Final Project Outline: Hartree Fock Method

Junlan Lu

1 Background

1.1 Problem statement

Fundamentally, the Hartree Fock method is a single electron approximation technique use to approximate the wave function and energy for a quantum many body system.[1] In general, the properties of molecules and intermolecular interactions can be solve for by solving for the Schrodinger equation given by

$$i\hbar\frac{d\Psi}{dt} = \hat{H}\Psi \qquad \text{where} \qquad \hat{H} = \frac{\hat{p}^2}{2m} + \hat{V} \tag{1}$$

By employing the Born-Oppenheimer approximation and the mean field approximation, which neglects the time-dependency of the nuclei-nuclei interactions and approximates the electron-electron interaction as an electron-mean field interaction respectively. The result for the time-independent electronic Schrodinger equation is that

$$\Bigg[\underbrace{-\frac{1}{2m_e}\sum_i^{N_e}\nabla_i^2}_{\hat{T}_e} + \underbrace{\sum_i^{N_p}\sum_j^{N_e}\frac{k_e Z_j e^2}{r_{ij}}}_{\hat{V}_{eN}} + \underbrace{\sum_{i>j}\frac{k_e Z_i Z_j e^2}{R_{ij}}}_{\hat{V}_{NN}} + \underbrace{\sum_{i>j}\frac{k_e e^2}{r_{ij}}}_{\hat{V}_{ee}}\Bigg]\Psi(\mathbf{r},\mathbf{R}) = E_{el}\Psi(\mathbf{r},\mathbf{R}) \tag{2}$$

Operating under the mean field theory, the function form for the wavefunction in Hartree-Fock theory will be a Slater determinant of spin orbitals.

$$\Psi = \frac{1}{\sqrt{N!}}\begin{vmatrix} \chi_1(x_1) & \chi_2(x_1) & \dots & \chi_{N_e}(x_1) \\ \chi_1(x_2) & \chi_2(x_2) & \dots & \chi_{N_e}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \chi_1(x_{N_e}) & \chi_2(x_{N_e}) & \dots & \chi_{N_e}(x_{N_e}) \end{vmatrix} \tag{3}$$

which implies that the *wavefunction can be approximated by a single Slater determinant made up of one spin orbital per electron.* The task at hand is to solve (2) to find the total ground state energy. In this project, we will solve the self-consistent field equations of the two-level system (only two-electrons), and compare it to experimental values.

1.2 Derivation of the two-electron system

Since we are only dealing with the Helium atom, there is only 1 nucleus and thus, the $\hat{V}_{NN}$ term disappears from (2). Then, the Hamiltonian for N electrons moving about 1 heavy nucleus of charge Z located at the origin can be written as

$$H = \sum_{i=1}^{N}\frac{p_i^2}{2m} - \sum_{i=1}^{N}\frac{Ze^2}{r_i} + \frac{1}{2}\sum_{i\neq j=1}^{N}\frac{e^2}{r_{ij}} \tag{4}$$

where the r_i are the locations of the electron and r_{ij} are the separation between the electrons i and j. Here, we neglect smaller terms like the spin-orbit interaction, hyperfine interactions, and recoil motion of the nucleus, and relativity.

Accounting for the spin, we develop the notation that the electron orbital is given by $\psi(r, \vec{t}, \sigma) = \phi(r,t) \left|\pm\frac{1}{2}\right\rangle$. The self-consistent methods are based on the Rayleigh Ritz variational principle,[2] which states that the ground state eigenfunction of the Hamiltonian is that wavefunction that minimizes the expectation value of H,

$$E =< \psi|H|\psi > \tag{5}$$

constrained by the Pauli-exclusion principle and that the wavefunction must be normalized to unity:

$$\int |\Psi|^2 d^N x = 1 (\text{integration over all space and spin coordinates of all N electrons}) \tag{6}$$

.

From before, we see that under our approximations, each of the electrons move independently in an orbital ψ_α and that Psi is properly normalized it the single-particle wavefunctions are orthonormal:

$$\delta_{\alpha,\alpha'} \int \delta_{\alpha,\alpha'} \psi^*_\alpha(\vec{r})\psi_{\alpha'}(\vec{r}) d^3r = \delta_{\alpha,\alpha'} \tag{7}$$

After substituting our wavefunction defined in (3) and (7) into (5):

$$E = \sum_{\alpha=1}^{N} \langle\alpha| \frac{p^2}{2m} |\alpha\rangle + \int [-\frac{Ze^2}{r} + \frac{\Phi(\vec{r})}{2}]\rho(\vec{r})d^3r - \frac{1}{2}\sum_{\alpha,\alpha'=1}^{N} \delta_{\sigma_\alpha,\sigma_{\alpha'}} \langle\alpha\alpha'| \frac{e^2}{r_{ij}} |\alpha'\alpha\rangle \tag{8}$$

The first term represents the kinetic energy, which is the sum of the kinetic energies of the single particle orbitals. The first term inside the integral represents the electron-nucleus attraction, and the second term inside the integral represents the electron-electron repulsion with the total charge of $-Ne$ distributed in space with density $\rho(\vec{r})$. The final term is the exchange energy, which arises from the anti-symmetry of the trial wavefunction. In the case of Helium with $N = 2$ the exchange term is 0 because $\delta_{\sigma_\alpha,\sigma_{\alpha'}} = 0$ by the Pauli-exclusion principle.

