Solving Schrodinger's Equation using Shooting method

Problem 1

```
# Solving Hydrogen Atom with Python

# Library imports
from scipy import *
from scipy import integrate
from scipy import optimize
from matplotlib import pyplot as plt
import numpy as np
import time

# Modifies print function to write in console and file simultaneously
def fprint(data):
    filename = "HydrogenShooting.out"
    with open(filename, "a") as f:
        f.write(data + "\n")
        print(data)


# Schrodinger equation as a function
def Schroed_deriv(y, r, l, En):
    "Given y=[u,u'] returns dy/dr=[u',u'']"
    (u, up) = y
    return np.array([up, (l * (l + 1) / r ** 2 - 2 / r - En) * u])

# Subroutine to implement shooting method
def Shoot(En, R, l):
    Rb = R[::-1]
    du0 = -1e-5
    ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En))
    ur = ub[:, 0][::-1]
    norm = integrate.simps(ur ** 2, x=R)
    ur *= 1.0 / np.lib.scimath.sqrt(norm)

    ur = ur / R ** l

    f0 = ur[0]
    f1 = ur[1]
    f_at_0 = f0 + (f1 - f0) * (0.0 - R[0]) / (R[1] - R[0])
    return f_at_0

# Subroutine to find the Bound state using shooting method
def FindBoundStates(R, l, nmax, Esearch):
    n = 0
    Ebnd = []
    u0 = Shoot(Esearch[0], R, l)
    for i in range(1, len(Esearch)):
        u1 = Shoot(Esearch[i], R, l)
        if u0 * u1 < 0:
            Ebound = optimize.brentq(
                Shoot, Esearch[i - 1], Esearch[i], xtol=1e-16, args=(R, l)
            )
            Ebnd.append((n+1, l, Ebound))
            if len(Ebnd) > nmax:
                break
            n += 1
            fprint(
                "Found bound state at E=%14.9f E_exact=%14.9f l=%d n=%d"
                % (Ebound, -1.0 / (n + l) ** 2, l, n)
            )
        u0 = u1

    return Ebnd

# Starting of calculations
fprint("Starting Calculations " + time.asctime() + "\n")
```

```
t0 = time.time()
Esearch = -1.2 / np.arange(1, 20, 0.2) ** 2

R = np.logspace(-6, 2, 500)

nmax = 7
Bnd = []
for l in range(nmax - 1):
    Bnd += FindBoundStates(R, l, nmax - l, Esearch)


Bnd.sort(key=lambda x: x[2])
Rb = R[::-1]
du0 = -1e-5
pqnum = [1, 2, 3]
azimqnum = ["s", "p", "d"]
fig, ax = plt.subplots()  # Create a figure and an axes.

ax.set_xlabel("r")  # Add an x-label to the axes.
ax.set_ylabel("u(r)")  # Add a y-label to the axes.
ax.set_title("Eigenvalue Plot")  # Add a title to the axes.

for n, l, En in Bnd:
    if l < 3 and 4 > n > l:
        fprint("----------------------------------------------------------------------")
        fprint("Ploting for " + str(n) + azimqnum[l] + " with Energy = " + str(En))
        ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En))
        ur = ub[:, 0][::-1]
        norm = integrate.simps(ur ** 2, x=R)
        ur *= 1.0 / np.lib.scimath.sqrt(norm)
        ax.plot(R, ur, label=str(n) + azimqnum[l])
ax.legend()
plt.savefig("HydrogenatonEigenvalues.png")
fprint("**************************************")

fprint("Calculations are done." + time.asctime())
fprint("Time taken for calculations " + str(time.time()-t0) + " secs")
```

