Computational Physics
Final Project

Kyungmin Park
Department of Physics, University of Seoul

Topic

- Solve numerically the radial Schrodinger equation.

- Given a spherically symmetric potential,
- Obtain numerically the radial wavefunctions in a spherically symmetric potential.
- Analyze the eigen energies and their corresponding eigenstates(eigen functions).

Goal

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

→ Find En and R(r)!

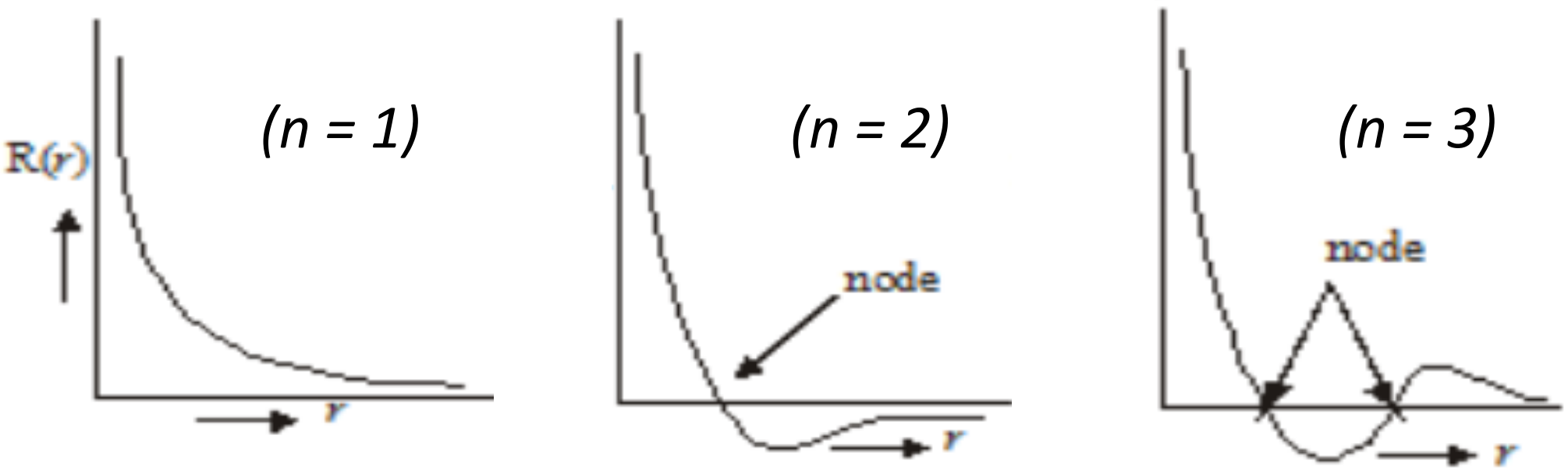
Idea – Abstract

- When given an arbitrary energy value E, you can get a corresponding radial wavefunction numerically by integration → *runge-kutta* method.
- If the arbitrary energy value is “appropriate” (E: nth eigenenergy), the corresponding radial wavefunction obtained is the nth eigenfunction.

→First, find the nth eigenenergy.

→Then, obtain the corresponding eigenfunction.

Idea – 1. Find the nth eigenenergy



- The number of nodes = n – 1

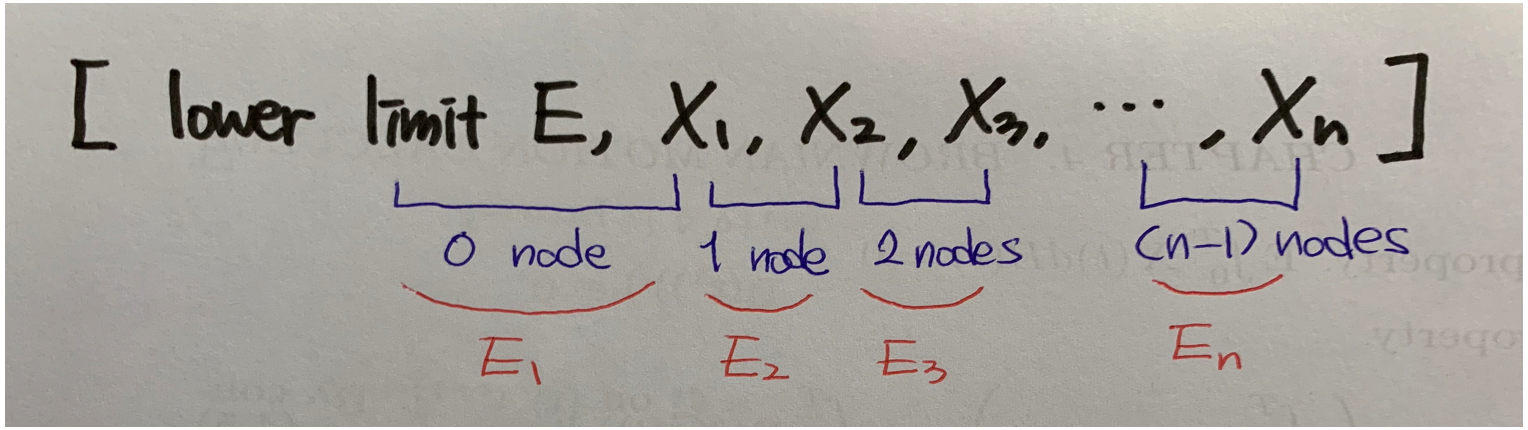
→ nth eigenfunction has (n-1) nonzero root(s).

Idea – 1. Find the nth eigenenergy

- Step #1. Starting from lower limit E with an appropriate E step value,
- Step #2. Obtain its corresponding radial wave function using runge-kutta method.
- Step #3. For each wave function, find the number of nonzero roots(nodes).
- Step #4. While looping, at certain energy values, the number of nonzero roots increases by one.
- Step #5. Every time step #4 happens, mark the energy value: append the value in an array.

Idea – 1. Find the nth eigenenergy

- After these steps, we get



- nth eigenvalue lies in the range [Xn-1, Xn]

→ From the possible range we got for the nth eigenvalue, we need to find the value E closest to the eigenvalue.

Idea – 1. Find the nth eigenenergy

- Radial wave function obtained corresponding to E values that are definitely not an eigenvalue will not satisfy the boundary condition for a normalizable wave function. – Psi = 0 when r is large enough.
- Between the range [Xn-1, Xn], find a E value whose corresponding radial wave function satisfies the boundary condition.

→ nth eigenvalue En obtained!

Idea – 2. Find the nth radial eigenfunction

- Obtain the radial wave function corresponding to the nth eigenvalue we got – using the *runge-kutta* module.

→ nth eigenfunction R(r) obtained!

Code Implementation

1. Obtain the possible range for eigenvalues using the runge-kutta module.

```
def calculate(E_lower_limit):
    node = 0
    nodeArray = []
    energyBoundary = []

    E = 0.0
    Estep = 0.01
    nLoop = 0

    energyBoundary.append(E_lower_limit)

    while (node < n):
        E = E_lower_limit + Estep*nLoop
        def energy_in_joule():
            return energy_in_eV()*eV_to_joule

        def energy_in_eV():
            return E

        # runge-kutta
        def F(x,y):
            F = np.zeros(2)
            F[0] = y[1]
            F[1] = ((potentialV(x)-energy_in_joule())/C)*y[0]
            return F
```

```
# at r = 0
y = np.array([0.0, 1.0*angstr])

h = 0.1*angstr
freq = 2
X,Y = integrate(F,x,y,xStop,h)

# the number of nodes
node = 0
for i in range(len(Y)-1):
    if ( np.sign(Y[i,0]) != np.sign(Y[i+1,0]) ):
        node = node + 1
node = node -1
nodeArray.append(node)
if ( len(nodeArray) > 1 ):
    if ( nodeArray[nLoop-1] != nodeArray[nLoop] ):
        energyBoundary.append(round(E,4))

nLoop = nLoop + 1
```

Code Implementation

2. Between the range, determine the eigenvalue whose corresponding eigenfunction satisfies the boundary condition.

```
box_boundary = []
energy = energyBoundary[n-1]

nstep = 0

while (energy < energyBoundary[n]):
    energy = (energyBoundary[n-1] + Estep*nstep)
    energy_in_joule = energy*eV_to_joule
    if (energy != energyBoundary[n]):
        # runge-kutta
        def G(x,y):
            G = np.zeros(2)
            G[0] = y[1]
            G[1] = ((potentialV(x)-energy_in_joule)/C)*y[0]
            return G
        X1,Y1 = integrate(G,x,y,xStop,h)

        box_boundary.append(abs( Y1[len(Y1)-1,0] - 0.0))
    nstep = nstep + 1

index = np.argmin(box_boundary)
eigenvalue = energyBoundary[n-1] + index*Estep
```

Code Implementation

3. Find the radial wave function corresponding to the eigenvalue, and normalize it.

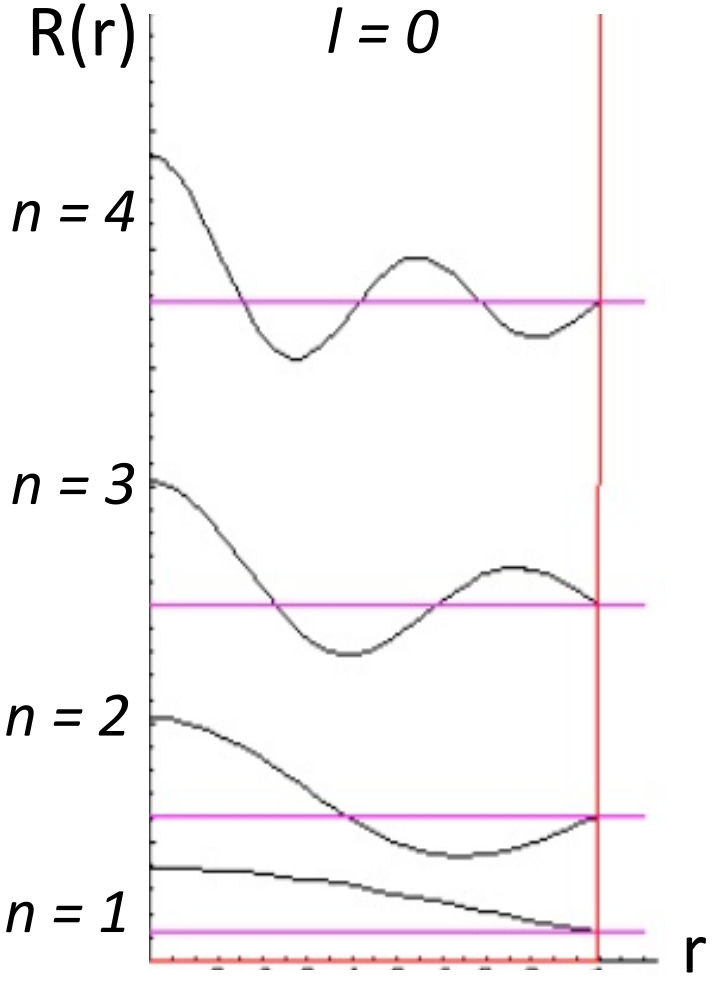
```
# before normalization
R = []
for i in range(len(Y)):
    R.append(Y[i,0]/(X[i]+1e-30))

# normalization
Rnorm = []
integ = 0.0
for i in range(len(R)):
    Rnorm.append(R[i]**2*X[i]**2)
    integ = integ + h*(Rnorm[i])

Rafter = []
for i in range(len(R)):
    Rafter.append(R[i]/integ)

for i in range(len(R)):
    Rnorm[i] = Rnorm[i]/integ
```

```
# plot
fig = plt.figure()
title = "Radial Function ( n = " + str(n) + ", l = " + str(l) + " )"
plt.title(title,y=1.06)
ax1 = fig.add_subplot(1,2,1)
ax1.plot(X,Rafter,'o')
ax1.axhline(0,color="black")
ax1.set_xlabel("r")
ax1.set_ylabel("R(r)")
ax1.set_xticklabels([])
ax1.set_yticklabels([])
ax1.set_title("R(r)")
ax2 = fig.add_subplot(1,2,2)
ax2.plot(X,Rnorm,'o-')
ax2.axhline(0,color="black")
ax2.set_xlabel("r")
ax2.set_ylabel("R(r)**2*r**2")
ax2.set_xticklabels([])
ax2.set_yticklabels([])
ax2.set_title("R(r)**2*r**2")
plt.show()
```

Application – #1. Infinite Well

- Infinite well – the simplest case!
- Let's verify the algorithm with the simplest case.

- Consider n = 1, l = 0 first
- For now, set the energy lower bound value to zero, which we'll discuss later.