In the above expression, the first term each one-body matrix element of the kinetic enery is given by the integral expression:

$$\langle\alpha| \frac{p^2}{2m} |\alpha\rangle = -\frac{\hbar^2}{2m}\int \psi^*_\alpha(\vec{r})\nabla^2\psi_\alpha(\vec{r})d^3r \tag{9}$$

Furthermore, the electron density is a sum in (8) is a sum of the single particle densities:

$$\rho(\vec{r}) = \sum_{\alpha=1}^{N} |\psi_\alpha(\vec{r})|^2 \tag{10}$$

and the electrostatic potential generated by the electrons is

$$\Phi(\vec{r}) = e^2 \int \frac{\rho(\vec{r'})}{|\vec{r} - \vec{r'}|} d^3r'$$

Alternatively, in differential form by the Poisson equation

$$\nabla^2\Phi = -4\pi e^2 \rho(\vec{r})$$

For the 2-electron problem, the ground state of their motion around the nucleus is in the $1s^2$ configuration where both electrons have the same spatial state. For convenience, we let the trial single-particle wavefunction be

$$\psi = \frac{1}{\sqrt{4\pi}r} R(r) \left|\pm\frac{1}{2}\right\rangle \tag{11}$$

where we have implicitly assumed that there is no angular dependence to the wavefunction. From (2), the many-body wavefunction becomes

$$\Psi = \frac{1}{\sqrt{2}}\frac{1}{4\pi r_1 r_2}R(r_1)R(r_2)[\left|\frac{1}{2}\right\rangle\left|-\frac{1}{2}\right\rangle - \left|-\frac{1}{2}\right\rangle\left|\frac{1}{2}\right\rangle] \tag{12}$$

From (7), the normalization condition becomes

$$\boxed{\int_0^\infty R^2(r)dr = 1} \tag{13}$$

while (8) becomes

$$\boxed{E = 2 * \frac{\hbar^2}{2m}\int_0^\infty (\frac{dR}{dr})^2 dr + \int_0^\infty \big[-\frac{Ze^2}{r} + \frac{1}{4}\Phi(r)\big]\rho(r)4\pi r^2 dr} \tag{14}$$

and the definition of the density in (10) reduces to:

$$\rho(r) = 2 * \frac{1}{4\pi r^2}R^2(r) \tag{15}$$

and Poisson's equation defining Φ reduces to

$$\frac{1}{r^2}\frac{d}{dr}\big[r^2\frac{d\Phi}{dr}\big] = -4\pi e^2\rho \tag{16}$$

In the full Hartree-Fock approximation, E is a functional of R and is required to be stationary with respect to all possible norm-conserving variations of the single-particle wavefunction.[3] With the normalization constraint (13) enforced by the method of Lagrange multipliers, and arbitrary variations of $\delta R(r)$ requires

$$\delta(E - 2\epsilon\int_0^\infty R^2 dr) = 0 \tag{17}$$

where ϵ is the Lagrange multiplier to be determined after variation. The standard techniques of variational calculates lead to solving the Schroedinger-like equation.

$$\boxed{\big[-\frac{\hbar^2}{2m}\frac{d^2}{r^2} - \frac{Ze^2}{r} + \frac{1}{2}\Phi(r) - \epsilon\big]R(r) = 0} \tag{18}$$

This involves choosing an eigenvalue ϵ of the single-particle Hamiltonian Φ.

Summary The boxed equations (14,18) are the 2 coupled non-linear differential equations in 1D that form the Hartree-Fock approximation. So far, we see that this is already a dramatic simplification of the otherwise 6-dimensional Schroedinger equation we would have to solve otherwise.

1.3 The Numerov Method

To solve equations (16,18), it's possible to use the Runge-Kutta method, but instead we will focus on using the Numerov method, an ODE solver with the same accuracy as a fifth-order Runge-Kutta solver.

