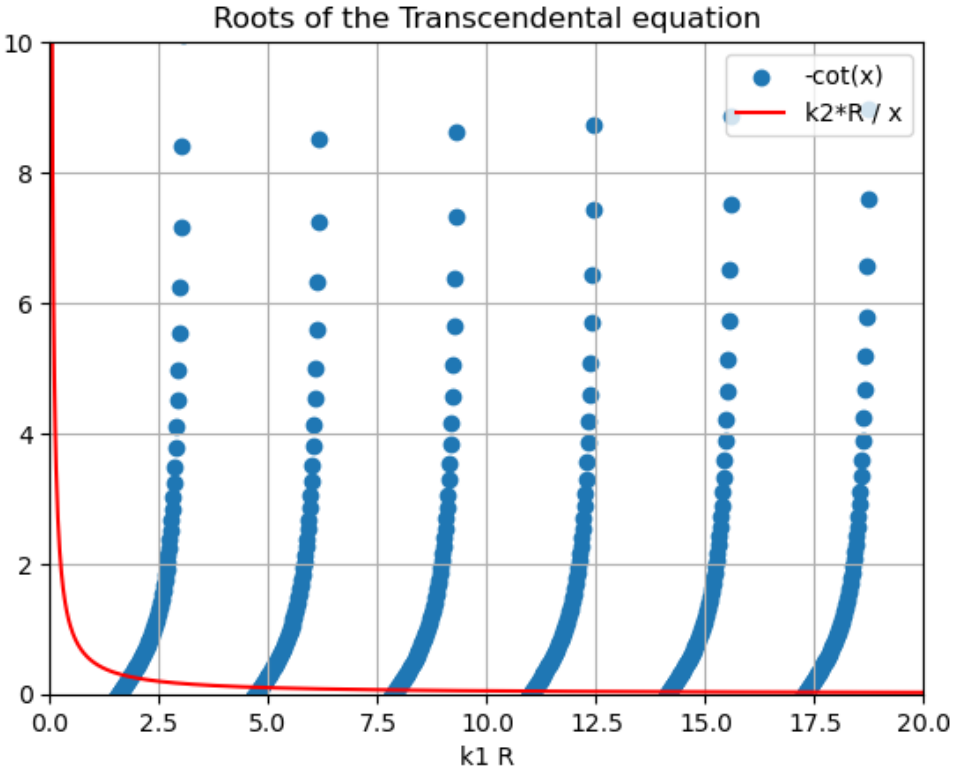


Fig.2 shows a plot of y = -cot x (blue) and y= $\frac{k_2 R}{x}$ *(red). Where the functions intersect are roots to our transcendental equation.*

The first method I employed to find roots of the transcendental equation was to plot both the left and right hand side of the equation on the same plot and to examine where they intersected. However, in the next part of the report I found that it would be beneficial to write the equation in the from $f(x) = 0$ in order to use it in my Newton Raphson formula shown in Fig.3 below.

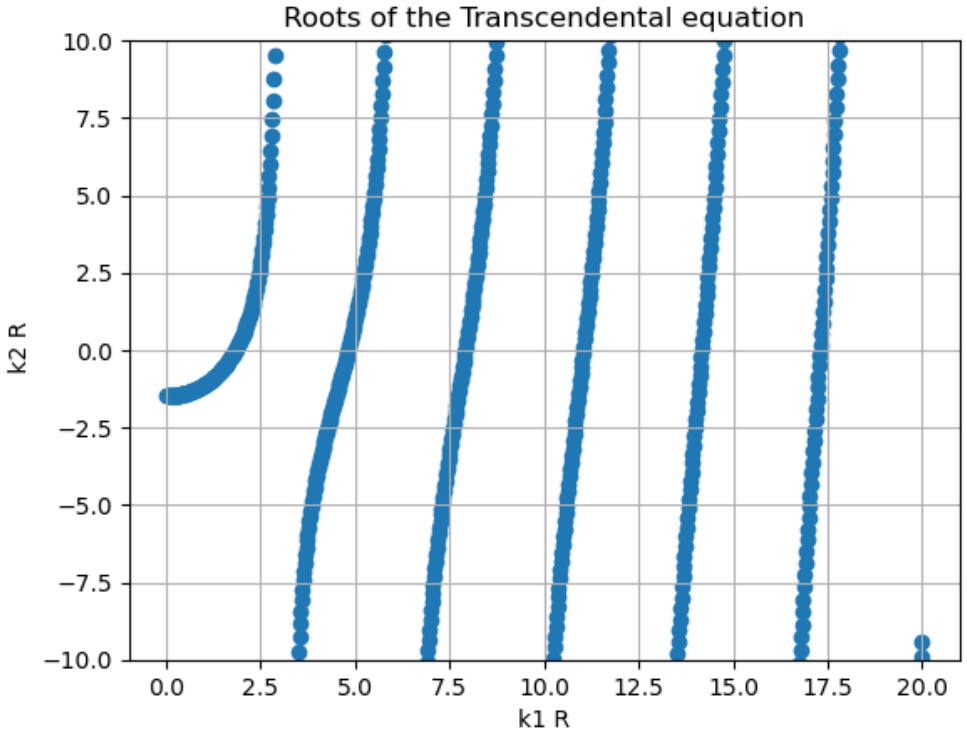


Fig.3 shows a plot of y = -xcot x - k_2R (blue) and where y=0 is a root for the transcendental equation

The first three roots found, using the bisection method along with the graphical methods, were $k_1R = 1.836,\ 4.813,\ 7.914$ which correspond to values of $V_0 = 33.84\ MeV,\ 220.83\ MeV,\ 375.48\ MeV$.

Using constants;

Where $\hbar = 1.0545E-34\ J.s$
m =$1.6E-27\ kg$
R =$2.1\ fm$
E = $2.23\ Mev$ [8]

Write an implementation of the Newton-Raphson method to determine V_0 iteratively from an initial estimate.

Next using the above transcendental formula in the form $f(x) = 0$, the Newton Rapshon method was deployed in order to get a more accurate value for V_0. Using an initial estimate of 1.9 for our root and a tolerance of 1E-15 the Newton Raphson found the first root at 1.8303, second root 4.813 and third root at 7.9153. Corresponding to values of V_0 of 34.036 MeV, 221.493 MeV and 593.275 MeV. As we can see, the values for the roots and the values of V_0 are in good agreement with one another for both the bisection method and the Newton Raphson method.

```
Enter Guess: 1.9
Tolerable Error: 1E-15
Maximum Step: 20
Run-1, x1 = 1.833836 and f(x1) = 0.007694
Run-2, x1 = 1.830395 and f(x1) = 0.000019
Run-3, x1 = 1.830387 and f(x1) = 0.000000
Run-4, x1 = 1.830387 and f(x1) = 0.000000

Required root is: 1.83038684
V0 is: 33.84577967 MeV
```

Fig.4 shows an example of the output of the Newton Raphson

We can see how our Newton method converges with each step it takes as f(x1) gets closer and closer to zero. Here in the example code above for our first root at 1.8303, we can see that our initial guess of 1.9 was not far off and within four iterations, the root was found. The initial guess is very important. If we had started with an initial guess that was far away from our root, the method may not have converged at all. However, with this accurate estimated initial guess, the Newton-Raphson algorithm converged on a solution relatively quickly within a few iterations.