Application – #1. Infinite Well

- A code that will help check the algorithm

→Given the *analytic* solution for eigenenergies, print the energy value and obtain *numerically* the corresponding radial wave function.

→If this code fails – the resulting wave function is different from the analytically obtained wave function, the *runge-kutta* module would be the problem, giving the wrong answer to a differential equation.

Application – #1. Infinite Well

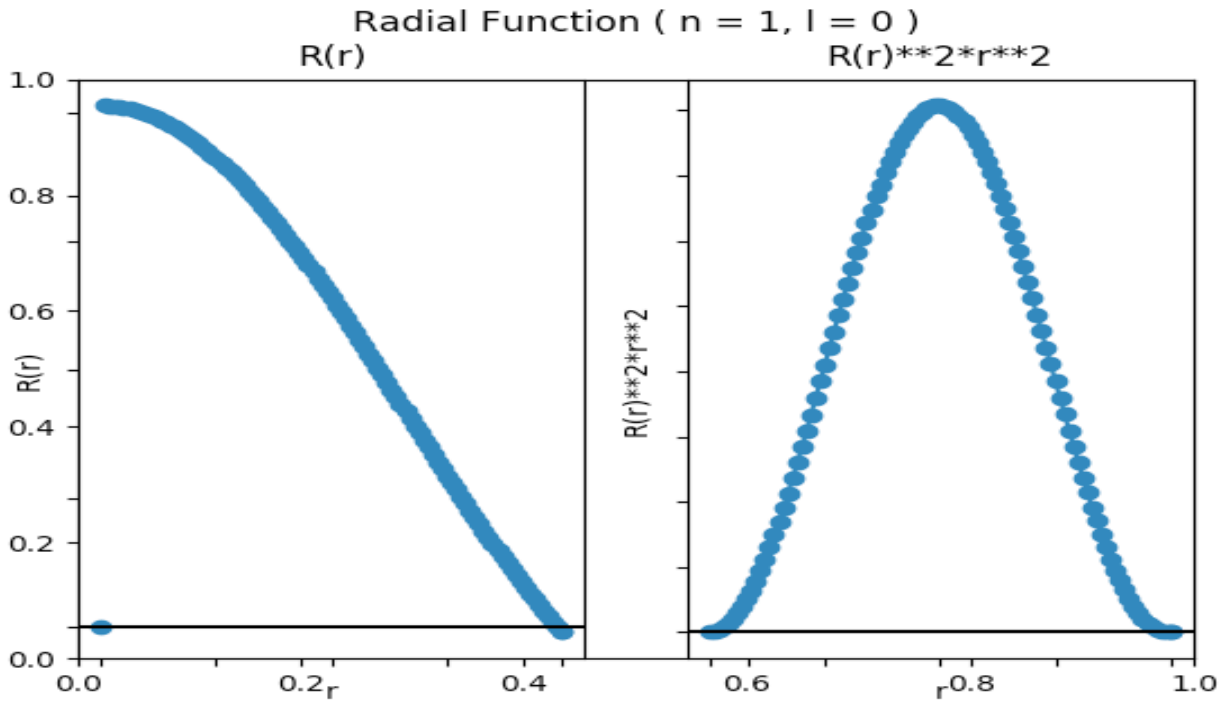
A code that will help check the algorithm

```
# define potential & eigenenergy
n = 1
l = 0

vp = 0.0
# square well (particle in a box)
def potentialVeff(x):
    return (vp+(C*l*(l+1)/(x**2+1e-50)))

def energy():
    return n**2*hbar**2.0*np.pi**2.0/(2.0*me*L**2.0)

print("n =",n,round(energy()*joule_to_eV,2))
```

```
# runge-kutta
def F(x,y):
    F = np.zeros(2)
    F[0] = y[1]
    F[1] = ((potentialVeff(x)-energy())/C)*y[0]
    return F

y = np.array([0.0, 1.0*angstr])

h = 0.1*angstr
freq = 2
X,Y = integrate(F,x,y,xStop,h)
```

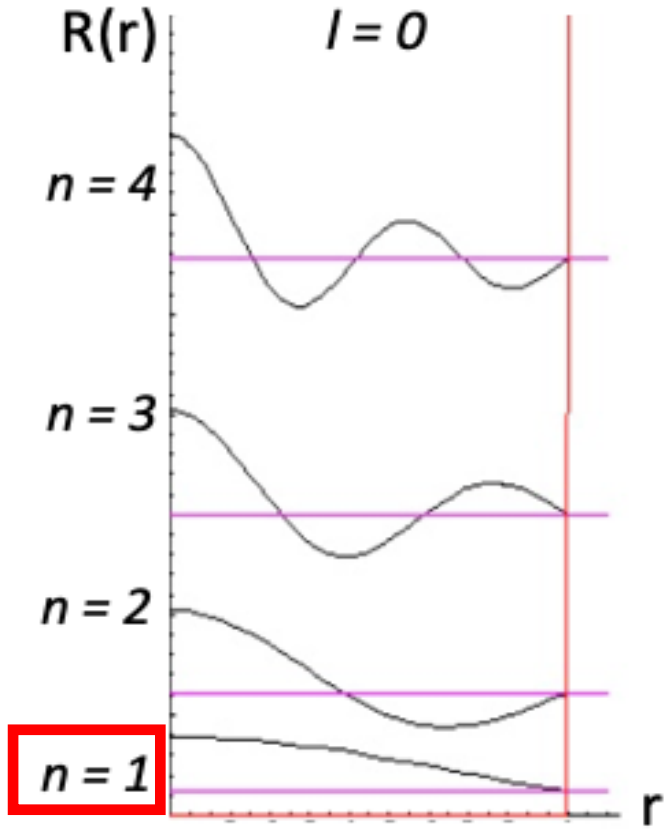
Application – #1. Infinite Well

Numerical Output



→Integration part of the code is verified!

Analytic Solution



Application – #1. Infinite Well

My code

```
n = 1
l = 0

# define potential
vp = 0
def potentialVeff(x):
    return (vp+(C*l*(l+1)/(x**2+1e-50)))
```

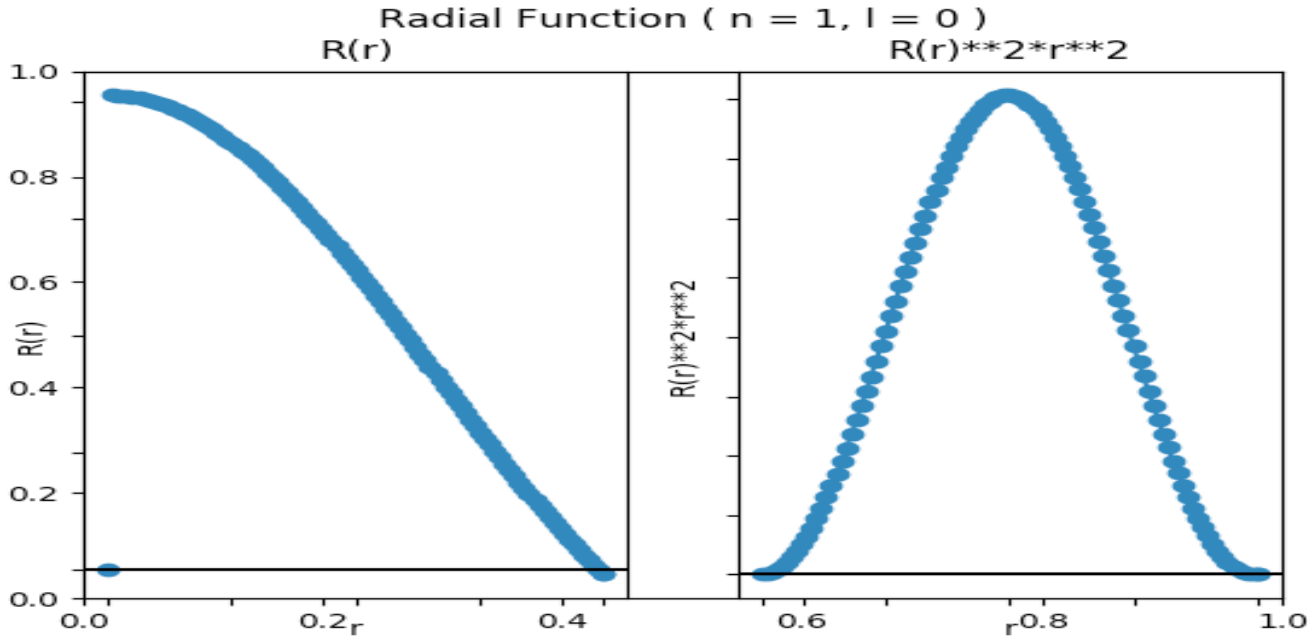
Application – #1. Infinite Well

- Verify the eigenvalue-finding part of the code.

→ From n = 1 to n = 4, compare the results with the analytic ones.

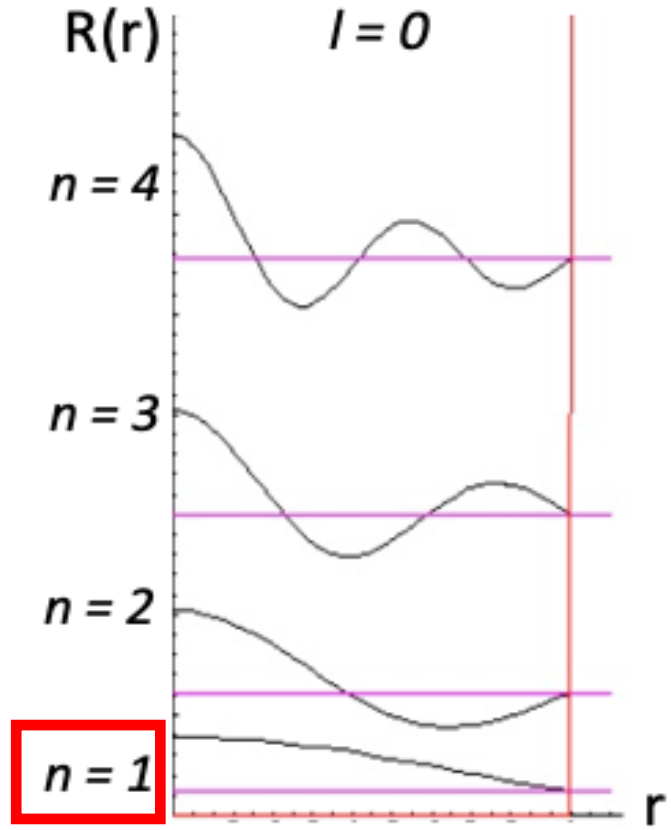
Application – #1. Infinite Well

Numerical Result (n=1)



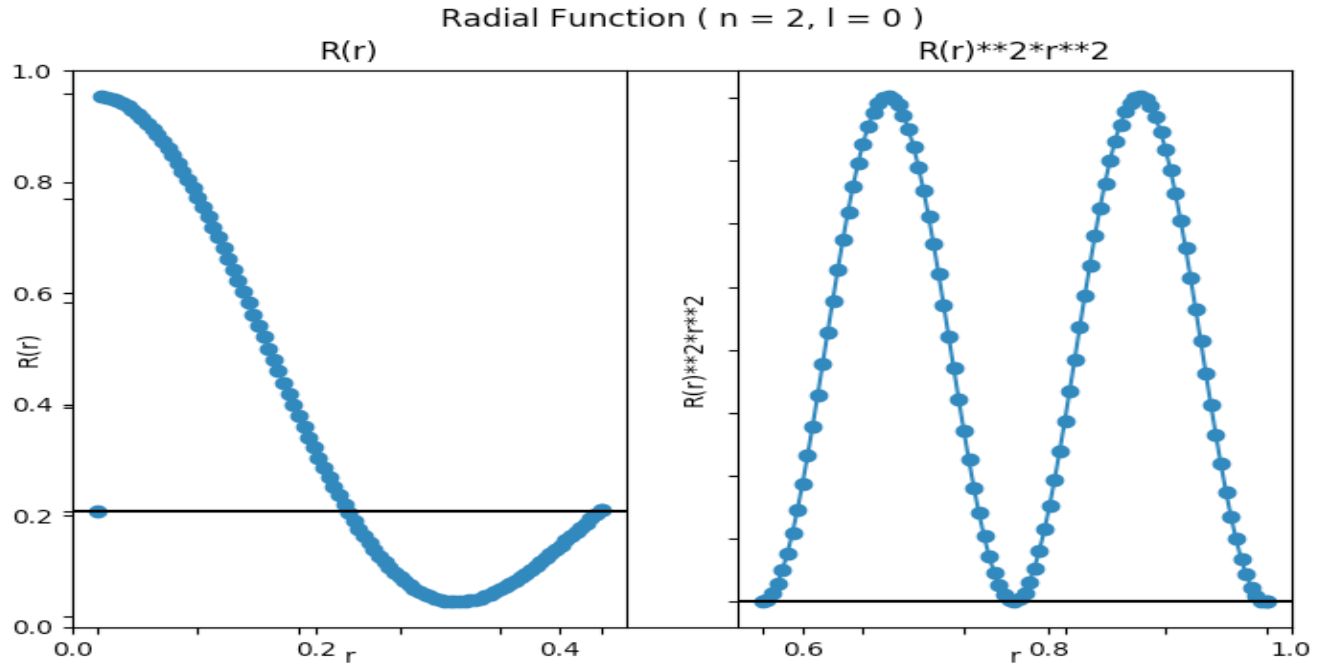

```
Analytic Solution: E = 0.3728218954629418 eV
Eigenvalue Range   [0, 0.38]
Energy eigenvalue at n = 1   -> 0.37 eV
```