The Numerov method solves a class of problems in the form of a linear, second order equation:

$$\frac{d^2y}{dx^2} + k^2(x)y = S(x) \tag{19}$$

where S is an inhomogeneous driving term and $k^2$is a real function. The Numerov algorithm is a simple and efficient method for integrating the second-order differential equations in the above form. Writing the finite difference definition for $y''(x)$,

$$y''(x) = \frac{y(x+h) - 2y(x) - y(x-h)}{h^2} - \frac{h^2}{12}y''''(x) + O(h^4)$$

where the second term in the expression is the error term written out explicitly. To approximate,

$$\begin{aligned} y''(x) + k^2(x)y(x) &= S(x) \qquad \text{take the second serivative of both sides} \\ y''''(x) &= \frac{d^2}{dx^2}(-k^2y + S) \qquad \text{rearrange in finite difference} \\ &= -\frac{(k^2y)_{n+1} - 2(k^2y)_n + (k^2y)_{n-1}}{h^2} + \frac{S_{n+1} - 2S_n + S_{n-1}}{h^2} + O(h^2) \end{aligned}$$

After some substitution into the original finite difference definition, we get the following solution:

$$\boxed{(1 + \frac{h^2}{12}k^2_{n+1})y_{n+1} - 2(1 - \frac{5h^2}{12}k^2_n)y_n + (1 + \frac{h^2}{12}k^2_{n-1}) = \frac{h^2}{12}(S_{n+1} + 10S_n + S_{n-1}) + O(h^6)} \tag{20}$$

where we can arrange and solve for y_{n+1} if we know the other terms.

In the case of the specific problem we are trying to solve, the two are the 1) the Poisson equation (16) and 2) the eigenvalue equation (18). For the Poisson equation, we make a standard substitution

$$\Phi(r) = \frac{\varphi}{r} \tag{21}$$

which results in solving the differential equation

$$\frac{d^2\varphi}{dr^2} = -4\pi r\rho \tag{22}$$

which is in the form of (19) given above. Thus, the solution to the Poisson equation is

$$\varphi_{n+1} = \frac{h}{12}(-4\pi e^2 r)(\rho_{n+1} + 10\rho_n + \rho_{n-1}) - \varphi_{n-1} + 2\varphi_n \tag{23}$$

Likewise, equation (18) is also written in the form of (19) and can be rearranged to solve for $R(r)$. The solution using Numerov's algorithm is

$$R_{n+1} = \frac{2(1 - \frac{5h^2}{12}\left[\frac{2m}{\hbar^2}(\frac{Ze^2}{r} - \frac{1}{2}\Phi(r) + \epsilon)\right]_n)R_n - (1 + \frac{h^2}{12}\left[\frac{2m}{\hbar^2}(\frac{Ze^2}{r} - \frac{1}{2}\Phi(r) + \epsilon)\right]_{n-1})R_{n-1}}{1 + \frac{h^2}{12}\left[\frac{2m}{\hbar^2}(\frac{Ze^2}{r} - \frac{1}{2}\Phi(r) + \epsilon)\right]_{n+1}} \tag{24}$$

Solving for equations (23,24) provides a recursion relation for integrating forward in r with local error $O(h^6)$. This method is also more efficient than the Runge-Kutta because it requires. What should be noted is that in general, y_1 and and y_0 need to be calculated prior to using the recursion relation to integrate forward. The method and reasoning to calculating φ_0, φ_1 goes as follows: since the wavefunction must have finite potential Φ at the origin, we have to make $\varphi_0 = 0$ so that that condition is satisfied. For φ_1, we use the definition of φ in equation (21) and definition of the Coulomb potential at the origin as

$$\Phi(0) = 4\pi e^2 \int \rho(r)dr \tag{25}$$

to approximate

$$\phi(h) = 4\pi e^2 * h * \int \rho(r)dr \tag{26}$$

We have thus concluded our discussion on the Numerov method. In the following paragraph, we will discuss a correction to the estimation φ that results in greater accuracy. Solutions to the homogeneous equation of (22) can be added to any particular solution of (22). That is, we can add $\varphi = Ar$ or $\varphi = B/r$ (A, B are constants) to the solution to φ and it will be correct. The latter term, $\varphi = B/r$ is the physical solution (clearly) because we require the potential to go to zero at $r = \infty$. However, imprecisions in numerical round-off errors in the integration process can add small admixtures of the $\varphi = Ar$ to the solution. To mitigate this issue, we subtract a multiple of the unphysical solution to guarantee the correct physics behavior in the asymptotic region. The example code is seen in Koonin Ch. 3.[3]

1.4 The Shooting Method

The shooting method is an iterative solution to find the eigenfunctions and eigenvalues of a differential equation. We guess a trial eigenvalue to an eigenvalue equation (in this assignment, the eigenvalue equation is (18)) and generate a solution by integrating the solution as an initial value problem. We change the trial eigenvalue and integrate again, repeating the process until a trial eigenvalue satisfies the proper boundary conditions.