Result

Starting Calculations Thu Oct 28 14:29:37 2021

Found bound state at E= -1.000000033 E_exact= -1.000000000 l=0 n=1
Found bound state at E= -0.250000000 E_exact= -0.250000000 l=0 n=2
Found bound state at E= -0.111111111 E_exact= -0.111111111 l=0 n=3
Found bound state at E= -0.062500002 E_exact= -0.062500000 l=0 n=4
Found bound state at E= -0.039999836 E_exact= -0.040000000 l=0 n=5
Found bound state at E= -0.027733156 E_exact= -0.027777778 l=0 n=6
Found bound state at E= -0.019181545 E_exact= -0.020408163 l=0 n=7
Found bound state at E= -0.249999997 E_exact= -0.250000000 l=1 n=1
Found bound state at E= -0.111111111 E_exact= -0.111111111 l=1 n=2
Found bound state at E= -0.062500002 E_exact= -0.062500000 l=1 n=3
Found bound state at E= -0.039999960 E_exact= -0.040000000 l=1 n=4
Found bound state at E= -0.027742609 E_exact= -0.027777778 l=1 n=5
Found bound state at E= -0.019305078 E_exact= -0.020408163 l=1 n=6
Found bound state at E= -0.111111114 E_exact= -0.111111111 l=2 n=1
Found bound state at E= -0.062500003 E_exact= -0.062500000 l=2 n=2
Found bound state at E= -0.040000107 E_exact= -0.040000000 l=2 n=3
Found bound state at E= -0.027755899 E_exact= -0.027777778 l=2 n=4
Found bound state at E= -0.019509891 E_exact= -0.020408163 l=2 n=5
Found bound state at E= -0.062500000 E_exact= -0.062500000 l=3 n=1
Found bound state at E= -0.040000009 E_exact= -0.040000000 l=3 n=2
Found bound state at E= -0.027765346 E_exact= -0.027777778 l=3 n=3
Found bound state at E= -0.019768987 E_exact= -0.020408163 l=3 n=4

Found bound state at E= -0.040000010 E_exact= -0.040000000 l=4 n=1
Found bound state at E= -0.027772890 E_exact= -0.027777778 l=4 n=2
Found bound state at E= -0.020029141 E_exact= -0.020408163 l=4 n=3
Found bound state at E= -0.027776892 E_exact= -0.027777778 l=5 n=1
Found bound state at E= -0.020239681 E_exact= -0.020408163 l=5 n=2

Plotting for 1s with Energy = -1.000000033034256

Plotting for 2s with Energy = -0.24999999959366614

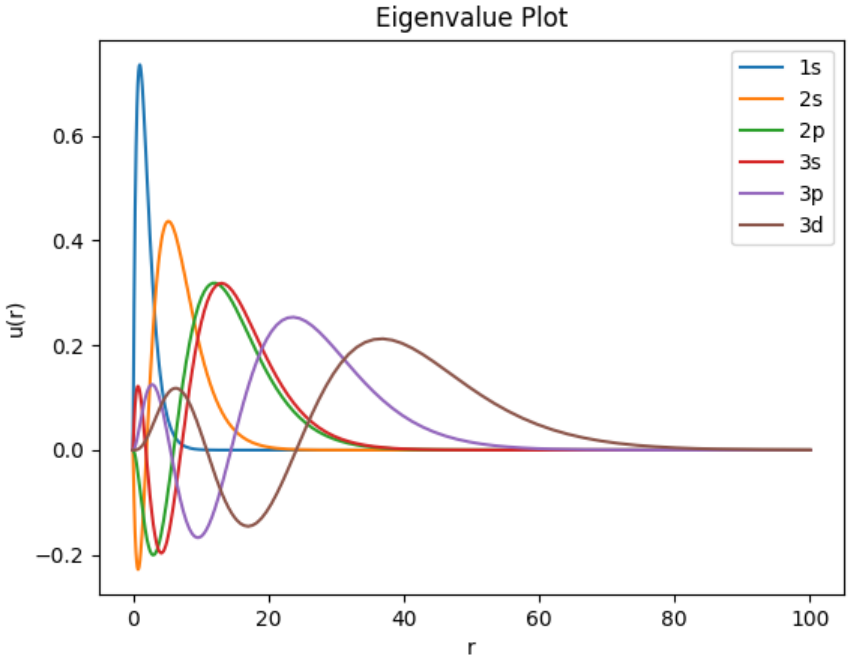
Plotting for 2p with Energy = -0.11111111111454078