Analytic Solution



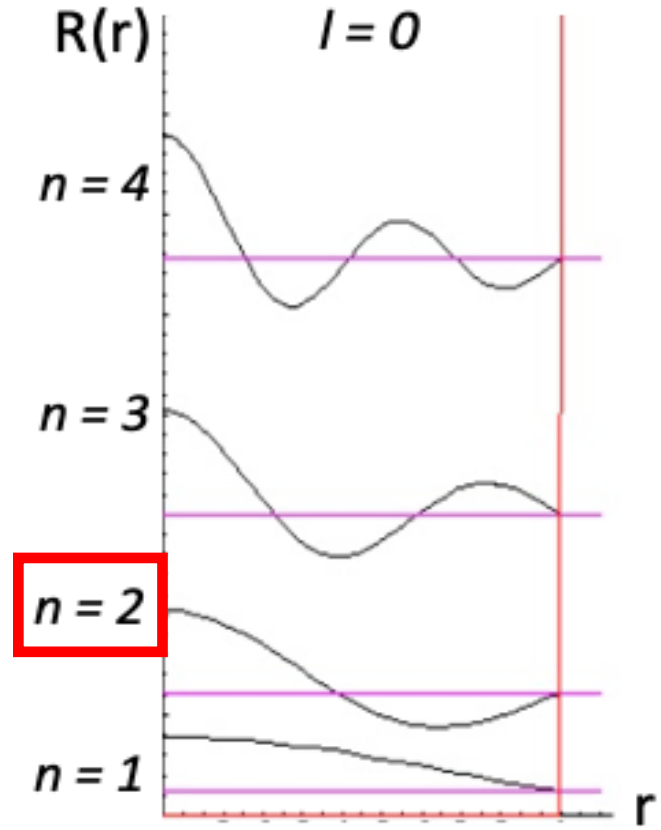
Application – #1. Infinite Well

Numerical Result (n=2)



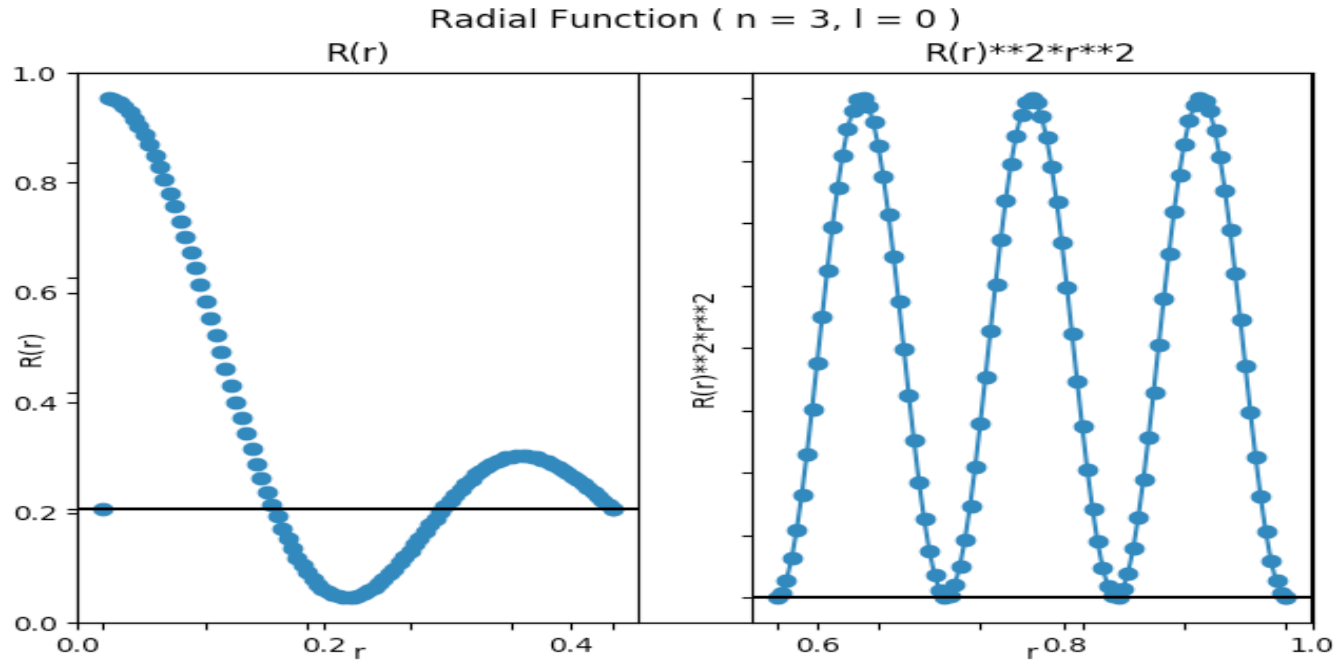

```
Analytic Solution: E = 1.4912875818517672 eV
Eigenvalue Range   [0, 0.38, 1.5]
Energy eigenvalue at n = 2   -> 1.49 eV
```

Analytic Solution



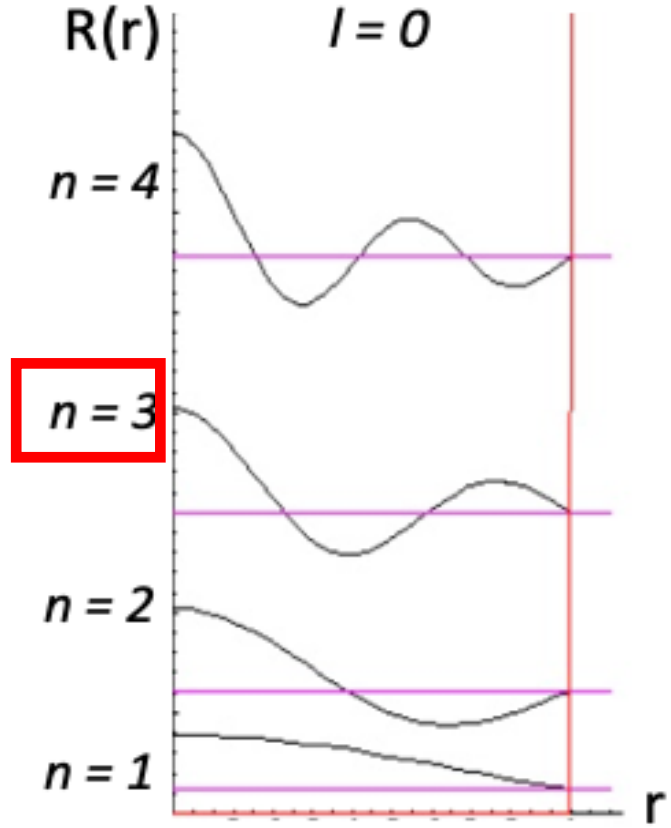
Application – #1. Infinite Well

Numerical Result (n=3)



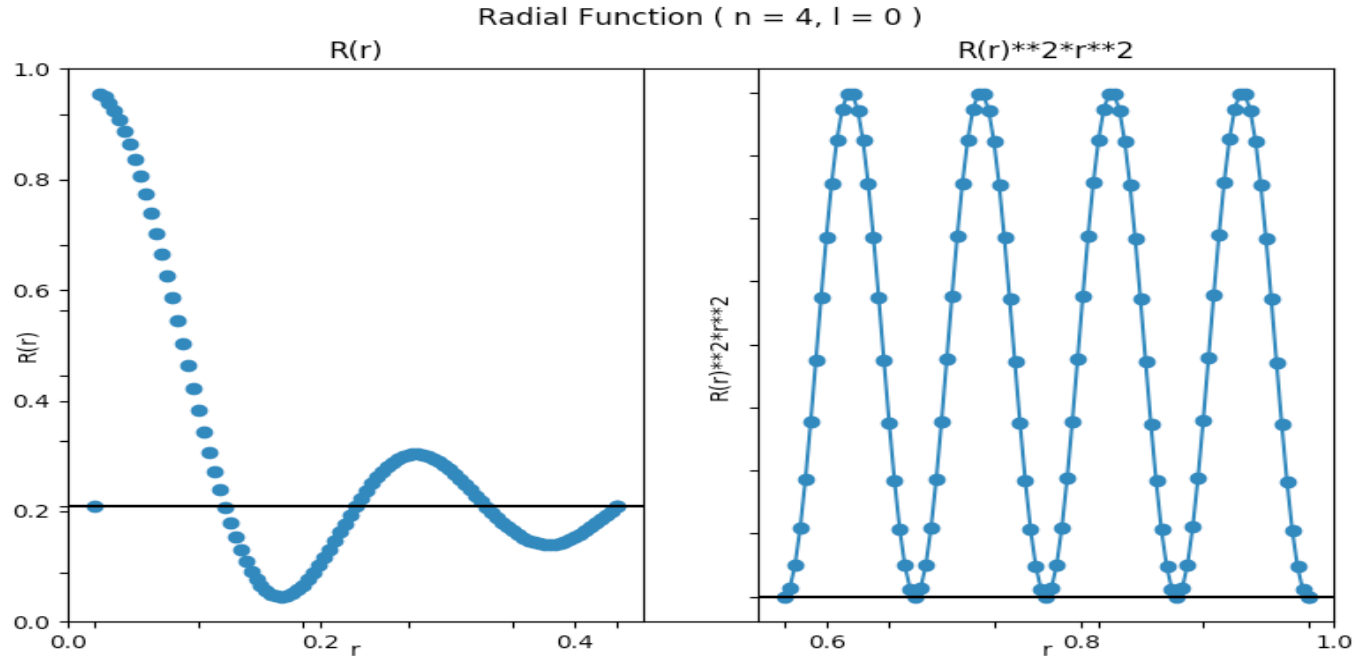

```
Analytic Solution: E = 3.3553970591664766 eV
Eigenvalue Range   [0, 0.38, 1.5, 3.36]
Energy eigenvalue at n = 3   -> 3.36 eV
```

Analytic Solution



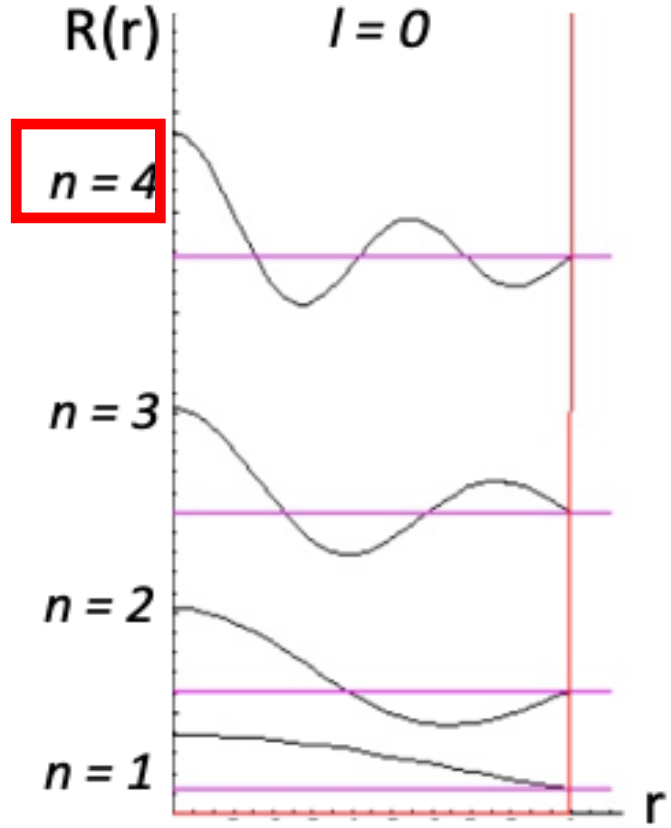
Application – #1. Infinite Well

Numerical Result (n=4)




```
Analytic Solution: E = 5.965150327407069 eV
Eigenvalue Range  [0, 0.38, 1.5, 3.36, 5.97]
Energy eigenvalue at n = 4  -> 5.96 eV
```