For the problem at hand, we integrate forward with a trial eigenvalue ϵ using the Numerov method from $r = 0$ to $r = \infty$ with the initial conditions

$$R(r = 0) = 0, R(r = h) = \delta \tag{27}$$

The number δ is arbitrary and can be chosen for convenience. This is because the solution for $R(r)$ will later be normalized. Upon integrating to $r = \infty$, we will generally find a non-vanishing value of R, and thus, must readjust ϵ until we find $R(r = \infty) = 0$ with specified tolerance. Then, we will have found the eigenvalue ϵ and eigenfunction $R(r)$.

1.5 Bisection Root-finding Method

The Bisection method is a root-finding method that repeatedly bisects an interval into a smaller subinterval in which the root must be contained. It takes advantage of the intermediate value theorem that given a bracket [a,b], if $f(a) < 0$ and $f(b) > 0$, then there is a root in the interval [a,b] for a continuous, well-behaved function. The `bisect` method provided in Numerical Recipes in section 9.1.1.[?] Once the root has been bracketed between the interval $[x_1, x_2]$, the midpoint at $x_3 = (x_1 - x_2)/2$ is evaluated. If $f(x_3)$ is the same sign as $f(x_1)$, then $x_1 < -x_3$. Otherwise if $f(x_3)$ is the same sign as $f(x_3)$, then $x_2 < -x_3$. The process repeats itself until $f(x_3)$ is within a specified tolerance.

In the context of the project, the bisection procedure is used to find the eigenvalue ϵ. When the boundary condition is satisfied, the function $R(\infty) vs. \epsilon$ will cross zero. The bisection method is called to find ϵ.

2 Algorithm and Program Execution

2.1 Units

For the numerical solution, we adopt a system of units where all lengths are measured in Angstroms and all energies are in electron volts. The constants used are:

$$\frac{\hbar^2}{m} = 7.6359eV - A^2; e^2 = 14.409ev - A, a_{bohr} = \frac{h-bar^2}{me^2}; Ry = \frac{e^2}{2a} = 13.595 \tag{28}$$

Previously, we have used $r = \infty$ to specify bounds of integration and boundary conditions. Since it is impossible to integrate to $r = \infty$, we specify a trial value of $r = \infty \approx R_{max} = 5$. The reason why R_{max} is not greater than 5 is because we have found that for values much larger than 5, the boundary conditions of $R(R_{max}) = 0$ is never satisfied due to the admixture of the undesirable exponentially growing solution in the classically forbidden region.

We use a step size $h = 5 * 10^{-4}$.

2.2 Algorithm

The following algorithm is implemented to estimate the ground-state energy of the Helium atom.

1. Initiate a trial wavefunction $R(r)$. In this example, we will compare

$$R_1(r) = 2(\frac{Z^*}{a_{bohr}})^{\frac{1}{2}}\frac{Z^*r}{a_{bohr}}e^{-Z^*r/a_{bohr}} \tag{29}$$

for $\boxed{Z^* = 2 - \frac{5}{16}}$, which is the result when a standard variational method is used for the two-electron system that takes R to be a hydrogenic 1s orbital parameterized by an effective charge Z^{*2} AND

$$R_2(r) = 2(\frac{Z^*}{a_{bohr}})^{\frac{1}{2}}\frac{Z^*r^2}{a_{bohr}}e^{-Z^*r/a_{bohr}} \tag{30}$$

which is decided arbitrarily. The reason to choose 2 values of initial wavefunctions is to see that variations in the initial trial wavefunction will still converge to the same result.

2. Solve for the potential $\Phi(r)$ and the density $\rho(r)$ generated by the initial wavefunction using the Numerov method as given in (23) and the definition of $\rho(r)$ in (15) respectively.

3. Use the shooting method to solve for the eigenvalue and eigenfunction $R(r)$ by solving for (18). Normalize the new eigenfunction using (13).

4. Re-calculate the new potential $\Phi(r)$ and the new energy E from (14). Recalculate the new density $\rho(r)$ where

$$\rho(r) = \rho_{new}(r) * w + \rho_{old}(r)(1-w) \tag{31}$$

where w is a relaxation parameter that determines how much of the ρ_{new} defined as the density produced by the new radial eigenfunction and the ρ_{old} are mixed. For this project, we only use $w = 0.5$.

5. Repeat steps 3 and 4 until the difference in total energy

$$|E_{new} - E_{old}| < tol \tag{32}$$

where in this project, $tol = 10^{-5}$.

2.3 Notes on `main.cpp`

The algorithm that we used in `main.cpp` was described above succinctly. However, there are a few subtleties in the implemented code that is worth discussing here. Firstly, to evaluate the integrals in (13) and (14), we use the trapezoid method**?** and integrate from 0 to R_{max}.