Plotting for 3s with Energy = -0.11111111088115905

Plotting for 3p with Energy = -0.06250000214456702

Plotting for 3d with Energy = -0.04000010720947516

Calculations are done. Thu Oct 28 14:29:43 2021
Time taken for calculations 6.419199466705322 secs



Problem 2

```
# Solving Hydrogen Atom with Python

# Library imports
from scipy import *
from scipy import integrate
from scipy import optimize
from matplotlib import pyplot as plt
import numpy as np
import time

# Modifies print function to write in console and file simultaneously
def fprint(data):
    filename = "HydrogenShootingZvar.out"
    with open(filename, "a") as f:
        f.write(data + "\n")
        print(data)
```

```
# Schrodinger equation as a function
def Schroed_deriv(y, r, l, En, Z):
    "Given y=[u,u'] returns dy/dr=[u',u'']"
    (u, up) = y
    return np.array([up, (l * (l + 1) / r ** 2 - 2 * Z / r - En) * u])


# Subroutine to implement shooting method
def Shoot(En, R, l, Z):
    Rb = R[::-1]
    du0 = -1e-5
    ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En, Z))
    ur = ub[:, 0][::-1]
    norm = integrate.simps(ur ** 2, x=R)
    ur *= 1.0 / np.lib.scimath.sqrt(norm)

    ur = ur / R ** l

    f0 = ur[0]
    f1 = ur[1]
    f_at_0 = f0 + (f1 - f0) * (0.0 - R[0]) / (R[1] - R[0])
    return f_at_0


# Subroutine to find the Bound state using shooting method
def FindBoundStates(R, l, nmax, Esearch, Z):
    n = 0
    Ebnd = []
    u0 = Shoot(Esearch[0], R, l, Z)
    for i in range(1, len(Esearch)):
        u1 = Shoot(Esearch[i], R, l, Z)
        if u0 * u1 < 0:
            Ebound = optimize.brentq(
                Shoot, Esearch[i - 1], Esearch[i], xtol=1e-16, args=(R, l, Z)
            )
            Ebnd.append((n, l, Ebound))
            if len(Ebnd) > nmax:
                break
            n += 1
            print(
                "Found bound state at E=%14.9f l=%d n=%d"
                % (Ebound, l, n)
            )
        u0 = u1

    return Ebnd


# Starting of calculations

fprint("Starting Calculations " + time.asctime() + "\n")
t0 = time.time()
Esearch = -1.2 / np.arange(1, 20, 0.2) ** 2

R = np.logspace(-6, 1.5, 500)
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.set_xlabel("r")  # Add an x-label to the axes.
ax.set_ylabel("u(r)")  # Add a y-label to the axes.
ax.set_title("Eigenvalue Plot")  # Add a title to the axes.
for Z in [1, 2, 4]:
    nmax = 2
    Bnd = []
    for l in range(nmax - 1):
        Bnd += FindBoundStates(R, l, nmax - l, Esearch, Z)
    Rb = R[::-1]
    du0 = -1e-5
    pqnum = [1, 2, 3]
    azimqnum = ["s", "p", "d"]

    for n, l, En in Bnd:
        if n == 1:
            fprint("-----------------------------------------------------------------")
            fprint("Ploting for " + str(n) + azimqnum[l] + " with Energy = " + str(En) + "for Z= " + str(Z))
            ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En, Z))
            ur = ub[:, 0][::-1]
            norm = integrate.simps(ur ** 2, x=R)
```

```
            ur *= 1.0 / np.lib.scimath.sqrt(norm)
            ax.plot(R, ur, label="Z=" + str(Z) + " " + str(n) + azimqnum[l])
ax.legend()
plt.savefig("HydrogenatonEigenvaluesZvar.png")
fprint("****************************************")

fprint("Calculations are done." + time.asctime())
fprint("Time taken for calculations " + str(time.time() - t0) + " secs")
```