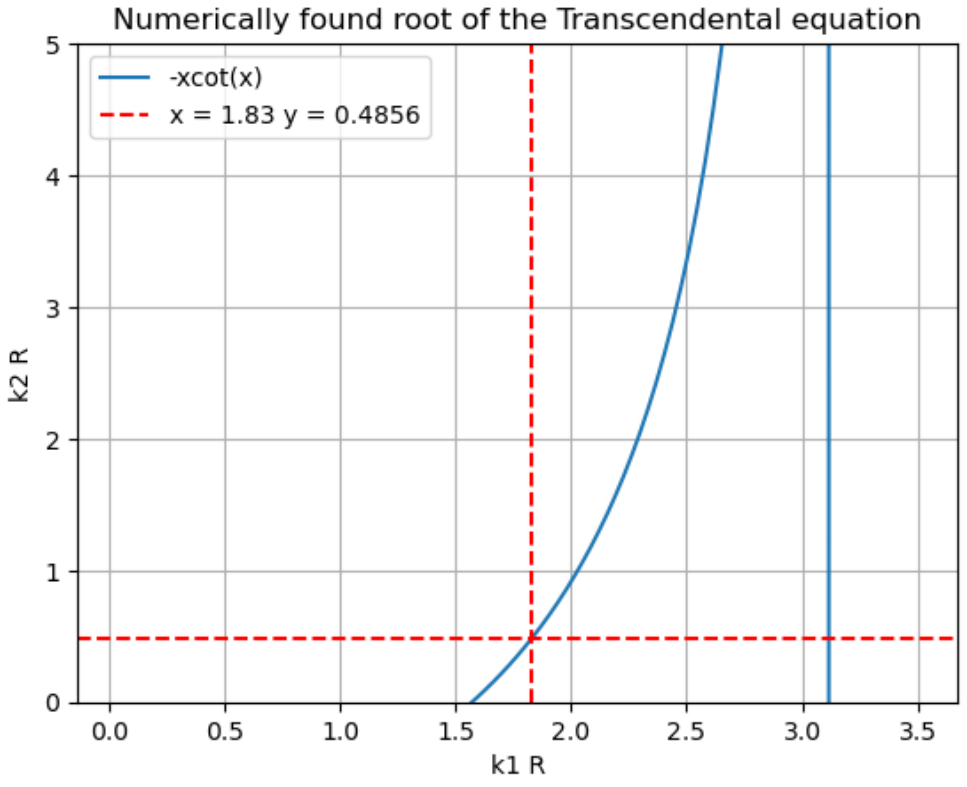


Fig.5 shows a plot of y = -xcot x (blue) and solved values for x = $k_1 R$ *and y =* $\frac{k_2 R}{x}$ *(red) using accepted values. Where the functions intersect are a solution which gives a value of* $V_0 = 33.828\ MeV$

The initial guess is also important if you have a function with multiple roots, like our transcendental equation, so the stringency of our initial guess is of utmost importance. Depending on the initial guess, we can converge on any of these roots that can be seen in Fig.3 .

Using accepted values for E and R [8] in and thus using these to work out values for x and y, shown in Fig.5, for the bound state of the deuteron, a value V_0 found was 33.828 MeV which is in agreement with the given value of 35 MeV.

How many bound states exist for the deuteron? Investigate how the values of V0 and a affect the allowed values of E.

The deuteron is the nucleus that contains 1 proton and 1 neutron. It is the only stable state for 2 nucleons. Experimentally we know that the deuteron has only one bound state. Deuteron has no excited state. It is because any excitation will easily make the system break apart[9].It is such a weakly bound system that the only “excited state” is the unbound system consisting of a free proton and a free neutron. Experimentally it is known that the deuteron binding energy is 2.225 MeV. The deuteron is therefore a very weak bound system when compared to a typical nucleon. Using our unit-free parameters we are able to find possible values for this binding energy depending on the values for V_0 and R (the separation of the proton and neutron)

$$x = k_1 r$$
$$y = k_2 r$$
$$z_0^2 = x^2 + y^2 = \frac{2m}{\hbar^2} r^2 V_0$$

By examining our z_0^2 we can see that this formula corresponds to a circle of radius z_0, which is dependant on our values for V_0 and R. By varying these and plotting against $y = -x\cot(x)$ we can find possible bound states and possible values for E.
The intersection of two curves is the solution of the values of x and y for the fixed value of z_0.
The results are as follows;

(i) $\frac{\pi}{2} > z_0 > \frac{3\pi}{2} = one\ bound\ state$
(ii) $\frac{3\pi}{2} > z_0 > \frac{5\pi}{2} = two\ bound\ states$
(iii) $\frac{5\pi}{2} > z_0 > \frac{7\pi}{2} = three\ bound\ states$

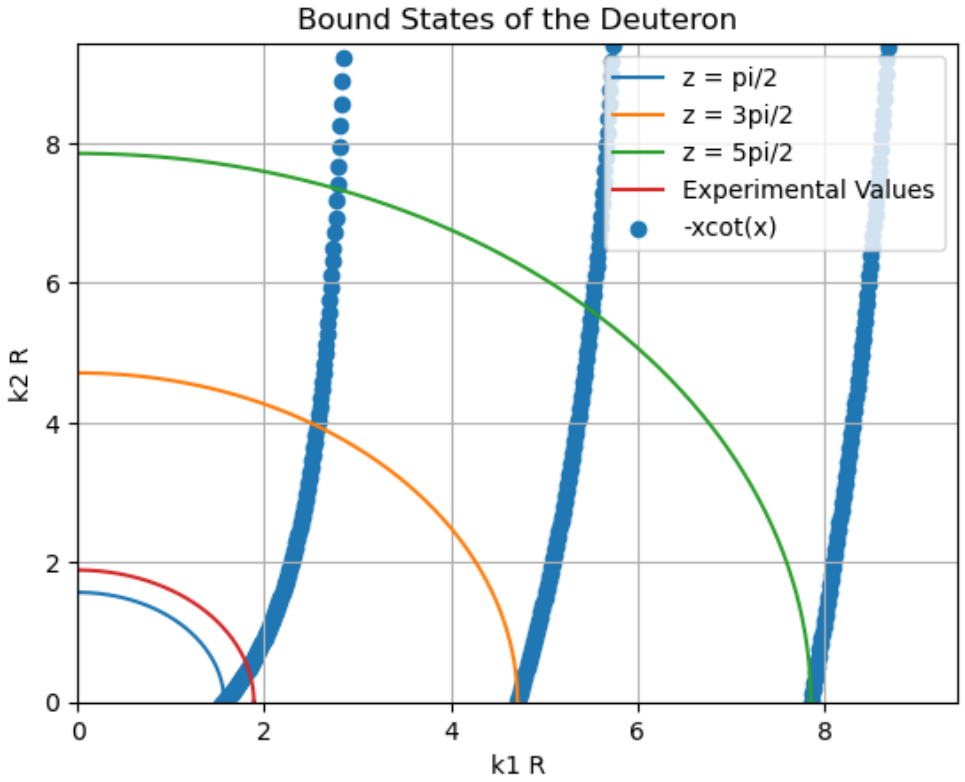


Fig.6 shows a plot of the possible bound states for the deuteron for various values of V_0 and R

As z_0 increases, the potential well becomes deeper and/or wider as V_0 and R both increase. Using our value for V_0 for the first found root and the accepted value for R = 2.1 fm we found that our z_0 value does fall between $\frac{\pi}{2} > z_0 > \frac{3\pi}{2}$ which agrees with what we would have expected.

In order to discuss the relation between z_0 and R, we use our formula $z_0$and V_0 = 33.83 MeV. The plot of z_0 vs R is shown below in Fig.7. When R = 2.1 fm, z_0 is between $\frac{\pi}{2}$ and π. This means that there is no bound state for l = 1, 2,....etc and that there is only one bound state for l = 0. [6]