The trapezoidal rule[1] is a numerical integration method for approximating a definite integral by approximating the area of the graph of a function $f(x)$ as a trapezoid and calculating its area. It follows that

$$\int_a^b dx f(x) \approx \sum_{i=0}^{N} \frac{f(x_{i+1}) - f(x_i)}{2} \Delta x = \frac{\Delta x}{2}(f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + ... + f(x_N)) \quad (33)$$

where $N \equiv$ number of intervals, $\Delta x \equiv \frac{b-a}{N}$, and $x_i = a + i * \Delta x$. Although more accurate integral algorithm exists, we use the trapezoid rule for simplicity.

`main.cpp` also tests for the `calculateE(), calculateRho(), calculatePhi(), shoot()`. These tests are used to give confidence that we have correctly implemented the Numerov method and shooting method. In `testCalculateE()` method, we verify the energies associated with the hydrogenic orbital as defined in (29) and compare it with the analytical variational method results of $E = -\frac{e^2}{a_{bohr}}\left[Z^2 - 5Z/8 + 25/256\right]$. Furthermore, it is possible to print out the potential energy contribution to E and the kinetic energy contribution to E and numerically verify the Virial theorem:

$$KE = -\frac{1}{2}PE \quad (34)$$

From our output in `textCalculate()`, we see that we have correctly implemented `calculateE(), calculateRho(), calculatePhi()` correctly as the calculated ground state energy of Helium is -76.6097 and analytical value of ground state energy of Helium is -77.4332.

In the `shoot()` calculation, for every iteration, we must find the largest negative eigenvalue ϵ from bisection to find the ground state eigenvalue and eigenfunction. Thus, we start searching for eigenenergies at an arbitrary large negative eigenvalue and increase by an increment $\Delta = 1eV$.

Furthermore, in the shooting method, we must use Numerov method to integrate forward as given in (24). However, it is important to note that in (24), the second term in the numerator of (24) will yield a $Ze^2/r[0] = Ze^2/0$ which is a divide by 0 error. Thus, it is important that the entire second term in the numerator be manually set to 0 for $R[n = 2]$.

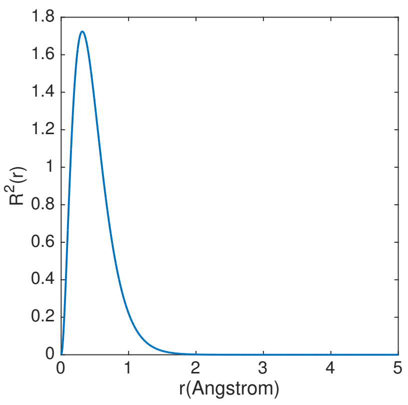
Now to test the shooting method, we set $R[1] = 1$ arbitrarily as $R(r)$ will later be normalized after solving for the correct eigenfunction. In `testShoot()`, we set $\Phi = 0$, which represents no electron-electron interaction (hence, 1 electron system). We calculate that the eigenenergy for the hydrogenic Helium ($Z = 2$) is $E = -54.3797$, which is in agreement with the relation $E = E_{hydrogen}/Z^2$ with $E_{hydrogen} = -13.6eV$.

3 Results and Discussion

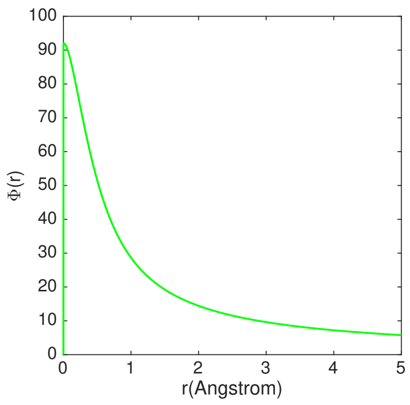
3.1 Function test results

First, we will show the results in our test functions `testCalculateE(), testShoot()`.

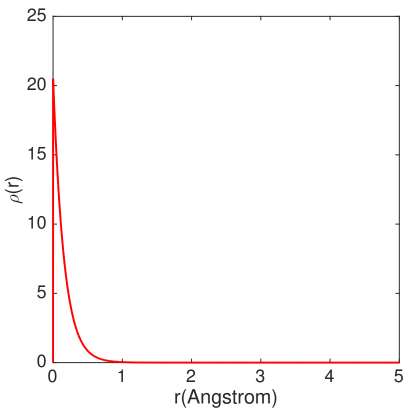
[1]Equation 4.1.11 in Numerical Recipes , 3rd edit, by Press et al