Results

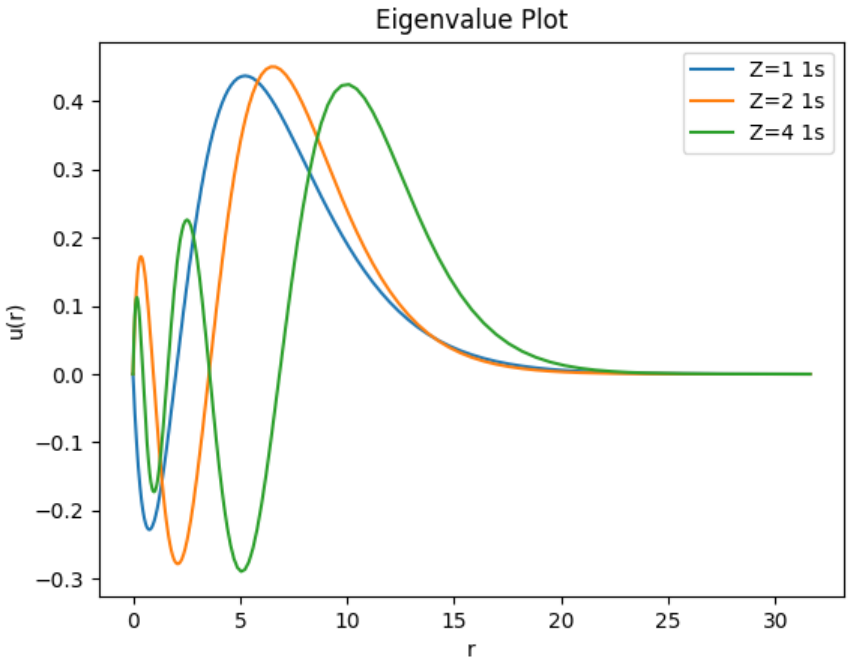
Starting Calculations Thu Oct 28 14:35:23 2021

Plotting for 1s with Energy = -0.24999960714225003for Z= 1

Plotting for 1s with Energy = -0.44444438744653for Z= 2

Plotting for 1s with Energy = -0.6400002235372901for Z= 4

Calculations are done. Thu Oct 28 14:35:23 2021
Time taken for calculations 0.46553826332092285 secs



Problem 3a

```
# Solving Hydrogen Atom with Python

# Library imports
from scipy import *
from scipy import integrate
from scipy import optimize
from matplotlib import pyplot as plt
import numpy as np
import time

def fprint(data):
    filename = "HydrogenSCP.out"
    with open(filename, "a") as f:
        f.write(data + "\n")
        print(data)
```

```
# Schrodinger equation as a function
def Schroed_deriv(y, r, l, En, lamda):
    "Given y=[u,u'] returns dy/dr=[u',u'']"
    (u, up) = y
    return np.array([up, (l * (l + 1) / r ** 2 - 2*np.exp(-r/lamda) / r - En) * u])


def Shoot(En, R, l, lamda):
    Rb = R[::-1]
    du0 = -1e-5
    ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En,lamda))
    ur = ub[:, 0][::-1]
    norm = integrate.simps(ur ** 2, x=R)
    ur *= 1.0 / np.lib.scimath.sqrt(norm)

    ur = ur / R ** l

    f0 = ur[0]
    f1 = ur[1]
    f_at_0 = f0 + (f1 - f0) * (0.0 - R[0]) / (R[1] - R[0])
    return f_at_0


def FindBoundStates(R, l, nmax, Esearch, lamda):
    n = 0
    Ebnd = []
    u0 = Shoot(Esearch[0], R, l,lamda)
    for i in range(1, len(Esearch)):
        u1 = Shoot(Esearch[i], R, l,lamda)
        if u0 * u1 < 0:
            Ebound = optimize.brentq(
                Shoot, Esearch[i - 1], Esearch[i], xtol=1e-16, args=(R, l,lamda)
            )
            Ebnd.append((n+1, l, Ebound))
            if len(Ebnd) > nmax:
                break
            n += 1
            fprint(
                "Found bound state at E=%14.9f l=%d n=%d"
                % (Ebound, l, n)
            )

        u0 = u1

    return Ebnd
fprint("Starting Calculations " + time.asctime() + "\n")
t0 = time.time()
for lamda in [10,20,50]:
    Esearch = -1.2 / np.arange(1, 20, 0.2) ** 2

    R = np.logspace(-6, 2, 500)

    nmax = 7
    Bnd = []
    for l in range(nmax - 1):
        Bnd += FindBoundStates(R, l, nmax - l, Esearch,lamda)


    Bnd.sort(key=lambda x: x[2])
    Rb = R[::-1]
    du0 = -1e-5
    pqnum = [1, 2, 3]
    azimqnum = ["s", "p", "d"]
    fig, ax = plt.subplots()  # Create a figure and an axes.

    ax.set_xlabel("r")  # Add an x-label to the axes.
    ax.set_ylabel("u(r)")  # Add a y-label to the axes.
    ax.set_title("Eigenvalue Plot")  # Add a title to the axes.

    for n, l, En in Bnd:
        if l == 0 and 0<n<4:
            fprint("----------------------------------------------------------------")
            fprint("Ploting for " + str(n) + azimqnum[l] + " with Energy = " + str(En) + " and lambda = " + str(lamda))
            ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En,lamda))
            ur = ub[:, 0][::-1]
            norm = integrate.simps(ur ** 2, x=R)
```

```
            ur *= 1.0 / np.lib.scimath.sqrt(norm)
            ax.plot(R, ur, label=str(n) + azimqnum[l])
    ax.legend()
    plt.savefig("HydrogenatonEigenvalueslambda=" + str(lamda) + ".png")
    fprint("****************************************")

fprint("Calculations are done." + time.asctime())
fprint("Time taken for calculations " + str(time.time()-t0) + " secs")
```