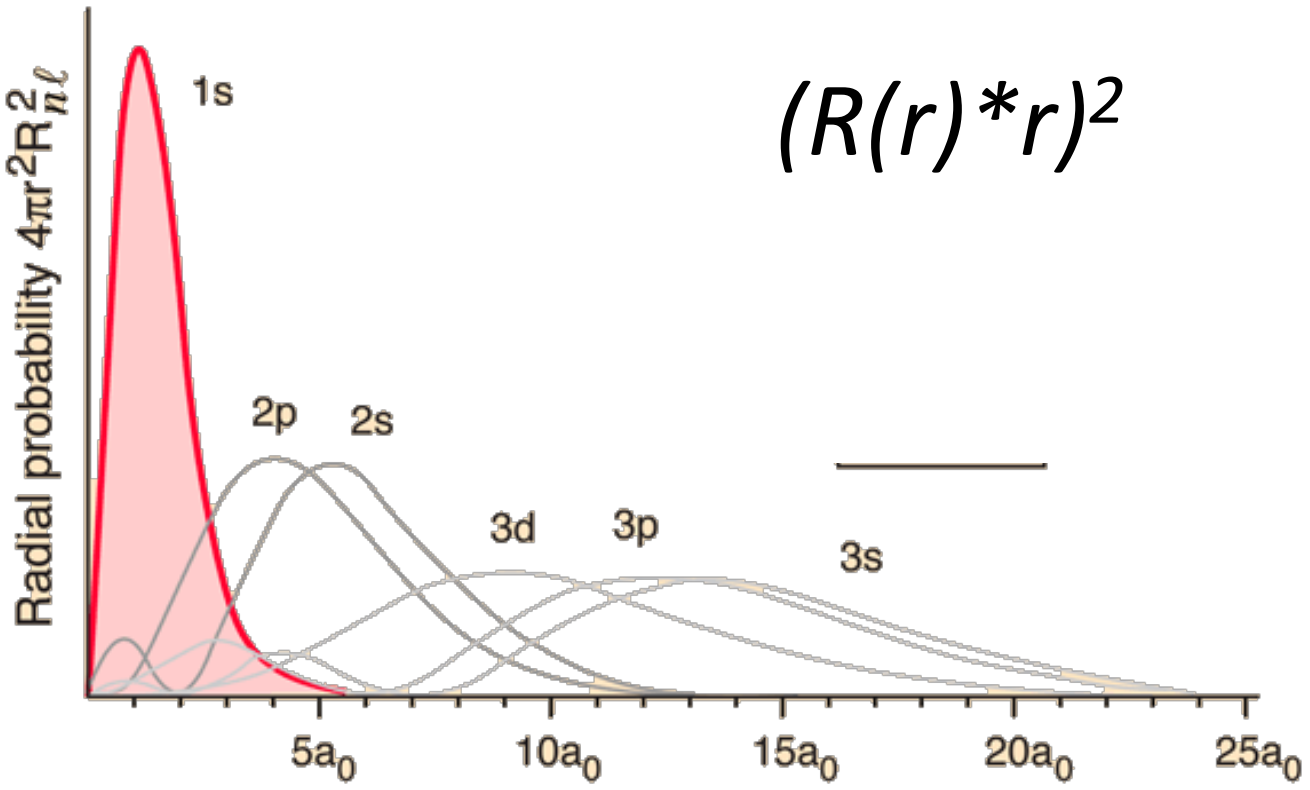
Analytic Solution



Application – #2. Hydrogen Atom

- Consider l = 0 first
- For now, set the energy lower bound value to -20eV, which we'll discuss later.
- From n = 1 to n = 4, compare the result with analytic ones.

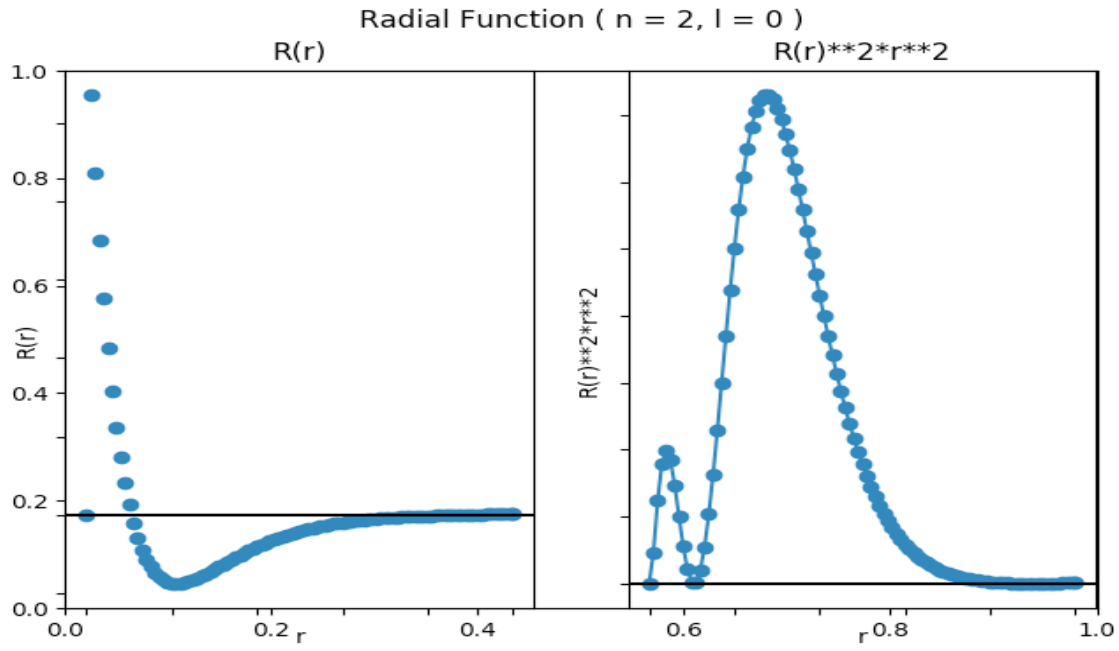
Analytic solution for radial probability



$(R(r)*r)^2$

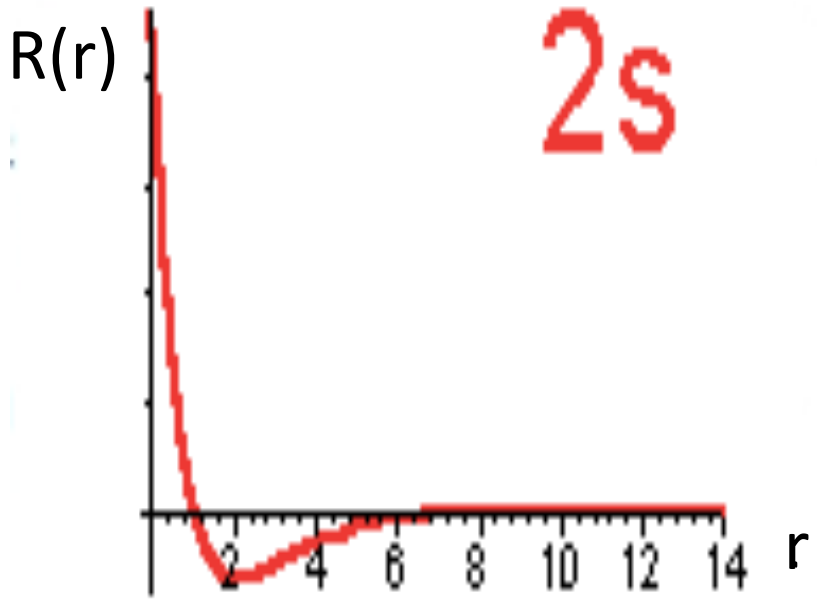
Application – #2. Hydrogen Atom

Result (n=2)




```
Energy eigenvalue at n = 2  -> -3.42 eV
Box size: width = 1e-09
```

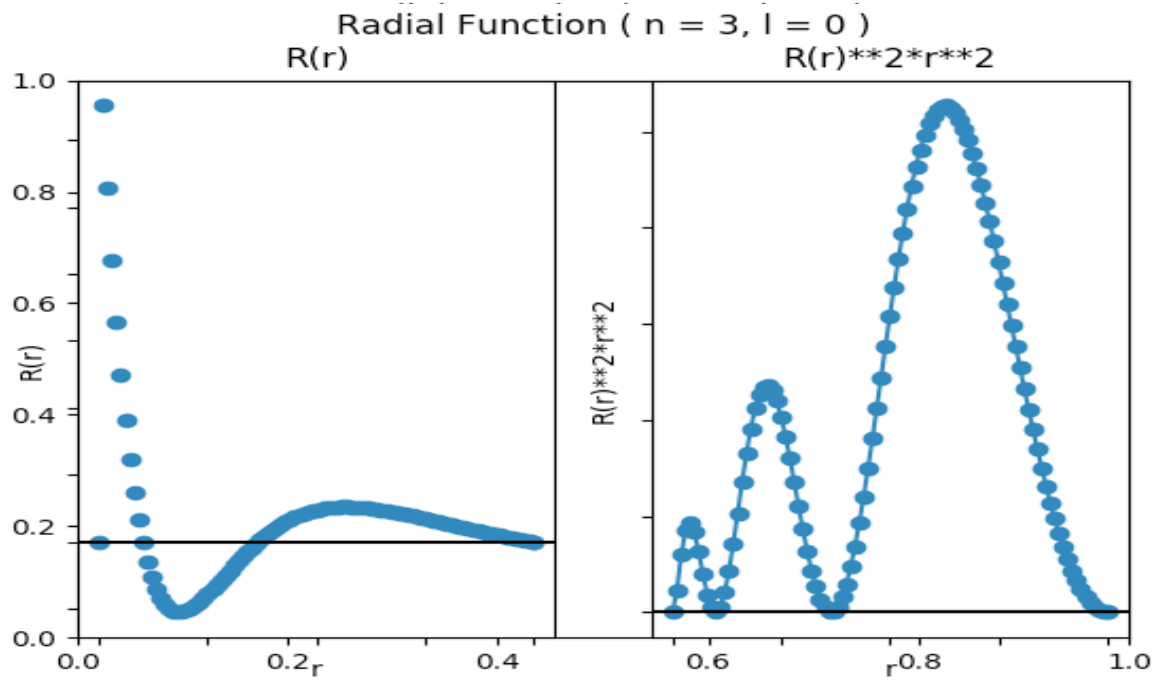
Analytic Solution



$E_2 = -13.6/2^2$ eV = -3.4 eV

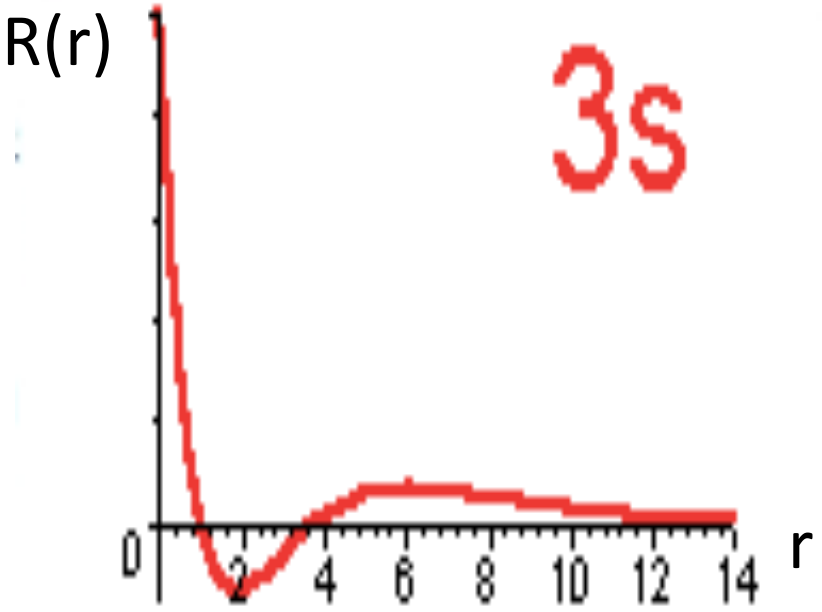
Application – #2. Hydrogen Atom

Result (n=3)




```
Energy eigenvalue at n = 3  -> -1.3 eV
Box size: width = 1e-09
```