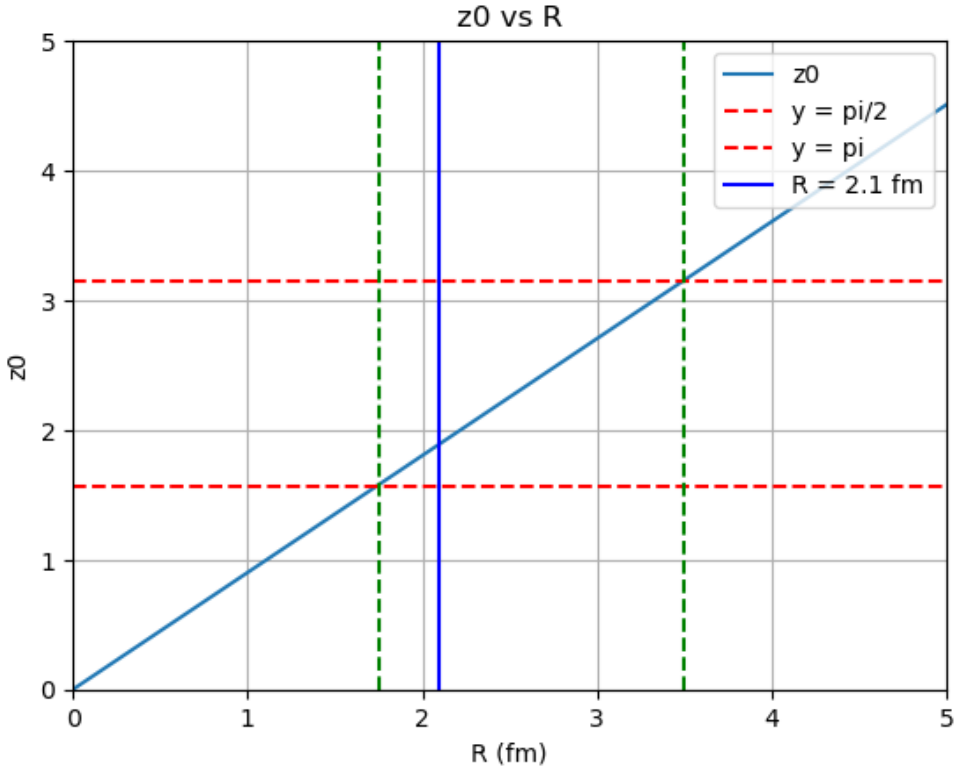


Fig.7 The relation between z_0 and R [fm]

Determine the wavefunction

Having determined a value for V_0, and hence an idea of the strength of the nuclear force that is responsible for binding the deuteron, it is possible to determine the wave function for the deuteron. This enables properties of the system, such as the expectation value for the separation of the proton and the neutron and the probability that this separation will be greater than the dimension of the potential well, to be explored.
We remember from before that;

$$u(r) = Ae^{-k_2 r} \; for \; r > R$$

$$u(r) = Bsin(k_1 r) \; for \; r < R$$

Here we make a plot of the wave function of the bound state for deuteron. Note that u is proportional to $sin(k_1 r) \; for \; r < R$. For $r > R$, the wave function is proportional to $e^{-k_2 R}$. Using constants determined from the normalisation of the wavefunction below.

$$A = \frac{e^{\alpha R}\sqrt{\kappa R\,\alpha R}\,\sin(\kappa R)}{\sqrt{\pi R}\sqrt{2\kappa R\,\alpha R + 2\kappa R\sin^2(\kappa R) - 2\alpha R\sin(2\kappa R)}}$$

$$B = \frac{\sqrt{\kappa R\,\alpha R}}{\sqrt{\pi R}\sqrt{2\kappa R\,\alpha R + 2\kappa R\sin^2(\kappa R) - 2\alpha R\sin(2\kappa R)}}$$

Fig.8 The normalisation constants for the wavefunction. Here $\kappa = k_1$ *,*$\alpha = k_2$*[6]*

The deuteron wave function for R = 2.1 fm can be seen below in Fig.9. Note how the exponential joins smoothly to the sinusoidal function at r = R, so that both $u(r)$ and $\frac{du}{dr}$ are continuous. If the wave function did not "turn over" inside r = R, it would not be possible to connect smoothly to a decaying exponential and there would be no bound state. [5] The function u has a peak at $\frac{r}{R} = \frac{\frac{\pi}{2}}{k_1 R} = \frac{\frac{\pi}{2}}{x} = 0.803$, where x = 1.83084.

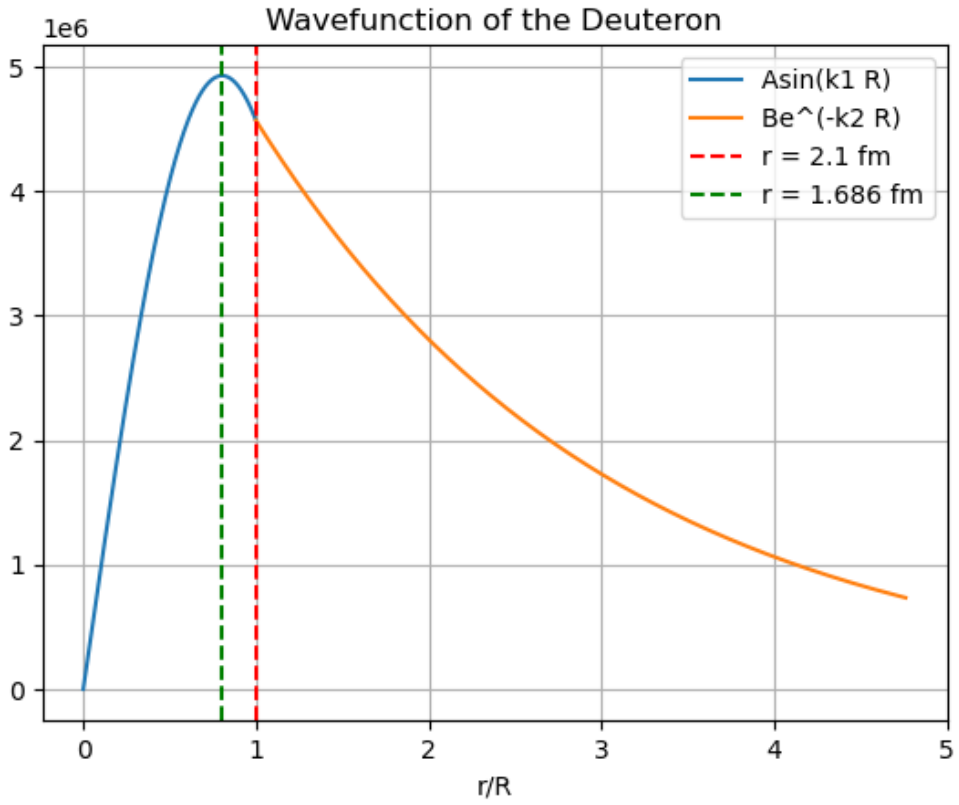


Fig.9 The plot of u(r) as a function of r/R. R = 2.1 fm, E = -2.23 MeV , V_0 *= 33.8 MeV*

We can see from Fig.9 , most of the time the wave function is outside the well. The weak binding means that wave function is just barely able to “turn over” in the well implying that the curve started to turn down so as to connect at r = R with the negative slope of the decaying exponential,[5] meaning the wave function extended beyond the nuclear interaction range and therefore an excited state was not expected. The wavefunction is sharply peaked at 0.803 corresponding to a radius of 1.686 fm, and the probability density, being its square,

is likewise peaked there as well. This is the wavefunction for a particle well localised at a position given by the peak, as the probability density is high there.[10] The expectation value for the separation of the proton and the neutron can be estimated to be this value as it is the probabilistic expected value of the separation r. It can be thought of as an average of all the possible outcomes of the radius as weighted by their likelihood calculated by;

$$\langle r \rangle = \int_0^\infty \psi^* r \psi \, dr \simeq 1.8 \ fm$$

The probability that this separation will be greater than the dimension of the potential well can be calculated as the following;

$$P = \int_R^\infty \psi^* \psi \, 4\pi r^2 \, dr \qquad R = 2.1 \, fm$$

$$= 4\pi C^2 \int_R^\infty e^{-2k_2 r} dr = 2\pi C^2 e^{-2k_2 r}$$

$$A sin(k_1 r) = C e^{-k_2 r} \ or \ C^2 e^{-2k_2 r} = A^2 sin^2(k_1 r) \simeq A^2$$

$$P = \frac{2\pi A^2}{k_2} \simeq 0.6586$$

Thus neutron and proton stay outside the range of nuclear forces approximately 70% of time which agrees with our plot of $u(r)$.

Conclusion

In this report, we examined the transcendental equation to find its solutions which gave us possible bound states for the deuteron. We also examined the wave function of the deuteron, expectation value of the separation of the proton and neutron and the probability this separation will be greater than the dimension of the potential well. When compared to previously found results our values agreed. We calculated the deuteron potential depth V_0, and a reasonable solution to the square potential well problem was obtained. Compared with the other solutions from other researchers, the obtained result from this study is among those values close to the typical potential depth value of 35 MeV[2]. The deuteron potential value that was found in this study shows the reliability of Python programming language and the accuracy of the Newton Raphson algorithm in their ability to produce accurate results with simplistic methods.