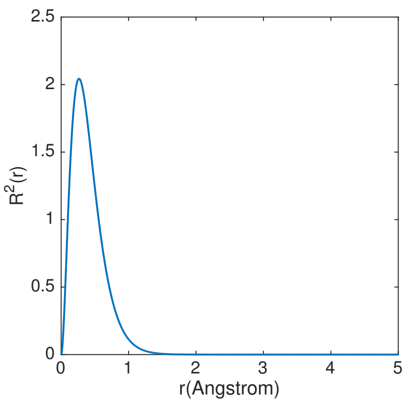
(a)



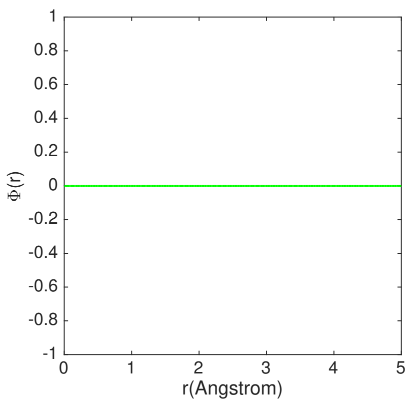
(b)



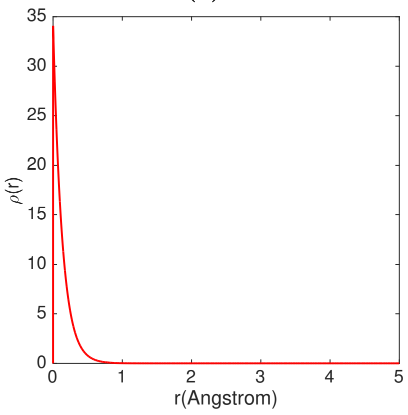
(c)



(d)



(e)

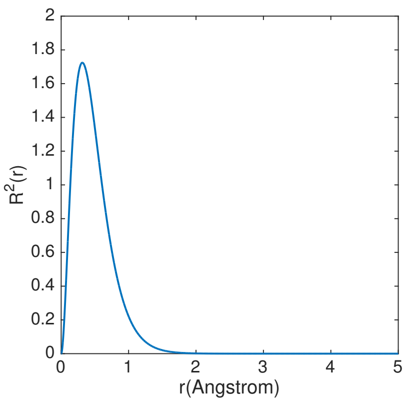


(f)

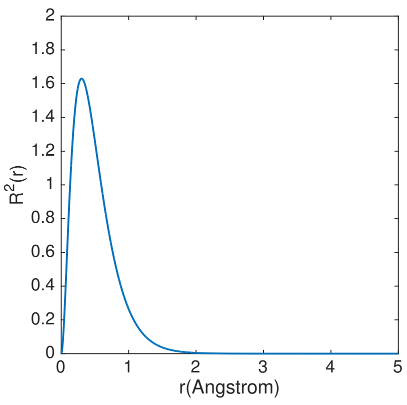
Figure 1: (a-c) are the square-wavefunction, potential, and density calculated for the trial wavefunction in (29). (e-f) are the square-wavefunction, potential, and density calculated after solving the eigenvalue equation (18) when $\Phi = 0$. It is important to see that the shooting method correctly returns a 1s radial wavefunction similar in shape to that of the 1s radial wavefunction of hydrogen.

3.2 Hartree Fock Results

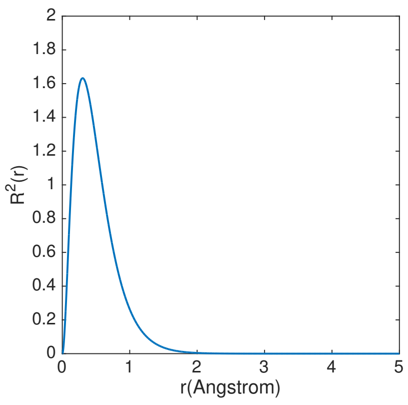
In *main.cpp*, every time we iterate through the algorithm, we record the total energy, radial wavefunction, density, and potential. In figure (2) we will show the result for when the trial wavefunction is given by (29) and in figure (3) we will show the result when the trial wavefunction is (30). The total energies at each iteration are tabulated in table 1.