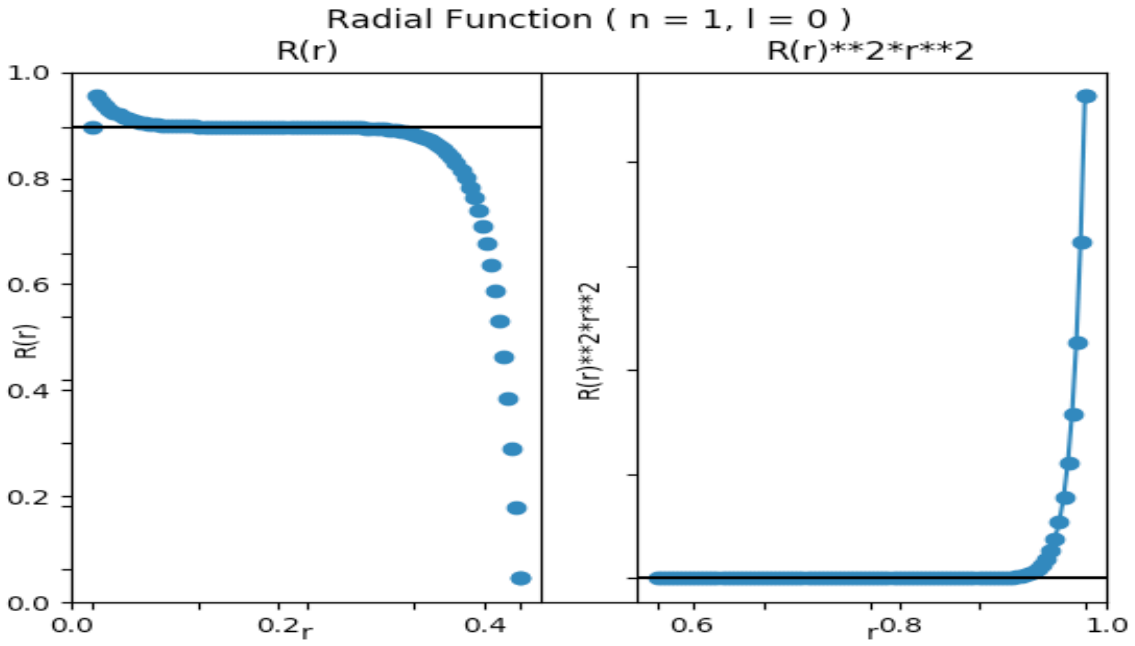
Analytic Solution



$E_3 = -13.6/3^2$ eV = -1.51 eV

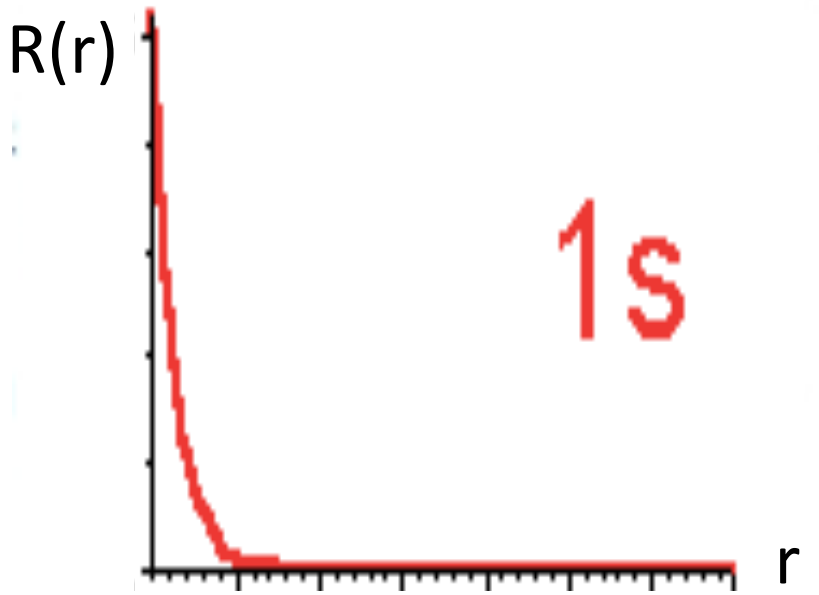
Application – #2. Hydrogen Atom

Result (n=1)




```
Energy eigenvalue at n = 1   -> -13.65 eV
Box size: width = 1e-09
```

Analytic Solution



$E_1 = -13.6/1^2$ eV = -13.6 eV

Application – #2. Hydrogen Atom

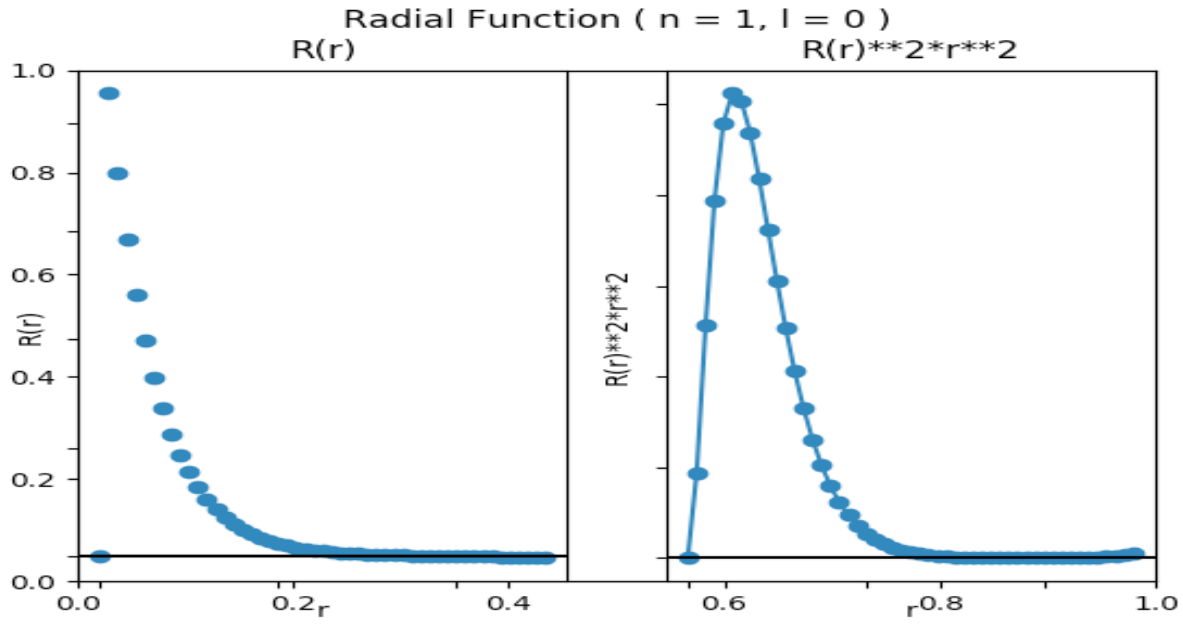
- Error #1: At n=1, we got the correct eigenvalue, but the radial wave function blows up at the end.

→Guess: due to the uniqueness of the shape of the function

→What if the box was smaller?

Application – #2. Hydrogen Atom

Result (n=1, smaller box)



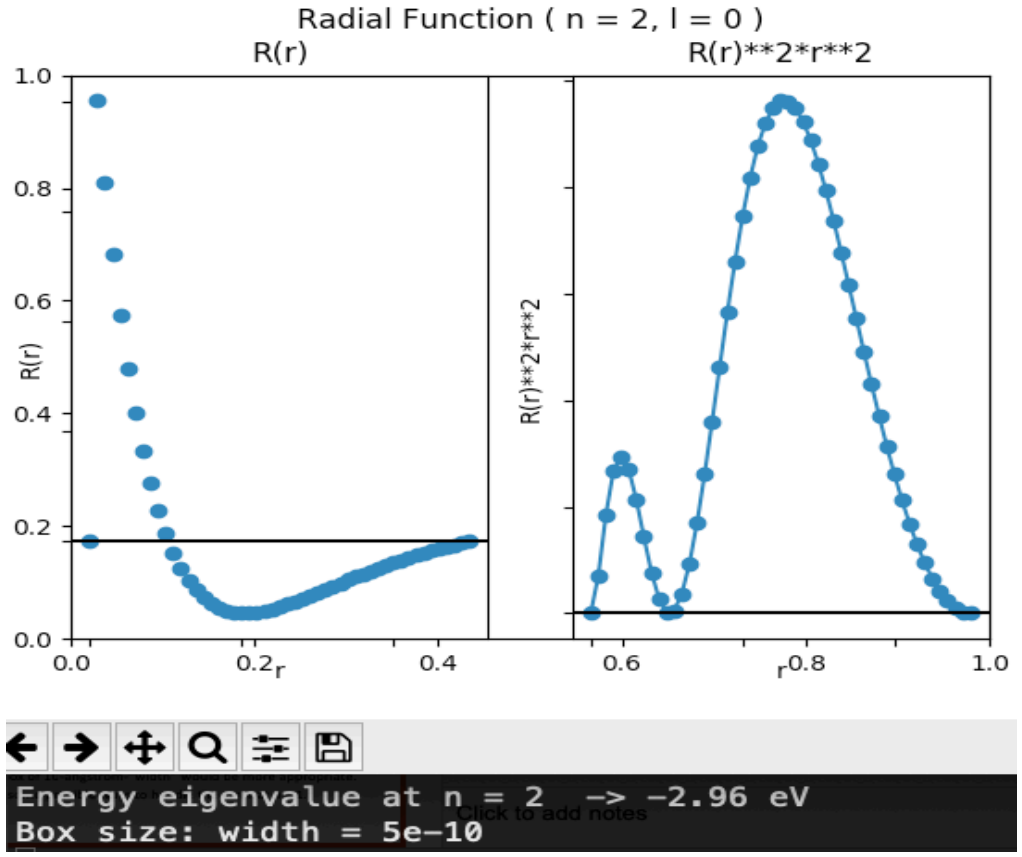

```
Energy eigenvalue at n = 1  -> -13.65 eV
Box size: width = 5e-10
```

→We got the same eigenvalue, and correct eigenfunction.

→But this box might be too small for other cases where n is larger than one.

Application – #2. Hydrogen Atom

Result (n=2, smaller box)



- Wrong eigenvalue

→ This box is too small!

→The original box of 10-angstrom-"width" would be more appropriate.

→We'll discuss how to handle this error at n = 1 later.

Application – #2. Hydrogen Atom

- Consider large n

Result (n=5, l=0)

```
Eigenvalue range  [-20, -13.65, -3.41, -1.3, 0.74, 3.68]
Energy eigenvalue at n = 5  -> 0.74 eV
Box size: width = 1e-09
```

→Error: Eigenenergy should be negative, but we got the positive result!

Application – #2. Hydrogen Atom

- Error #2: At n=5, eigenenergy should be negative, but we got the positive result.

→Guess: Difference between adjacent eigenvalues are too small, much smaller than integration step size h.

→We'll discuss how to handle this error later.