Results

Starting Calculations Thu Oct 28 14:29:15 2021

Found bound state at E= -0.814116090 l=0 n=1
Found bound state at E= -0.099856540 l=0 n=2
Found bound state at E= -0.006415762 l=0 n=3
Found bound state at E= -0.093068778 l=1 n=1
Found bound state at E= -0.003176651 l=1 n=2

Plotting for 1s with Energy = -0.8141160902058124 and lambda = 10

Plotting for 2s with Energy = -0.09985653957265886 and lambda = 10

Plotting for 3s with Energy = -0.006415762150970595 and lambda = 10

Found bound state at E= -0.903632873 l=0 n=1
Found bound state at E= -0.163542389 l=0 n=2
Found bound state at E= -0.038705110 l=0 n=3
Found bound state at E= -0.006182045 l=0 n=4
Found bound state at E= -0.161480770 l=1 n=1
Found bound state at E= -0.037115504 l=1 n=2
Found bound state at E= -0.005194530 l=1 n=3
Found bound state at E= -0.033831141 l=2 n=1
Found bound state at E= -0.003159253 l=2 n=2

Plotting for 1s with Energy = -0.90363287264789 and lambda = 20

Plotting for 2s with Energy = -0.16354238872216653 and lambda = 20

Plotting for 3s with Energy = -0.03870510974386353 and lambda = 20

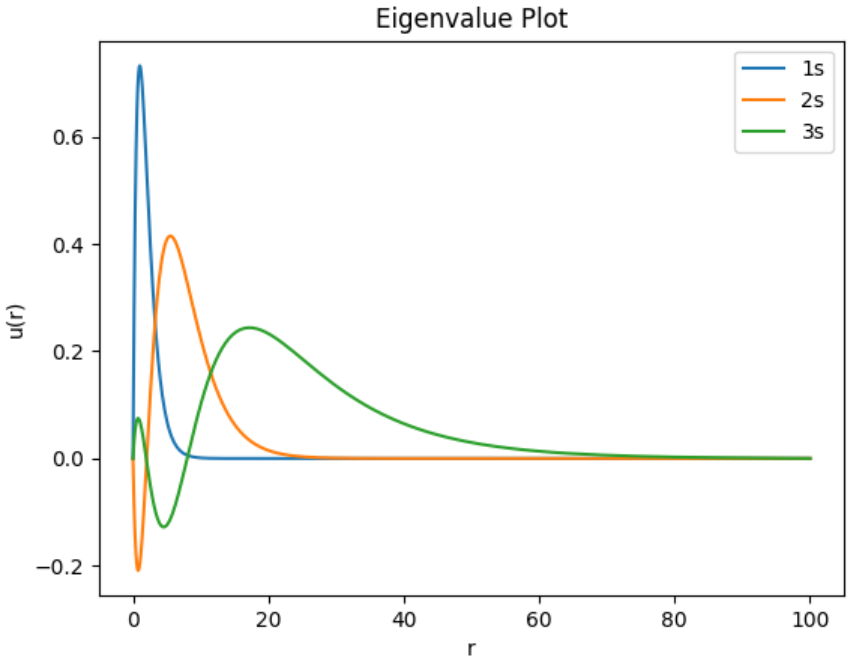
Found bound state at E= -0.960592228 l=0 n=1
Found bound state at E= -0.212296640 l=0 n=2
Found bound state at E= -0.076040030 l=0 n=3
Found bound state at E= -0.030758534 l=0 n=4
Found bound state at E= -0.012057274 l=0 n=5
Found bound state at E= -0.003393187 l=0 n=6
Found bound state at E= -0.211926793 l=1 n=1
Found bound state at E= -0.075704779 l=1 n=2
Found bound state at E= -0.030467616 l=1 n=3
Found bound state at E= -0.011819110 l=1 n=4
Found bound state at E= -0.003245691 l=1 n=5
Found bound state at E= -0.075030257 l=2 n=1
Found bound state at E= -0.029880120 l=2 n=2
Found bound state at E= -0.011336535 l=2 n=3
Found bound state at E= -0.028983959 l=3 n=1
Found bound state at E= -0.010595101 l=3 n=2
Found bound state at E= -0.009569763 l=4 n=1

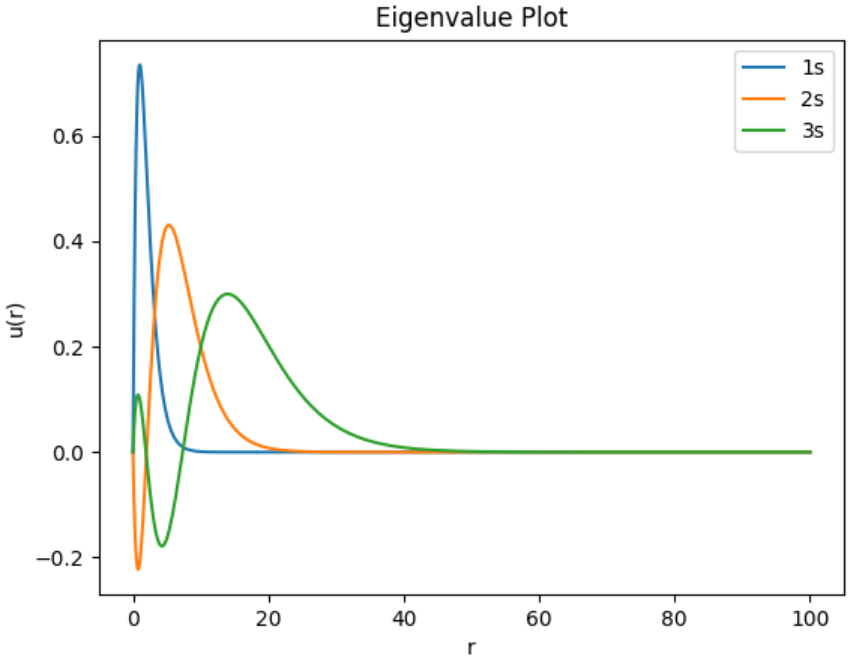
Plotting for 1s with Energy = -0.9605922279015445 and lambda = 50