References

[1] The Deuteron Lab Computational Lab Manual, UCD School of Physics 2023
[2] Numerical solution of deuteron potential depth using python, Adam Muhammad and I.A. Alnour 2022 IOP Conf. Ser.: Mater. Sci. Eng. 1231 012005
[3]Gsu.edu. (2019). The Deuteron. [online] Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/Particles/deuteron.html
[4] Gupta, V. (n.d.). Physics Nuclear and Particle Physics Nuclear Force and its Properties [online]
[5] Krane, K.S. (1988). Introductory nuclear physics. New Delhi: Wiley, Cop. Ch.4
[6] Masatsugu, D. and Suzuki, S. (n.d.). Available at: https://bingweb.binghamton.edu/~suzuki/QM_Graduate/Deuteron.pdf [Accessed 23 Feb. 2023]
[7]Fizell, Z. (2022). Develop Your Own Newton-Raphson Algorithm in Python. [online] Medium. Available at: https://towardsdatascience.com/develop-your-own-newton-raphson-algorithm-in-python-a20a5b68c7dd [Accessed 23 Feb. 2023]
[8] Elert, G. (2019). Glenn Elert. [online] The Physics Hypertextbook. Available at: https://physics.info/constants/
[9]Nuclear Physics 101. (2011). Deuteron. [online] Available at: https://nukephysik101.wordpress.com/2011/02/08/deuteron/
[10] Adams, A. (2013). Lecture 3. [online] Available at: https://ocw.mit.edu/courses/8-04-quantum-physics-i-spring-2013/19862abcdf8e4f7cf1321d060cd6dd7c_MIT8_04S13_Lec03.pdf [Accessed 23 Feb. 2023].

The Deuteron

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Constants and Formulas

In [144…
```
E= 2.23 #binding energy
R = 2.1E-15 #radius/seperation of proton and neutron in metres
V_0= 33.84577967 #mev #potetnial well depth
h=1.0545E-34 #hbar in J.s
m = 1.67E-27 /2 #reduced mass in Mev = proton mass * neutron mass  / proton mass +
k1 = np.sqrt((2 * m *(E - V_0)* 1.6E-13) / (h**2))  #1.6E-13 to convert J to MeV
k2 = np.sqrt(((2 * m * E)* 1.6E-13 / (h**2)))
z0 = np.sqrt((2 * m *(V_0)* 1.6E-13) * R**2 / (h**2))
```

```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\804709956.py:8: RuntimeWarning:
invalid value encountered in sqrt
  k1 = np.sqrt((2 * m *(E - V_0)* 1.6E-13) / (h**2))
```

Formula used to convert roots to V_0

In [72]:
```
def V_0finder(x):
    a = np.sqrt(x**2 + (k2*R)**2)
    z = a/np.sqrt(2*m*(R**2)/h**2)
    z = z**2 / 1.6E-13
    return z
V_0finder(1.839)
```

Out[72]:
```
34.14402517965981
```

Bisection Method formula

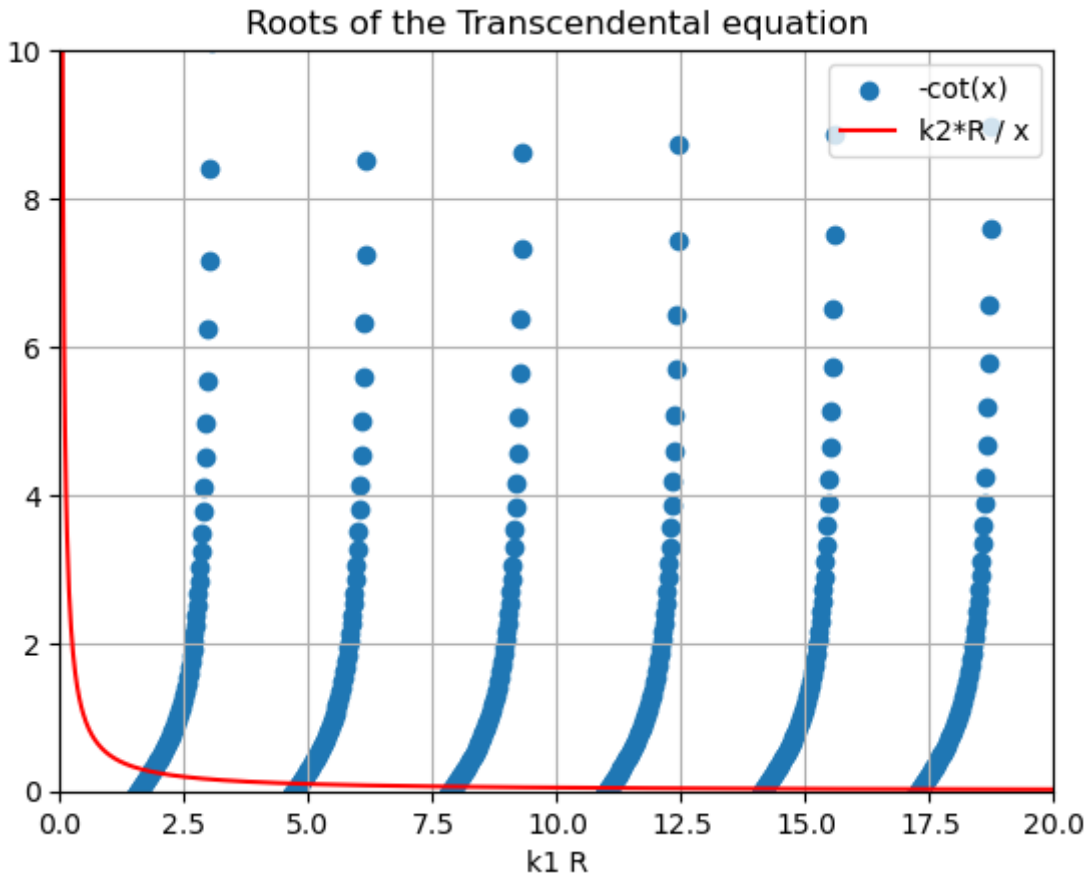
In [116…
```
def bisection(a,b):

    if (func(a) * func(b) >= 0):
        print("incorrent a and b\n")
        return

    c = a
    while ((b-a) >= 0.01):

        # Find middle point
        c = (a+b)/2

        # Check if root
        if (func(c) == 0.0):
            break

        # Decide the side to repeat
        if (func(c)*func(a) < 0):
            b = c
        else:
            a = c

    print("The root is : ","%.4f"%c)
```

Roots of Transcendental Equation

In [143…
```
x=np.linspace(0,20,1000)
plt.scatter(x,-1/np.tan(x))
plt.plot(x,k2*R/x,c="red")
plt.xlim(0,20)
plt.ylim(0,10)
plt.grid(True)
plt.legend(["-cot(x)","k2*R / x"],loc='upper right')
plt.title("Roots of the Transcendental equation");
plt.xlabel("k1 R")
;
```