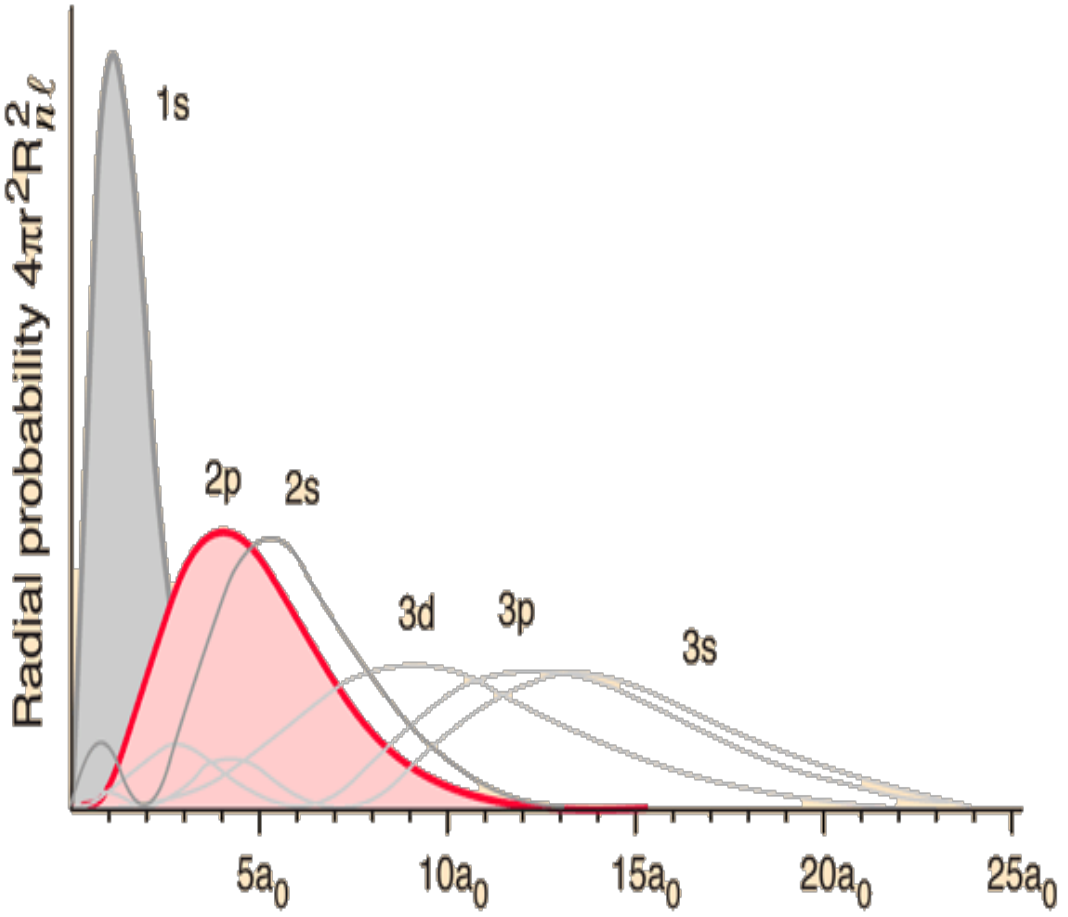
Application – #2. Hydrogen Atom

- Consider l = 1 (n=2)

→ 2p state

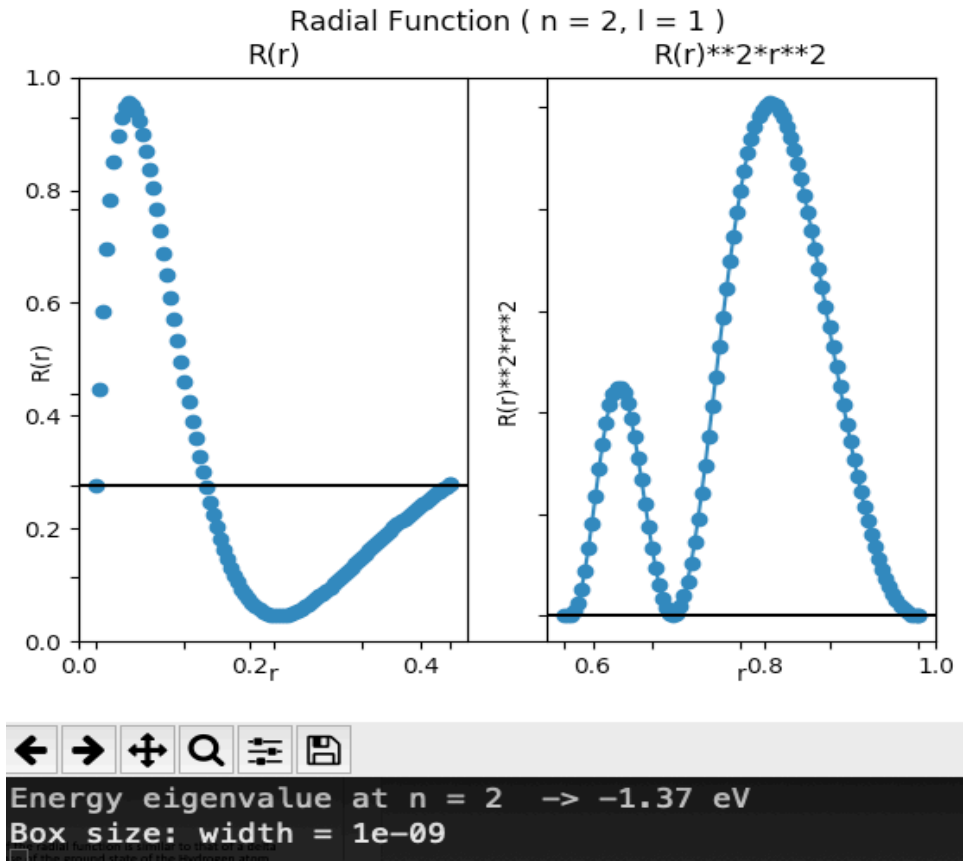
→ Consider the centrifugal term.

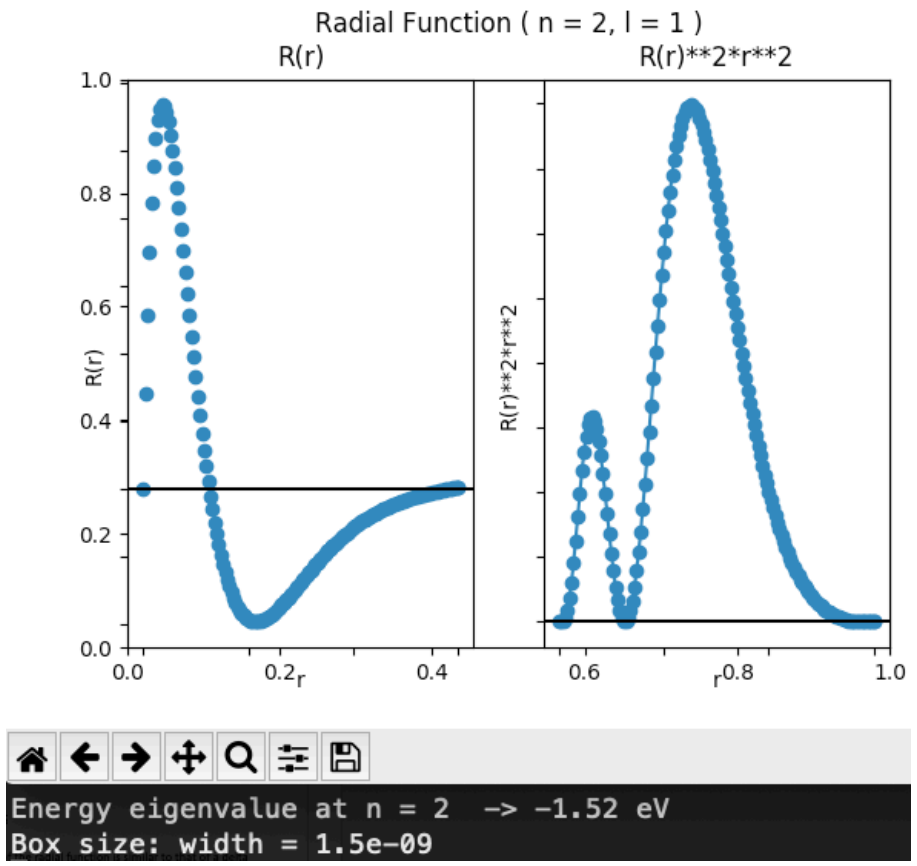
→ Analytically, eigenenergy should be the same as that of (l=0, n=2) case, about -3.42 eV.



Application – #2. Hydrogen Atom

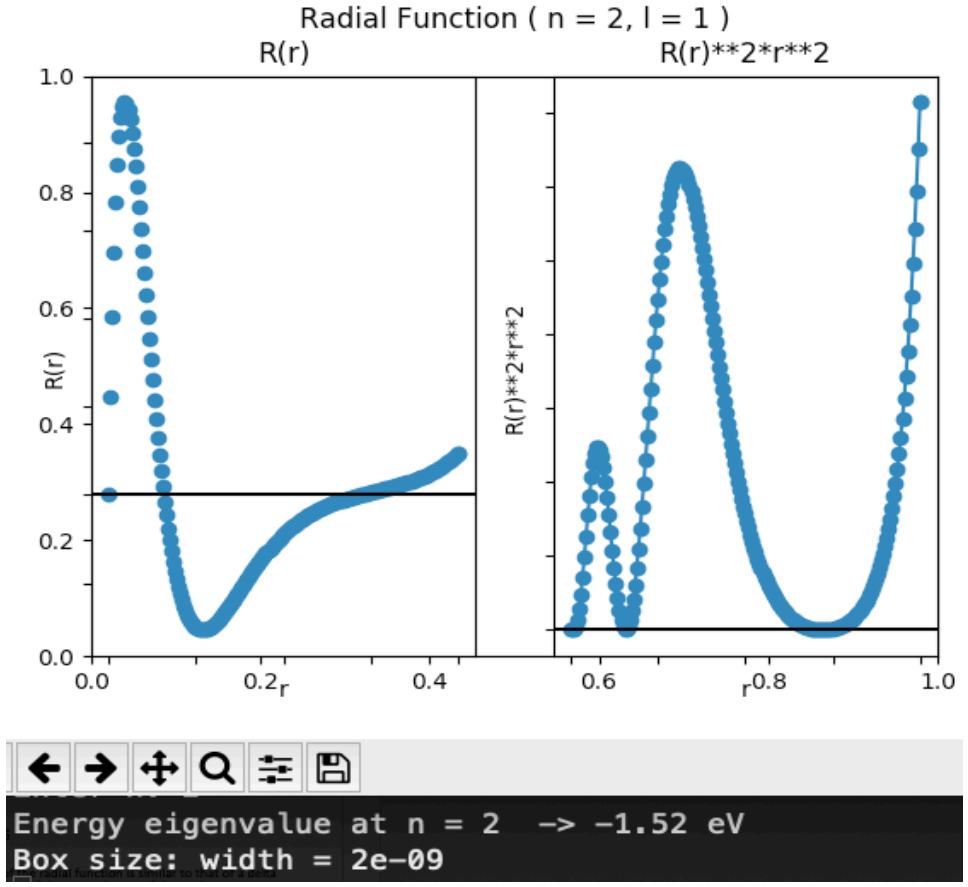
- Consider l = 1 (l=1, n=2)

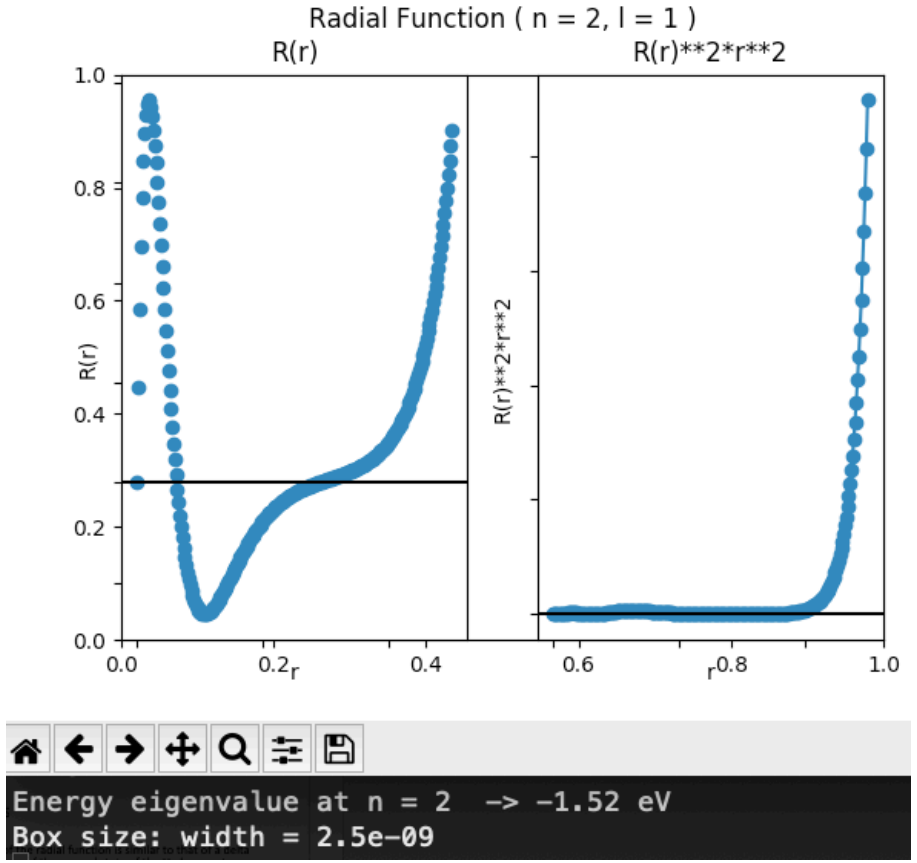




Application – #2. Hydrogen Atom

- Consider l = 1 (l=1, n=2)





Application – #2. Hydrogen Atom

- Error #3: when l = 1

→Regardless of the box size, we get the wrong eigenvalue and eigenfunction.