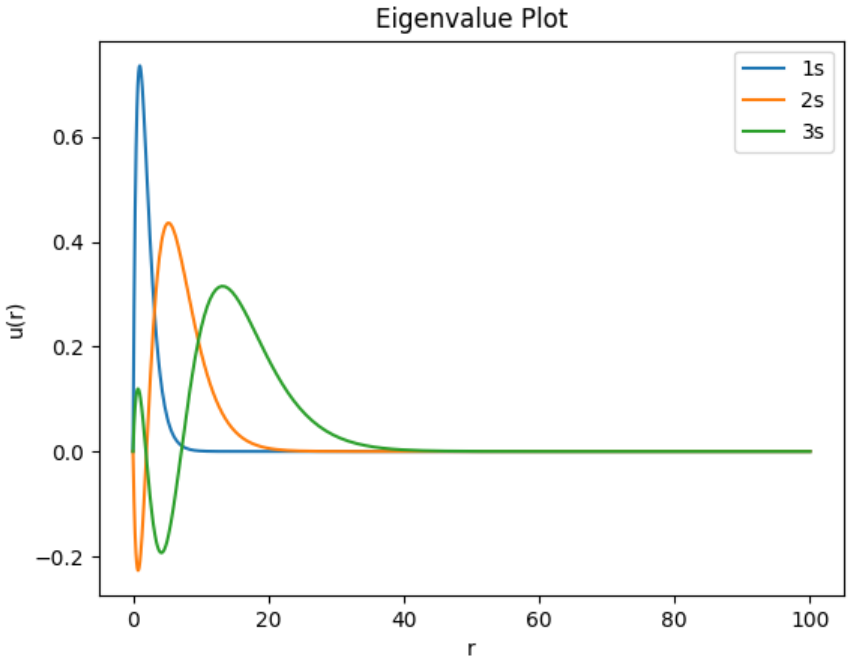
Plotting for 2s with Energy = -0.21229663955936776 and lambda = 50

Plotting for 3s with Energy = -0.0760400303707548 and lambda = 50

Calculations are done. Thu Oct 28 14:29:33 2021
Time taken for calculations 17.99623417854309 secs







Problem 3b

```
# Solving Hydrogen Atom with Python

# Library imports
from scipy import *
from scipy import integrate
from scipy import optimize
from matplotlib import pyplot as plt
import numpy as np
import time

def fprint(data):
    filename = "HydrogenSCPlambda10so.out"
    with open(filename, "a") as f:
        f.write(data + "\n")
        print(data)


# Schrodinger equation as a function
def Schroed_deriv(y, r, l, En, lamda):
    "Given y=[u,u'] returns dy/dr=[u',u'']"
    (u, up) = y
    return np.array([up, (l * (l + 1) / r ** 2 - 2*np.exp(-r/lamda) / r - En) * u])


def Shoot(En, R, l, lamda):
    Rb = R[::-1]
    du0 = -1e-5
    ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En,lamda))
    ur = ub[:, 0][::-1]
    norm = integrate.simps(ur ** 2, x=R)
    ur *= 1.0 / np.lib.scimath.sqrt(norm)

    ur = ur / R ** l

    f0 = ur[0]
    f1 = ur[1]
    f_at_0 = f0 + (f1 - f0) * (0.0 - R[0]) / (R[1] - R[0])
    return f_at_0


def FindBoundStates(R, l, nmax, Esearch, lamda):
    n = 0
```

```
    Ebnd = []
    u0 = Shoot(Esearch[0], R, l,lamda)
    for i in range(1, len(Esearch)):
        u1 = Shoot(Esearch[i], R, l,lamda)
        if u0 * u1 < 0:
            Ebound = optimize.brentq(
                Shoot, Esearch[i - 1], Esearch[i], xtol=1e-16, args=(R, l,lamda)
            )
            Ebnd.append((n+1, l, Ebound))
            if len(Ebnd) > nmax:
                break
            n += 1
            fprint(
                "Found bound state at E=%14.9f l=%d n=%d"
                % (Ebound, l, n)
            )

        u0 = u1

    return Ebnd
fprint("Starting Calculations " + time.asctime() + "\n")
t0 = time.time()
lamda = 10
Esearch = -1.2 / np.arange(1, 20, 0.2) ** 2
R = np.logspace(-6, 2, 500)
nmax = 15
Bnd = []
for l in range(nmax - 1):
    Bnd += FindBoundStates(R, l, nmax - l, Esearch,lamda)
Bnd.sort(key=lambda x: x[2])
print(Bnd)
Rb = R[::-1]
du0 = -1e-5
pqnum = [1, 2, 3]
azimqnum = ["s", "p", "d"]
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.set_xlabel("r")  # Add an x-label to the axes.
ax.set_ylabel("u(r)")  # Add a y-label to the axes.
ax.set_title("Eigenvalue Plot")  # Add a title to the axes.
for n, l, En in Bnd:
    if n == 3 and l<n:
        fprint("------------------------------------------------------------------------")
        fprint("Ploting for " + str(n) + azimqnum[l] + " with Energy = " + str(En) + " and lambda = " + str(lamda))
        ub = integrate.odeint(Schroed_deriv, [0.0, du0], Rb, args=(l, En,lamda))
        ur = ub[:, 0][::-1]
        norm = integrate.simps(ur ** 2, x=R)
        ur *= 1.0 / np.lib.scimath.sqrt(norm)
        ax.plot(R, ur, label=str(n) + azimqnum[l])
ax.legend()
plt.savefig("HydrogenatonEigenvalueslambda=10s" + ".png")
fprint("****************************************")
fprint("Calculations are done." + time.asctime())
fprint("Time taken for calculations " + str(time.time()-t0) + " secs")
```