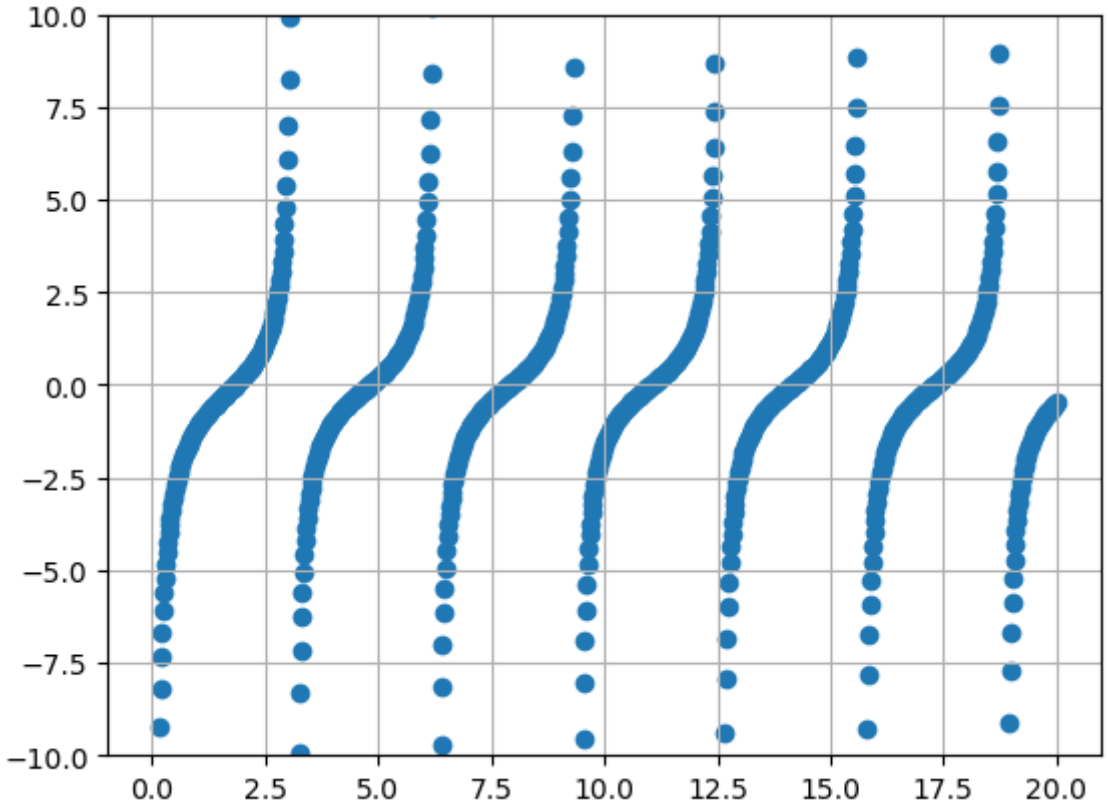
```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\2667406668.py:2: RuntimeWarning:
divide by zero encountered in true_divide
  plt.scatter(x,-1/np.tan(x))
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\2667406668.py:3: RuntimeWarning:
divide by zero encountered in true_divide
  plt.plot(x,k2*R/x,c="red")
```

Out[143]:



In [300…
```
x=np.linspace(0,20,1000)
plt.scatter(x,-1/np.tan(x)-k2*R/x)
plt.ylim(-10,10)
plt.grid(True)
def func(x):
    return -1/np.tan(x)-k2*R/x
```

```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\3276182556.py:2: RuntimeWarning:
divide by zero encountered in true_divide
  plt.scatter(x,-1/np.tan(x)-k2*R/x)
```

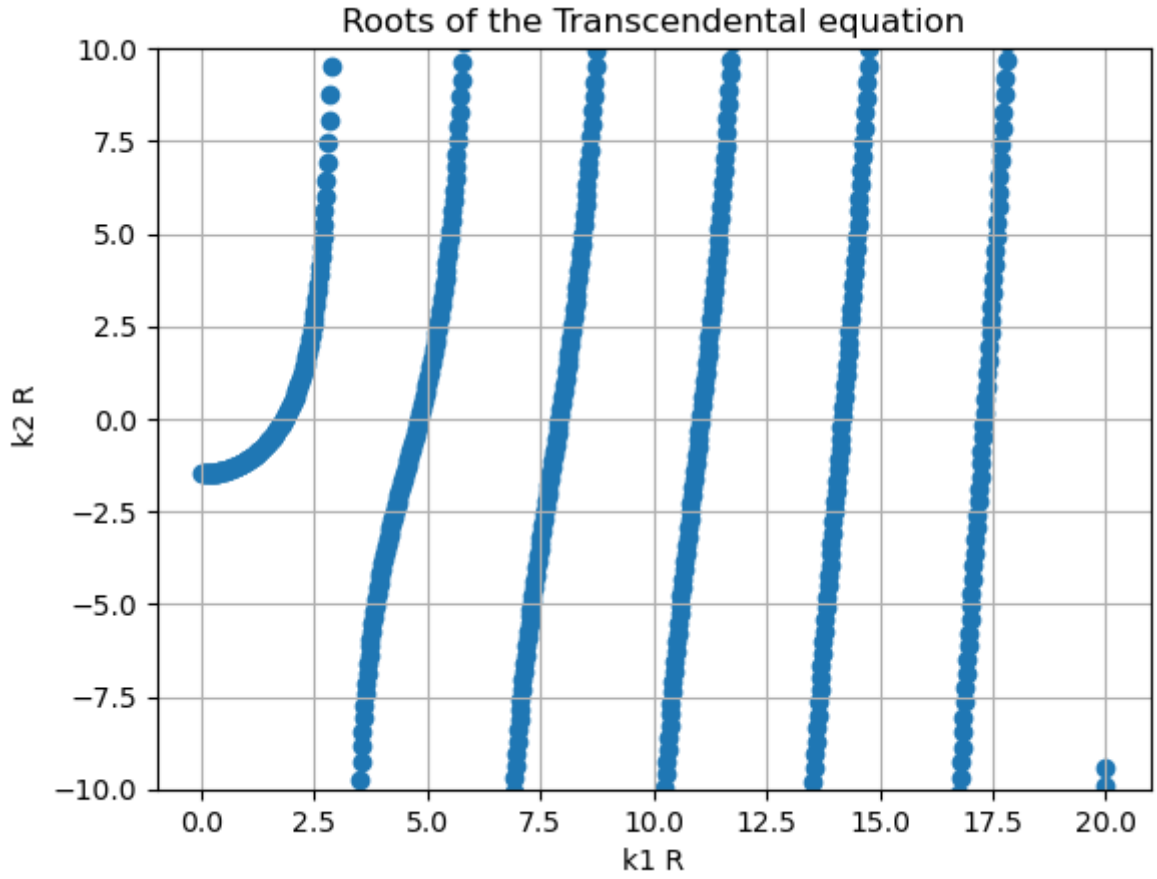


In [119…

```
plt.scatter(x,-x/np.tan(x)-k2*R)
plt.ylim(-10,10)
plt.grid(True)
plt.title("Roots of the Transcendental equation");
plt.xlabel("k1 R")
plt.ylabel(" k2 R")
;
def func(x):
    return -x/np.tan(x)-k2*R
a = 0
b = 2
bisection(a, b)
```

```
The root is :  1.8359
```

```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\1147602030.py:1: RuntimeWarning:
invalid value encountered in true_divide
  plt.scatter(x,-x/np.tan(x)-k2*R)
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\1147602030.py:9: RuntimeWarning:
invalid value encountered in double_scalars
  return -x/np.tan(x)-k2*R
```