- For all cases where l is not zero.

→ The same error!

Error Analysis

1. When the shape of the radial function is similar to that of a delta function, as in the case of the ground state of the Hydrogen atom.
2. When E is negative and n gets large enough for the absolute value of eigenenergy to become very small, approximating to zero: the difference between adjacent eigenvalues are too small, much smaller than the integration step size h.
3. When l is not zero.

Error Analysis

1. When the shape of the radial function is similar to that of a delta function, as in the case of the ground state of the Hydrogen atom.

→If the radial function blows up at the end, make the box smaller until it satisfies the boundary condition.

→First, let's "avoid" the error in a quite pedestrian manner.

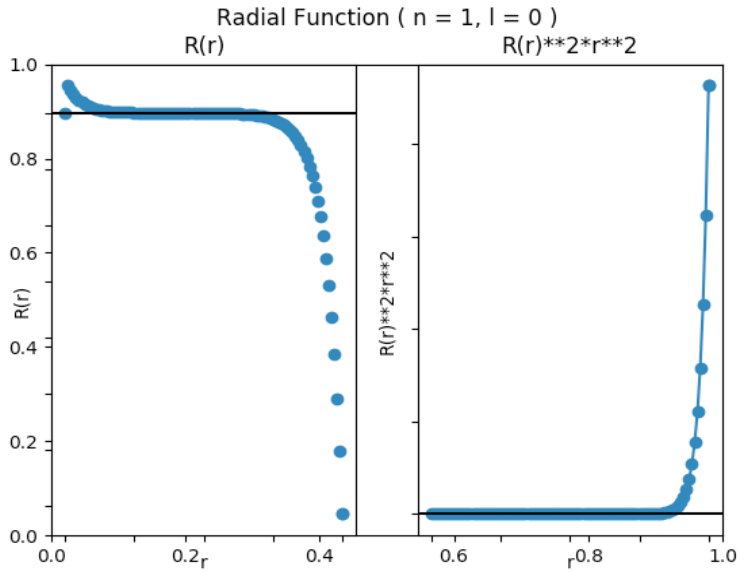
Error Analysis

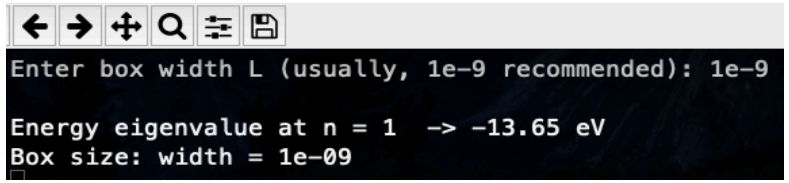
1. When the shape of the radial function is similar to that of a delta function, as in the case of the ground state of the Hydrogen atom.

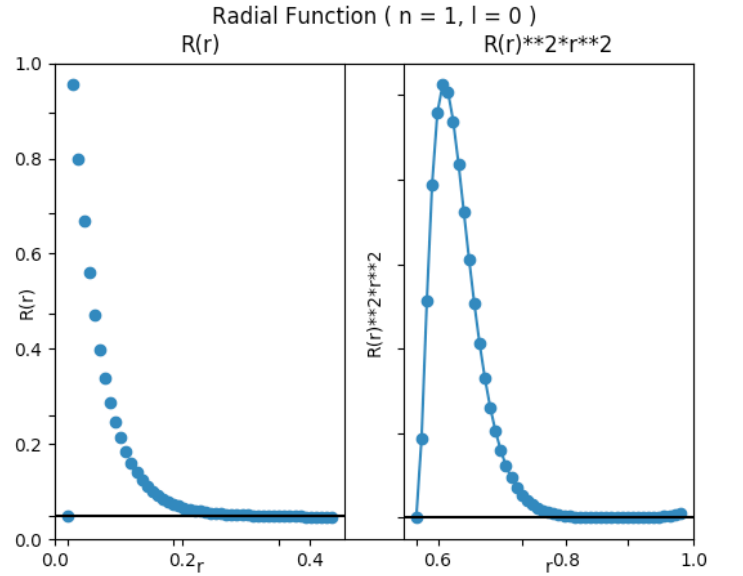
```
# hydrogen atom case
elif (choice == 2):
    # eigenenergy index (nth eigenenergy)
    n = int(input("Enter n: "))
    # width of the box
    xStop = float(input("Enter box width L (usually, 1e-9 recommended): "))
    # define potential
    def potentialVeff(r):
        vp = -q**2/(4*np.pi*permittivity*r+1e-50)
        return (vp+(C*l*(l+1)/(r**2+1e-50)))
```

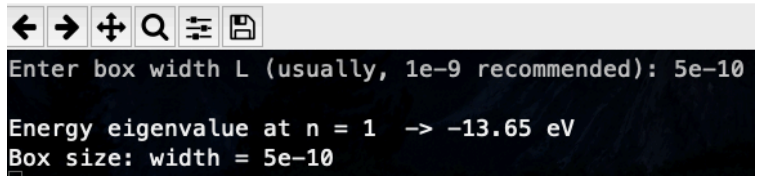
Error Analysis

1. When the shape of the radial function is similar to that of a delta function, as in the case of the ground state of the Hydrogen atom.









Error Analysis

2. The difference between adjacent eigenvalues are too small, much smaller than the integration step size h.

- If the sign of energy lower limit value is different from the current eigenvalue obtained, terminate the program and alert “Please enter a n value smaller than [current n value].”

```
    if ( np.sign(energyBoundary[n]) != np.sign(energyBoundary[0]) and np.sig
n(energyBoundary[0]) != 0 ):
        print("Please enter a n value smaller than",n)
    else:
        print("Energy eigenvalue at n =",n," ->",round(eigenvalue,2),"eV")
        print("Box size: width =",L)
```

Error Analysis

2. When E is negative and n gets large enough for the absolute value of eigenenergy to become very small, approximating to zero: the difference between adjacent eigenvalues are too small, much smaller than the integration step size h.

Result

```
Eigenvalue range  [-20, -13.65, -3.41, -1.3, 0.74, 3.68]
Please enter a n value smaller than 5
```

→ We can “avoid” the error!

Error Analysis

3. When l is not zero.

- Do not let the user choose the l value.
- Fix l value to zero in the code.

→ We can “avoid” the error!

Code Optimization – Things to Consider

1. How to “handle” the errors, not avoid it
2. How to find the energy lower bound value

Code Optimization – Things to Consider

1. How to “handle” the errors, not avoid it

→ Further discussion

2. How to find the energy lower bound value

→ Two possible ways

Code Optimization – Find the Lower Limit for E

- To find the lower limit for energy value from which we start the iteration.

- When $E > 0$, the lower limit should be zero.
- When $E < 0$, we do not know the lower limit yet.

→ First, find out whether the eigenenergies are positive or negative.

Code Optimization – Find the Lower Limit for E

- Find out whether the eigenenergies are positive or negative.

→ Use “try/except” exception handling technique.

```
try:
    # lower limit for E = 0
except:
```

→If energy is positive, the program will run the “try” part and end.

→Otherwise, the program will run the “except” part.

Code Optimization – Find the Lower Limit for E

try: # lower limit for E = 0

except: # ???

- I’ve come up with two possible ways to handle it.

1) Integrate backwards.
2) Looping the try/except part, *re*try with a lower limit E value; if the retry fails, *re*-retry with a much lower limit E value. Loop until the retry succeeds.

Code Optimization – Find the Lower Limit for E

1) Integrate backwards.

- The smaller the first eigenenergy gets, more memory-consuming it would be; even for the hydrogen atom (E1 = -13.6eV), a lot of iteration is required. → ineffective!

Code Optimization – Find the Lower Limit for E

2) Looping the try/except part, *re*try with a lower limit E value; if the retry fails, re-retry with a much lower limit E value. Loop until the retry succeeds.

- Every time “try” fails, retry with a 10eV-lower E limit value.
- For the hydrogen atom, after looping two times, we can get the answer.

Code Optimization – Find the Lower Limit for E

2) Looping the try/except part, *re*try with a lower limit E value; if the retry fails, re-retry with a much lower limit E value. Loop until the retry succeeds.

```
i = 0
while True:
    try:
        calculate(-10*i)
    except:
        i += 1
        continue
    else:
        break
```

Code for Accurate Results

1) Let l = 0

2) E > 0

- When the shape of the eigenfunction is similar to that of a delta function
- Correct eigenvalue / Wrong eigenfunction(it blows up at the end)

→ User runs the program again, entering a smaller box size

3) E < 0

- When the eigenenergy is too close to zero
- Correct eigenfunction / Wrong eigenvalue(it becomes positive)

→ Terminated & alert “Please enter a n value smaller than [current n]”

Code Implementation - Details

- User Input

1) Potential **V(r)**:

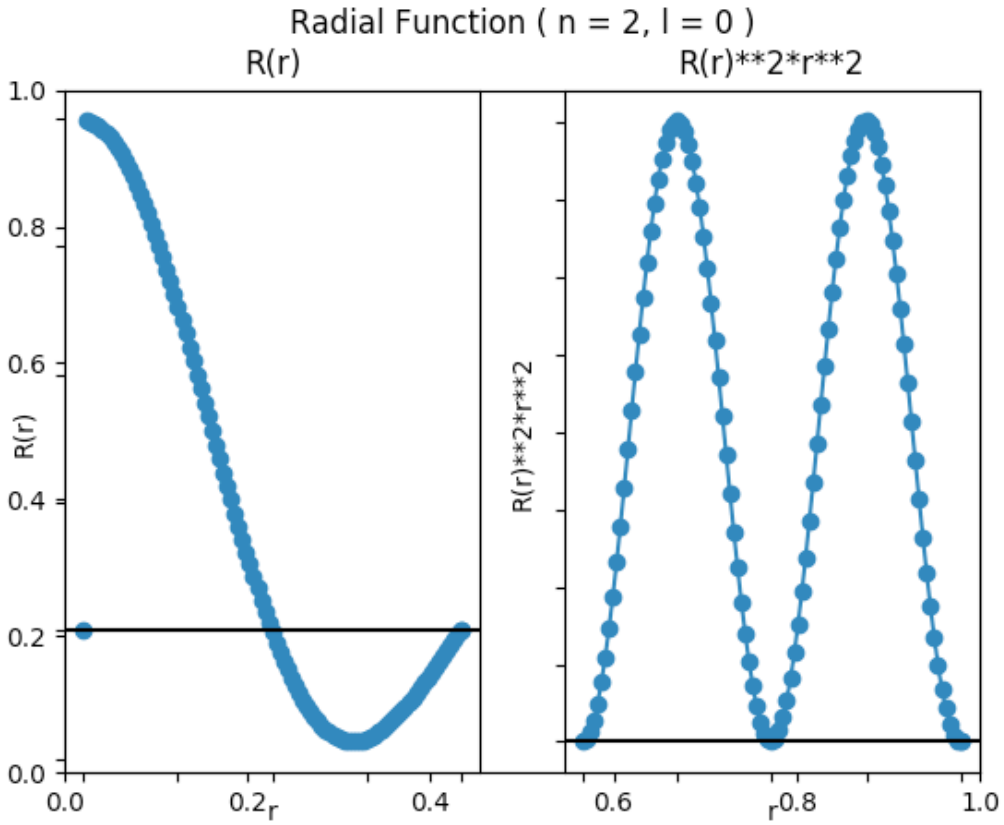
- Option 1. Infinite Well
- Option 2. Hydrogen Atom
- Option 3. Random Input

2) Eigenvalue Index **n**

3) Box size

Final Code – Infinite Well (n=2, l=0)