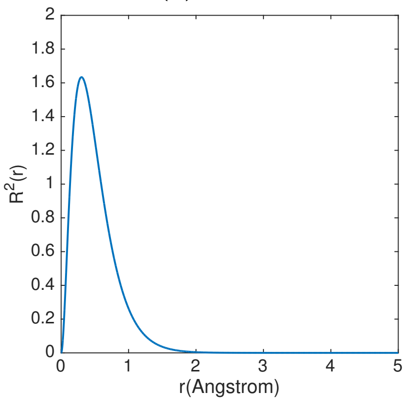
(a) i=0



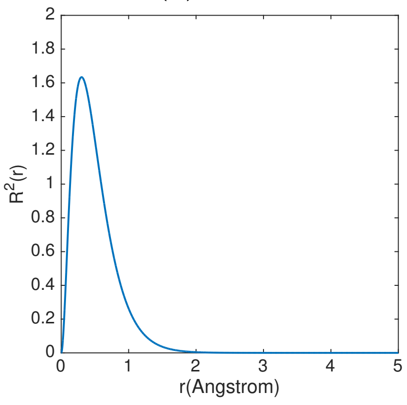
(b) i=1



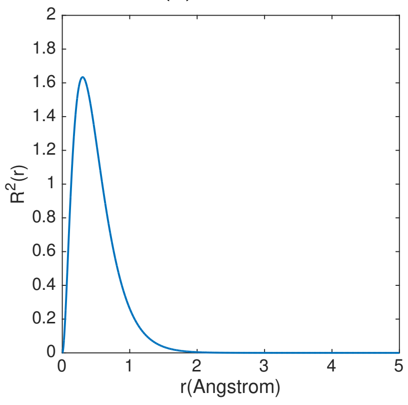
(c) i=2



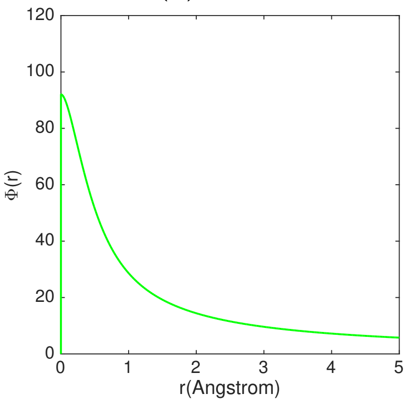
(d) i=5



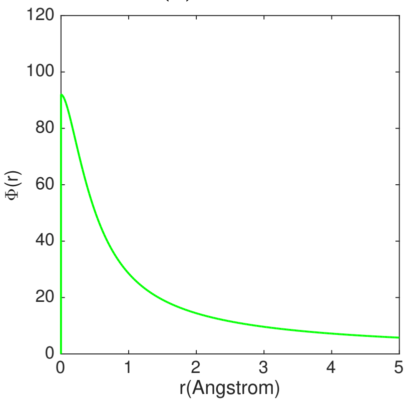
(e) i=7



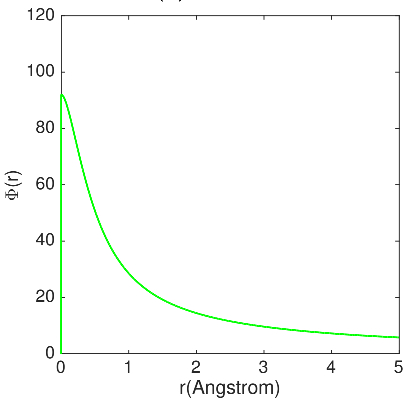
(f) i=13



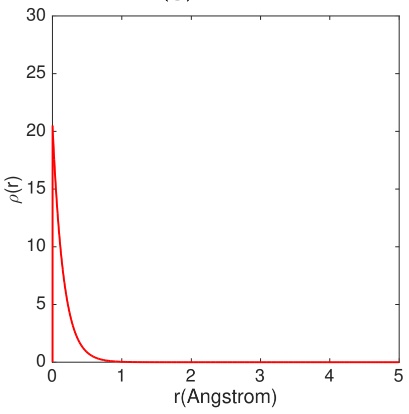
(g) i=0



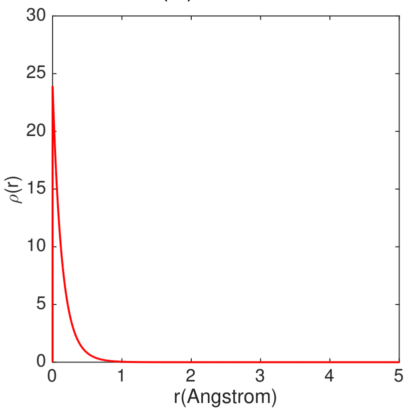
(h) i=5



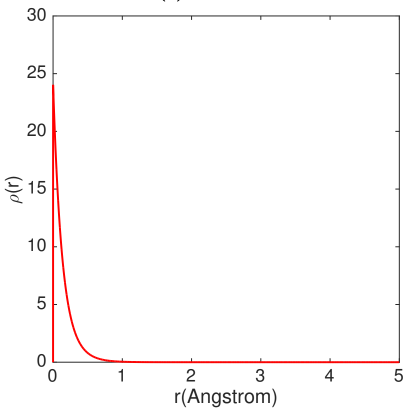