Newton Raphson Algorithim

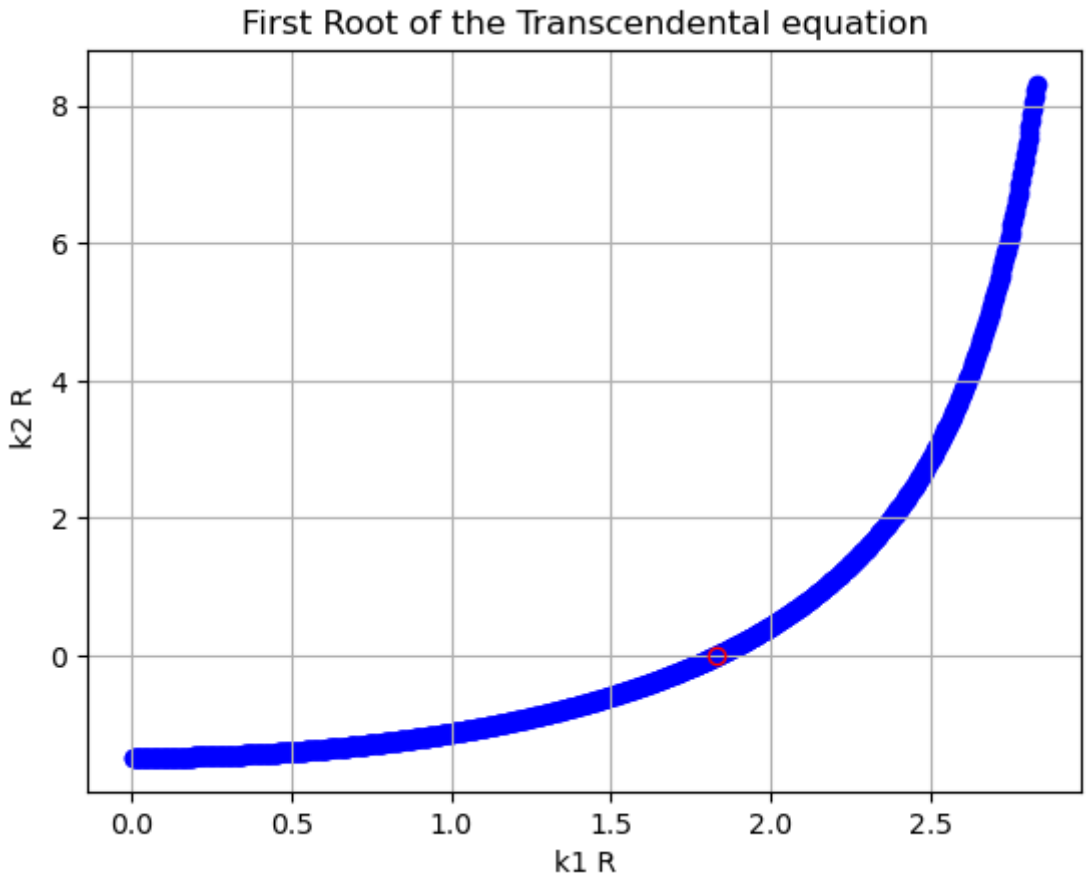
In [120…

```
# Function
def f(x):
    return -x/np.tan(x)-k2*R

# derivative of function
def g(x):
    return -1/np.tan(x) + x/((np.sin(x))**2)

# Newton Raphson Method

def newtonRaphson(x0,e,N):
    step = 1
    flag = 1
    condition = True
    while condition:
        if g(x0) == 0.0:
            print('Divide by zero error')
            break

        x1 = x0 - f(x0)/g(x0)
        print('Run-%d, x1 = %0.6f and f(x1) = %0.6f' % (step, x1, f(x1)))
        x0 = x1
        step = step + 1

        if step > N:
            flag = 0
            break

        condition = abs(f(x1)) > e

    if flag==1:
        print('\nRequired root is: %0.8f' % x1)
```

```
        def V_0finder(x):
            a = np.sqrt(x**2 + (k2*R)**2)
            z = a/np.sqrt(2*m*(R**2)/h**2)
            z = z**2 / 1.6E-13
            return z
        print('V0 is: %0.8f' % V_0finder(x1) ,'MeV')
        # Creating Data for the Line
        x_plot = np.linspace(0, x1+1, 1000)
        y_plot = f(x_plot)
        # Plotting Function
        fig = plt.figure()
        plt.scatter(x_plot, y_plot, c="blue")
        plt.plot(x1, f(x1), c="red", marker="o", fillstyle="none")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.title("First Root of the Transcendental equation");
        plt.xlabel("k1 R")
        plt.ylabel(" k2 R")
        plt.grid()
        plt.show()
    else:
        print('\nNot Convergent.')


# Input Section
x0 = input('Enter Guess: ')
e = input('Tolerable Error: ')
N = input('Maximum Step: ')

# Converting x0 and e to float
x0 = float(x0)
e = float(e)

# Converting N to integer
N = int(N)

# Starting Newton Raphson Method
newtonRaphson(x0,e,N)
```

```
Enter Guess: 1.9
Tolerable Error: 1E-15
Maximum Step: 20
Run-1, x1 = 1.833836 and f(x1) = 0.007694
Run-2, x1 = 1.830395 and f(x1) = 0.000019
Run-3, x1 = 1.830387 and f(x1) = 0.000000
Run-4, x1 = 1.830387 and f(x1) = 0.000000

Required root is: 1.83038684
V0 is: 33.84577967 MeV
```

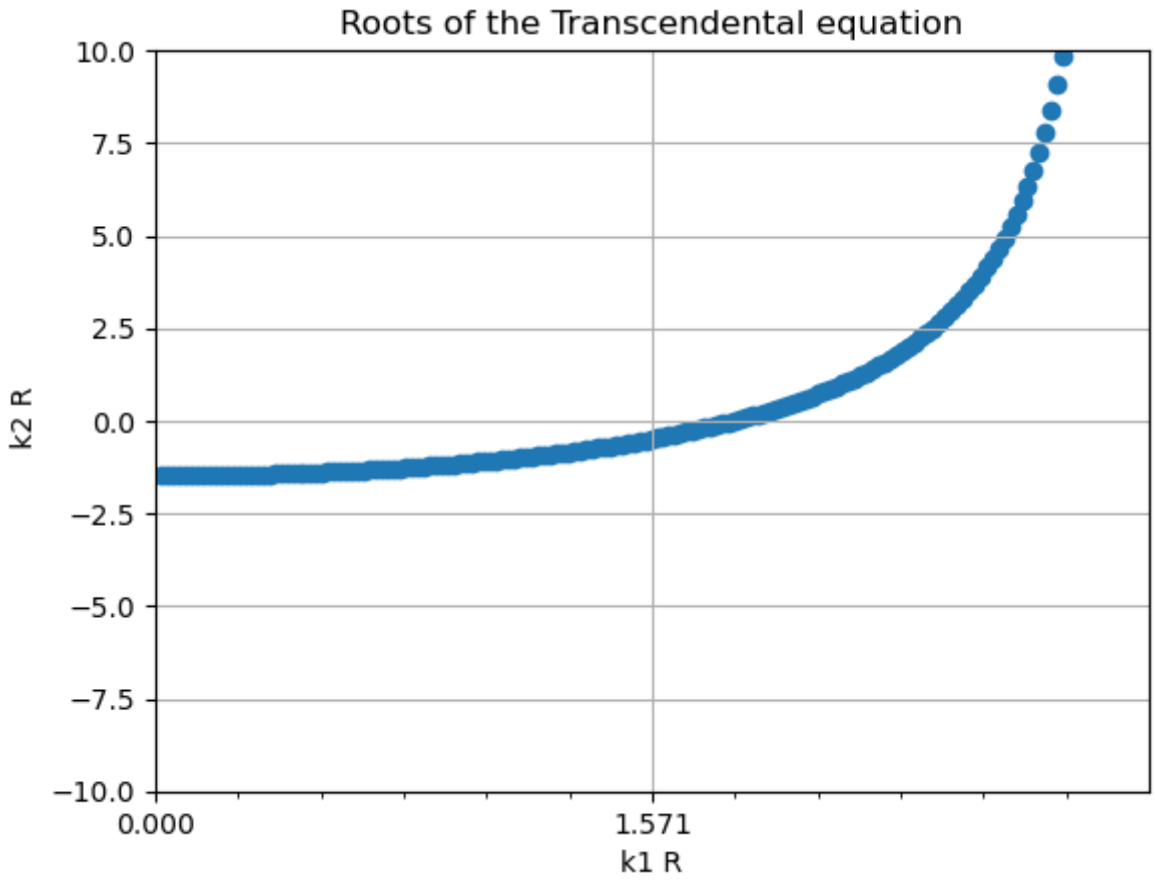
```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\4099048517.py:3: RuntimeWarning:
invalid value encountered in true_divide
  return -x/np.tan(x)-k2*R
```



In [124…

```
x = np.linspace(0, 3*np.pi,500)
plt.scatter(x,-x/np.tan(x)-k2*R)
plt.ylim(-10,10)
plt.title("Roots of the Transcendental equation");
plt.xlabel("k1 R")
plt.ylabel(" k2 R")
plt.grid(True)
plt.xlim(0,3.14)
ax = plt.gca()
ax.xaxis.set_major_locator(plt.MultipleLocator(np.pi / 2))
ax.xaxis.set_minor_locator(plt.MultipleLocator(np.pi / 12))
plt.show()
```

```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\4271252653.py:2: RuntimeWarning:
invalid value encountered in true_divide
  plt.scatter(x,-x/np.tan(x)-k2*R)
```