Results

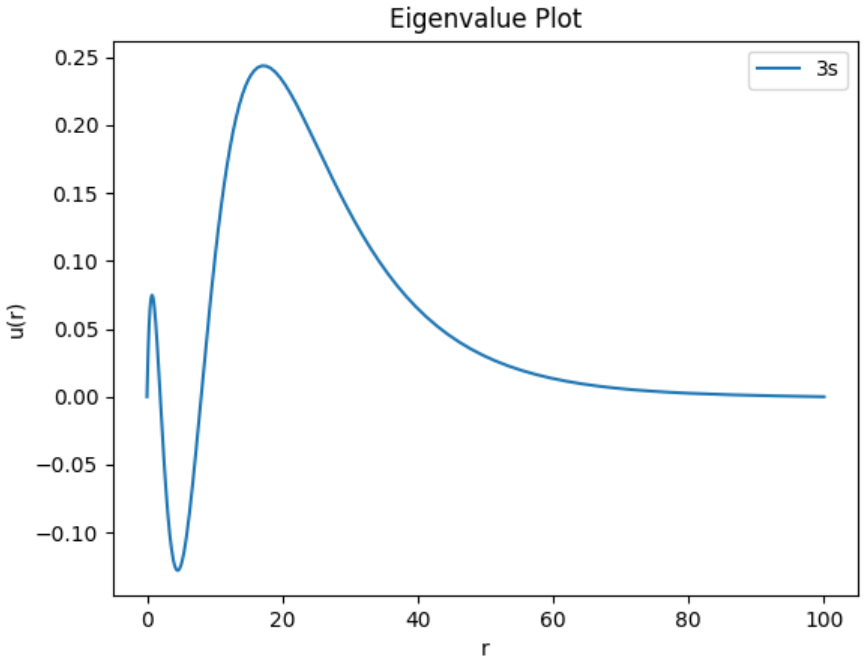
We didn't find any bound state for 3p and #d orbital when $\lambda = 10$

Starting Calculations Thu Oct 28 14:28:55 2021

Found bound state at E= -0.814116090 l=0 n=1
Found bound state at E= -0.099856540 l=0 n=2
Found bound state at E= -0.006415762 l=0 n=3
Found bound state at E= -0.093068778 l=1 n=1
Found bound state at E= -0.003176651 l=1 n=2

Plotting for 3s with Energy = -0.006415762150970595 and lambda = 10

Calculations are done. Thu Oct 28 14:29:11 2021
Time taken for calculations 15.991404056549072 secs


Eigenvalue Plot
3s
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
u(r)
0
20
40
60
80
100
r