Result



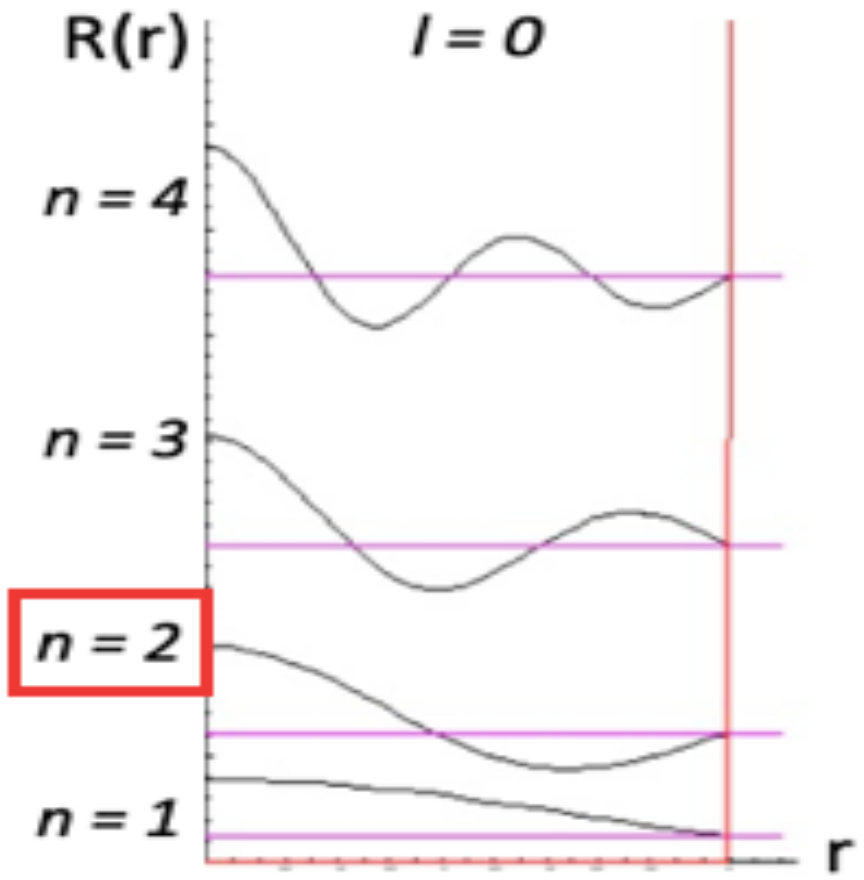

```
Central Potential ( l = 0 )
1. Infinite Potential Well
2. Hydrogen Atom
3. User Input Potential
Choose between the three options above - enter the nu
mber of your choice: 1
Enter n: 2
Enter box width L (usually, 1e-9 recommended): 1e-9

Energy eigenvalue at n = 2  -> 1.49 eV
Box size: width = 1e-09
```

Final Code – Infinite Well (n=2, l=0)

Analytic Solution

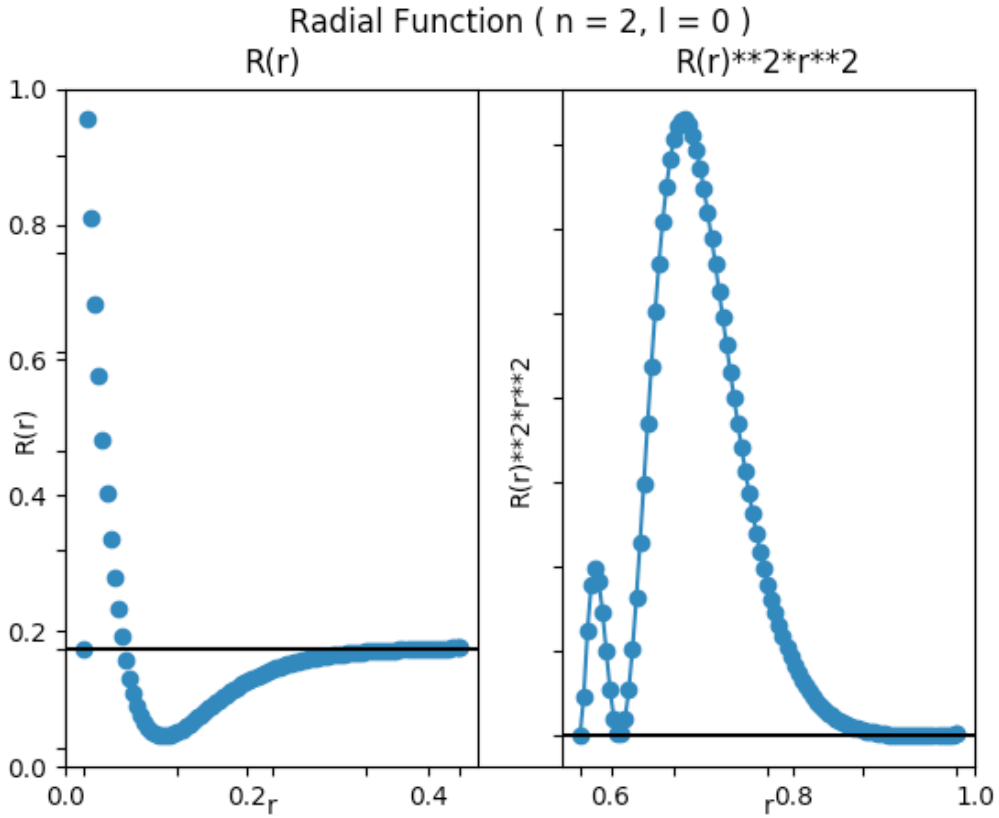
$$E_n = \frac{n^2 \pi^2 \hbar^2}{2mL^2} = \frac{n^2 h^2}{8mL^2}$$

$E_2 = 1.4913$ eV



Final Code – Hydrogen atom (n=2, l=0)

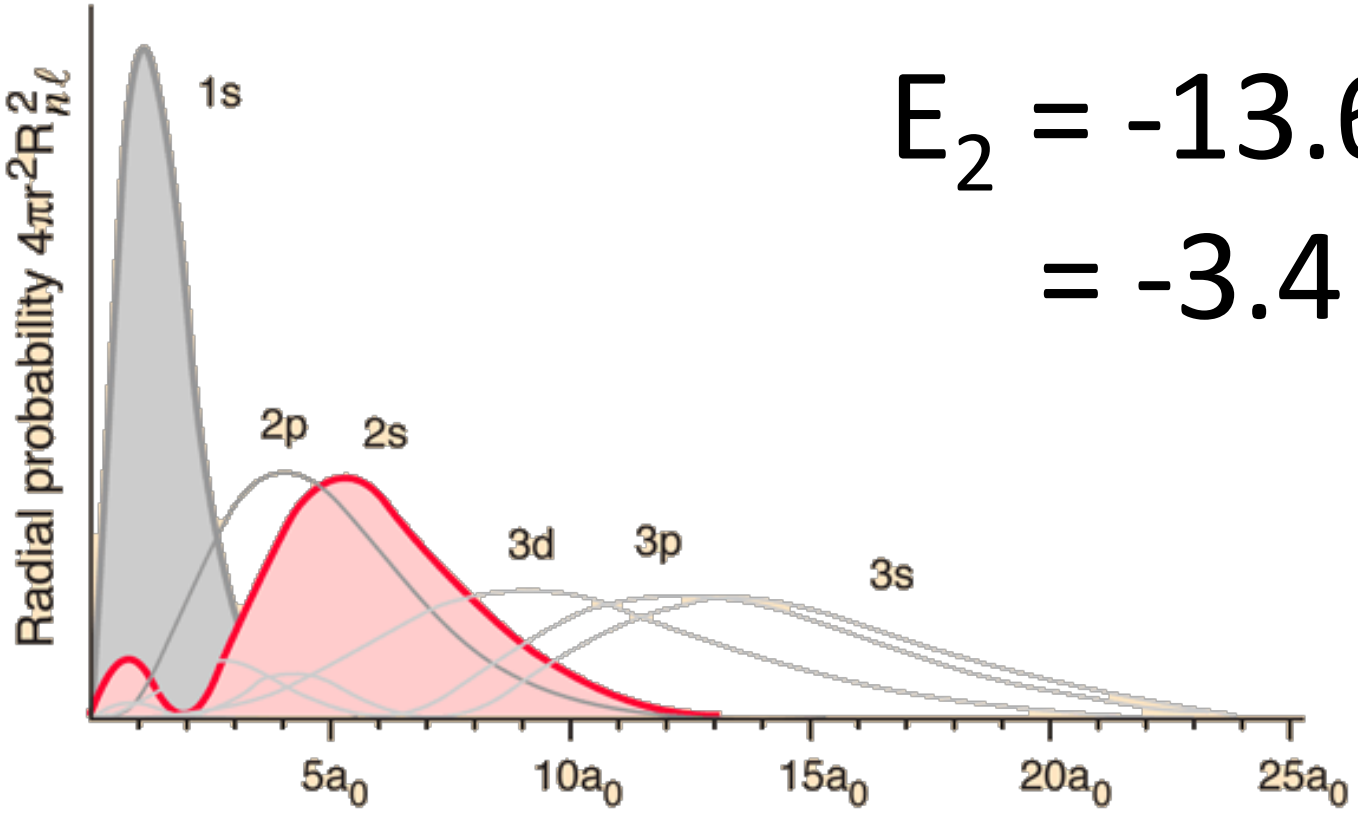
Result




```
Central Potential ( l = 0 )
1. Infinite Potential Well
2. Hydrogen Atom
3. User Input Potential
Choose between the three options above - enter the nu
mber of your choice: 2
Enter n: 2
Enter box width L (usually, 1e-9 recommended): 1e-9

Energy eigenvalue at n = 2  -> -3.42 eV
Box size: width = 1e-09
```

Final Code – Hydrogen atom (n=2, l=0)

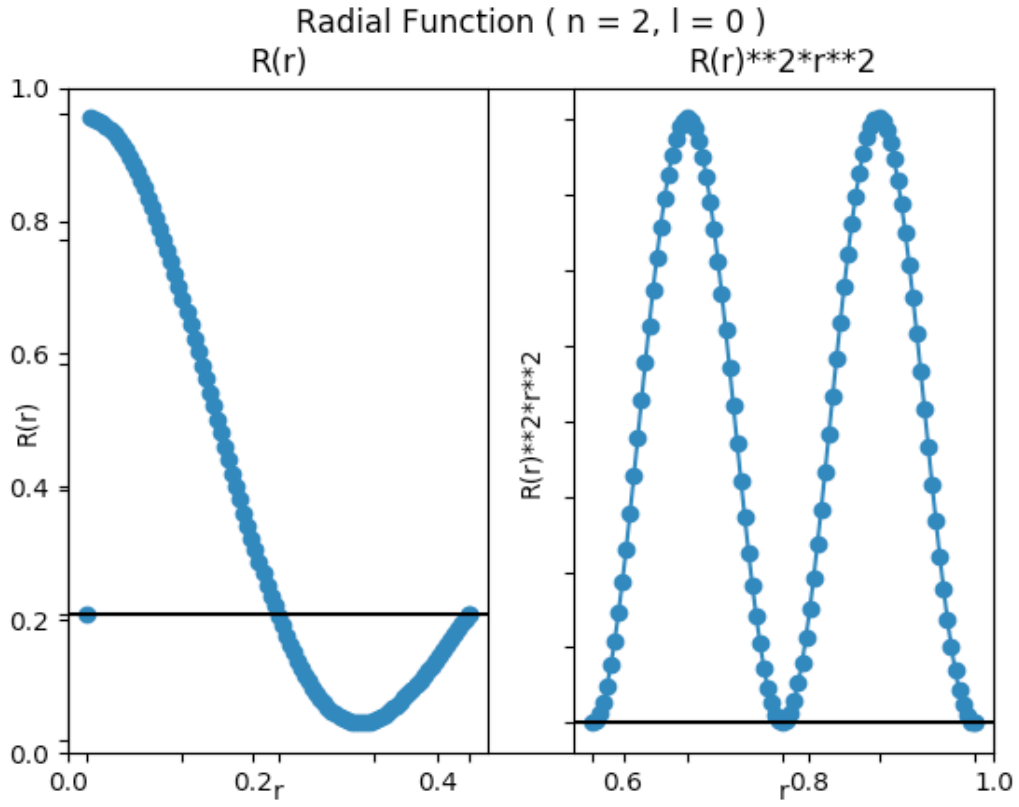
Analytic Solution



$$E_2 = -13.6/2^2 \text{ eV}$$
$$= -3.4 \text{ eV}$$

Final Code

- User input potential: V(r) = 1e-18 J = 6.24 eV




```
Central Potential ( l = 0 )
1. Infinite Potential Well
2. Hydrogen Atom
3. User Input Potential
Choose between the three options above - enter the nu
mber of your choice: 3
Enter n: 2
Enter box width L (usually, 1e-9 recommended): 1e-9
Enter V(r) (J*s): 1e-18

Energy eigenvalue at n = 2  -> 7.73 eV
Box size: width = 1e-09
```

Further Discussion

- Errors that need to be handled, not ”avoided”

1. When the shape of the radial function is similar to that of a delta function , as in the case of the ground state of the Hydrogen atom.

2. When E is negative and n gets large enough for the absolute value of eigenenergy to become very small, approximating to zero.

3. When l is not zero.

References

1) (n.d.). Retrieved June 9, 2019, from http://www.physics.csbsju.edu/QM/square.14.html
2) Hydrogen 1s Radial Probability. (n.d.). Retrieved June 9, 2019, from http://hyperphysics.phy-astr.gsu.edu/hbase/hydwf.html