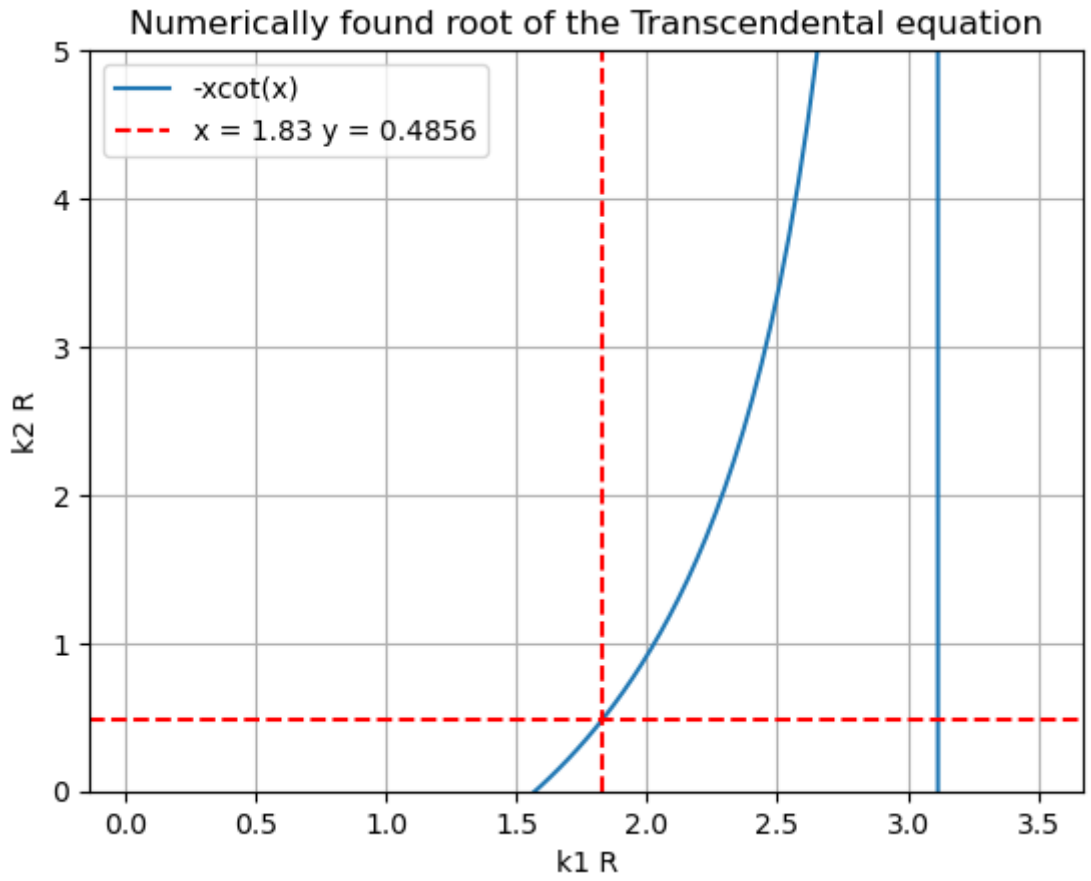


Bound States of Deuteron

In [128…

```
x = np.linspace(0,3.5,100)
plt.plot(x,-x/np.tan(x))
plt.axhline(y = 0.4856836, color = 'r', linestyle = '--')
plt.axvline(x = 1.83, color = 'r', linestyle = '--')
plt.ylim(0,5)
plt.title("Numerically found root of the Transcendental equation");
plt.xlabel("k1 R")
plt.ylabel("k2 R")
plt.grid(True)
plt.legend(["-xcot(x)","x = 1.83 y = 0.4856"]);
```

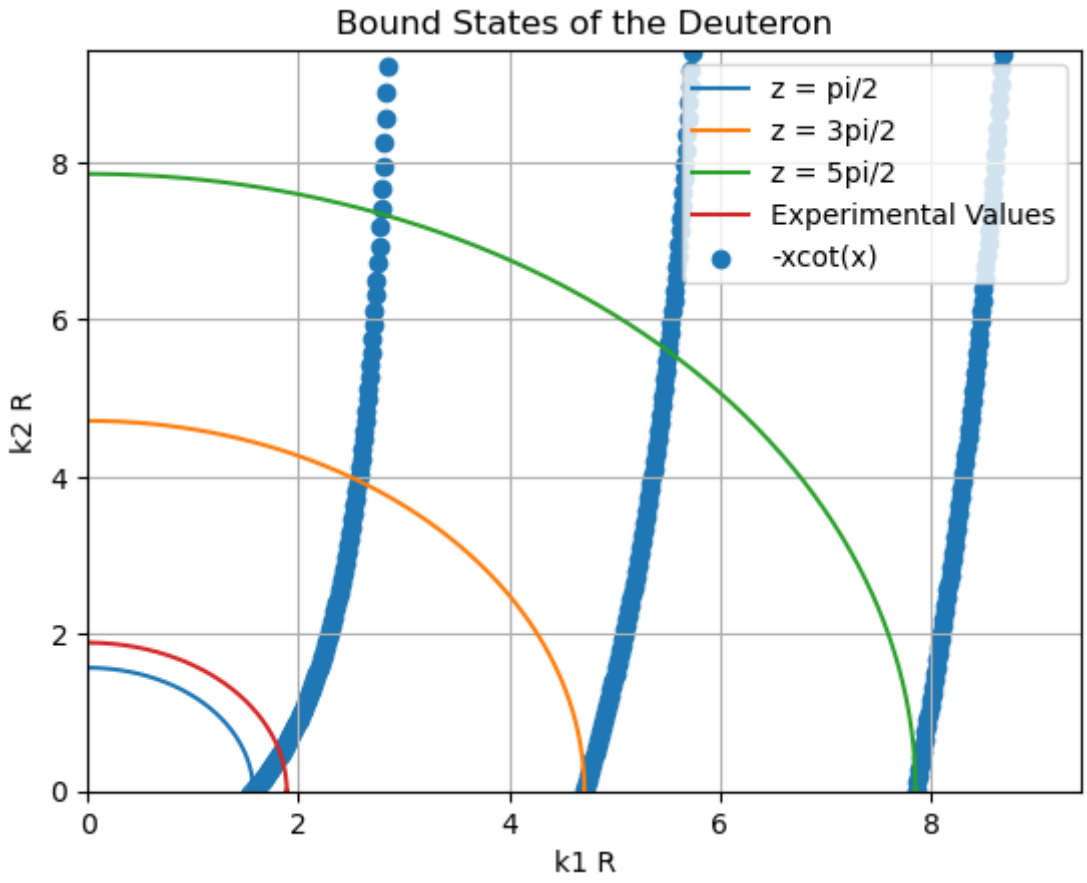
```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\1391926135.py:2: RuntimeWarning:
invalid value encountered in true_divide
  plt.plot(x,-x/np.tan(x))
```



In [235…

```
# x**2  + y**2 = r**2

theta = np.linspace(0, 2*np.pi, 1000)
r= np.pi/2
x = r*np.cos(theta)
y = r*np.sin(theta)
plt.plot(x,y)

r= 3*np.pi/2
x = r*np.cos(theta)
y = r*np.sin(theta)
plt.plot(x,y)

r= 5*np.pi/2
x = r*np.cos(theta)
y = r*np.sin(theta)
plt.plot(x,y)

r= 1.89
x = r*np.cos(theta)
y = r*np.sin(theta)
plt.plot(x,y)
x=np.linspace(0,10,1000)

plt.scatter(x,-x/np.tan(x))
plt.xlim(0,3*np.pi)
plt.ylim(0,3*np.pi)
plt.grid(True)

plt.title("Bound States of the Deuteron");
plt.xlabel("k1 R")
plt.ylabel("k2 R")
plt.grid(True)
plt.legend(["z = pi/2","z = 3pi/2","z = 5pi/2","Experimental Values", "-xcot(x)"],
```

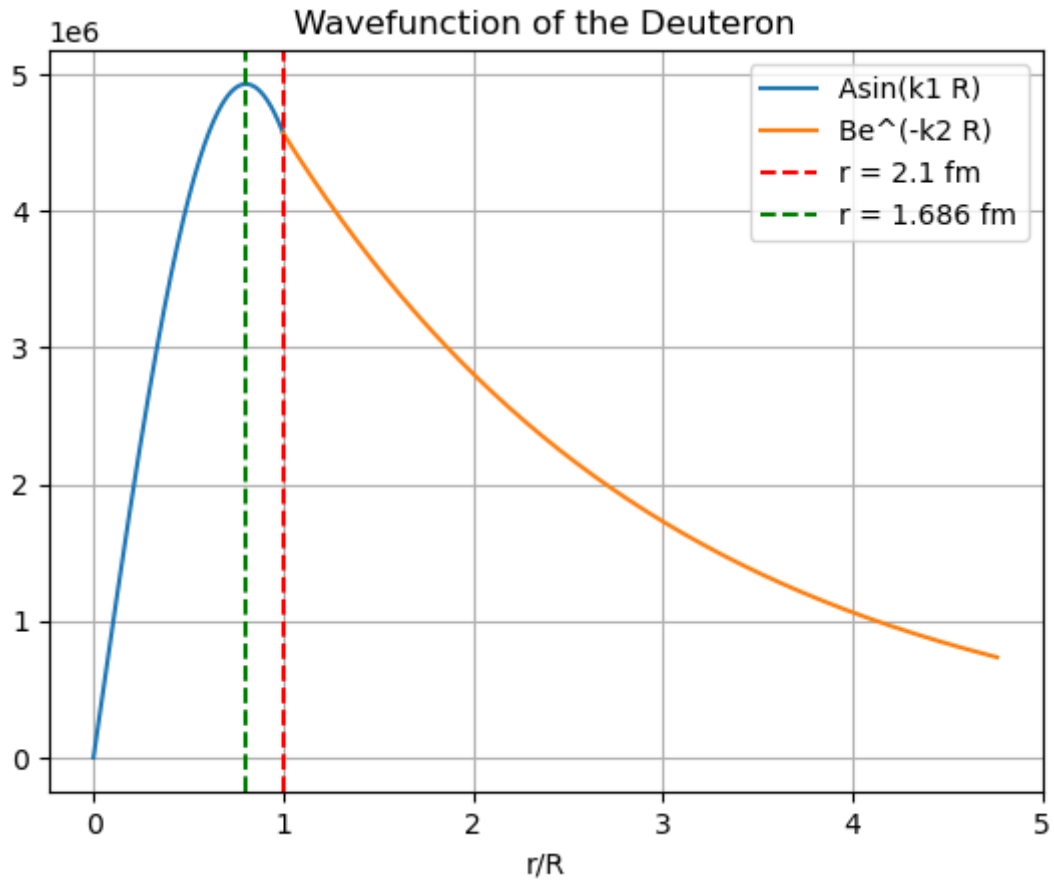
```
C:\Users\natha\AppData\Local\Temp\ipykernel_13604\3286943720.py:25: RuntimeWarnin
g: invalid value encountered in true_divide
  plt.scatter(x,-x/np.tan(x))
```



Wavefunction of Deuteron

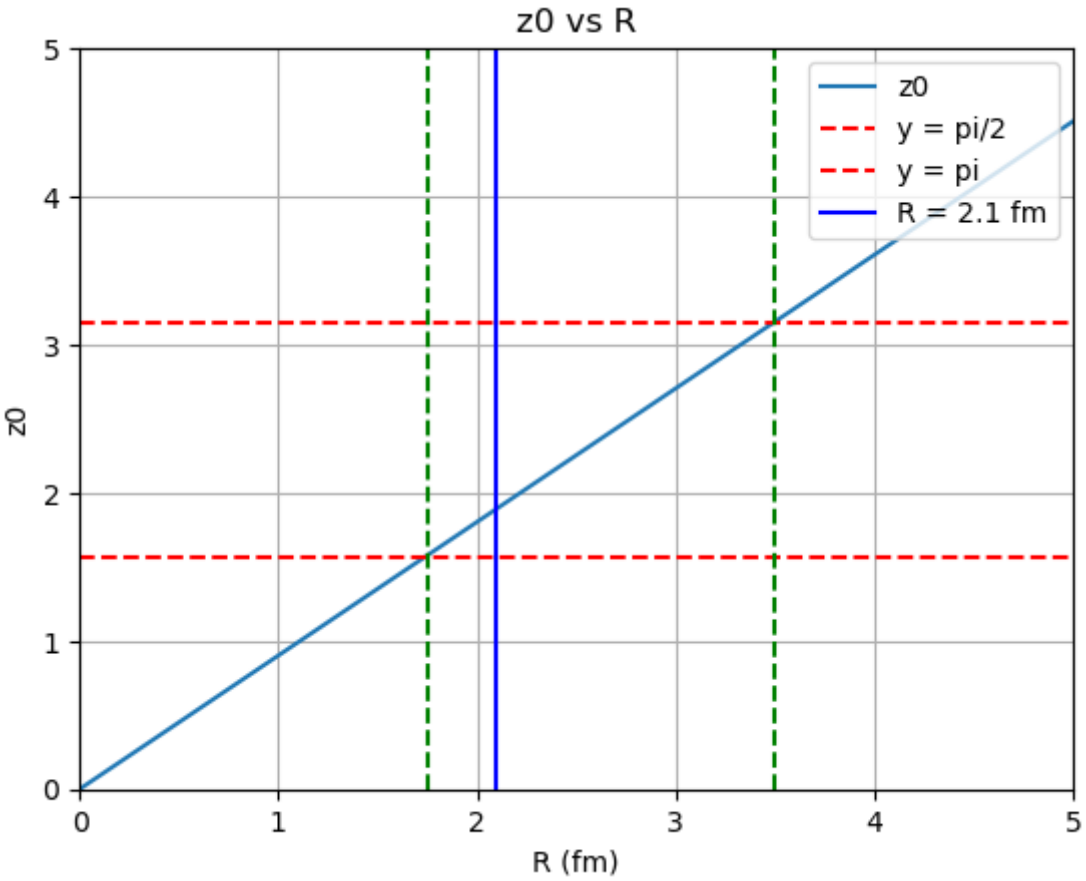
In [255…

```
k1 = np.sqrt((2 * m *(E + V_0)* 1.6E-13) / (h**2))
k2 = np.sqrt(((2 * m * E)* 1.6E-13 / (h**2)))
A1 = np.exp(k2*R) * np.sqrt(k1*R*k2*R) * np.sin(k1*R)
A11 = np.sqrt(k1*R*k2*R)
A2 = np.sqrt(np.pi * R) * np.sqrt(2*k1*R*k2*R + 2*k1*R* ((np.sin(k1*R))**2) - 2*k2
x = np.linspace(0,2.1E-15,10000)
plt.plot(x/R,(A11/A2)*np.sin(x*k1))
x= np.linspace(2.1E-15,10E-15,10000)
plt.plot(x/R,(A1/A2)*np.exp(-k2*x))
plt.axvline(x = 1, color = 'r', linestyle = '--')
plt.axvline(x = 0.803, color = 'g', linestyle = '--')
plt.grid(True)
plt.title("Wavefunction of the Deuteron");
plt.xlabel("r/R")
plt.legend(["Asin(k1 R)","Be^(-k2 R)","r = 2.1 fm", "r = 1.686 fm"], loc ="upper r
```



In [243…

```
def z(R):
    return np.sqrt(((2*m*V_0*1.6E-13)*R**2)/h**2)*1E-15
x = np.linspace(0,10,20)
plt.plot(x,z(x))
plt.xlim(0,5)
plt.ylim(0,5)
plt.grid(True)
plt.axhline(y = np.pi/2, color = 'r', linestyle = '--')
plt.axhline(y = np.pi, color = 'r', linestyle = '--')
plt.axvline(x = 2.1, color = 'b', linestyle = '-')
plt.axvline(x = 1.75, color = 'g', linestyle = '--')
plt.axvline(x = 3.5, color = 'g', linestyle = '--')
plt.title("z0 vs R")
plt.ylabel("z0")
plt.xlabel("R (fm)")
plt.legend(["z0","y = pi/2","y = pi", "R = 2.1 fm"], loc ="upper right");
```


z0 vs R
z0
y = pi/2
y = pi
R = 2.1 fm
z0
R (fm)
0
1
2
3
4
5