(i) i=13



(j) i=0

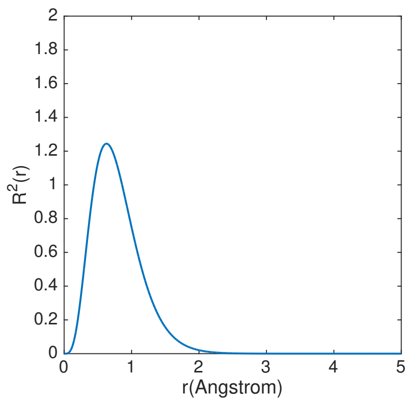


(k) i=5

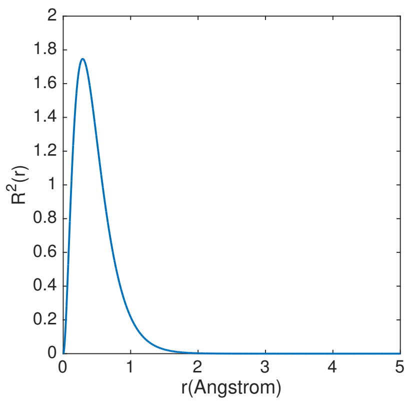


(l) i=13

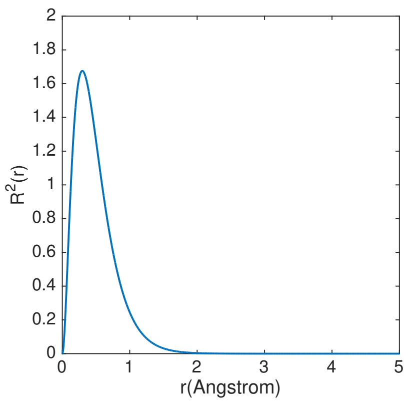
Figure 2



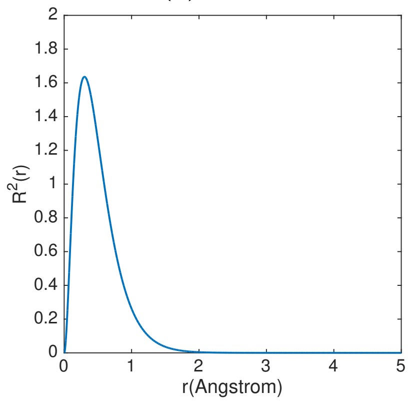
(a) i=0



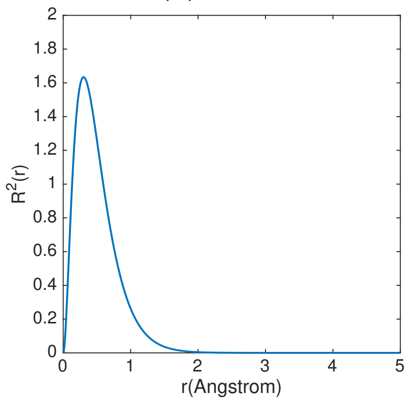
(b) i=1



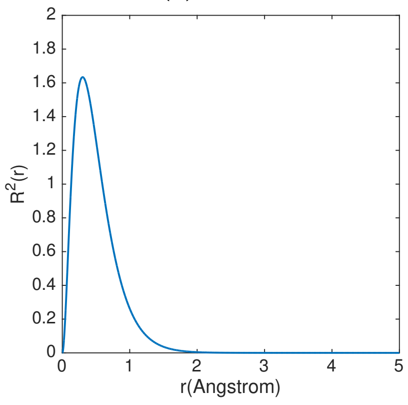
(c) i=2



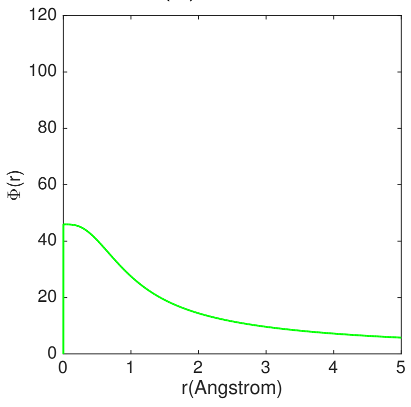
(d) i=5



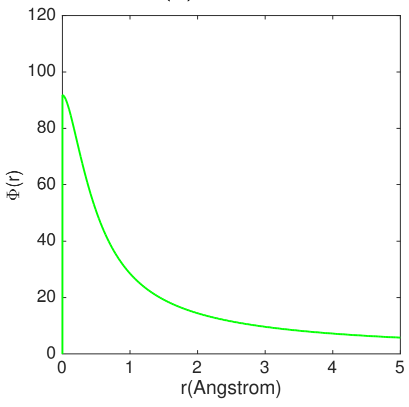
(e) i=7



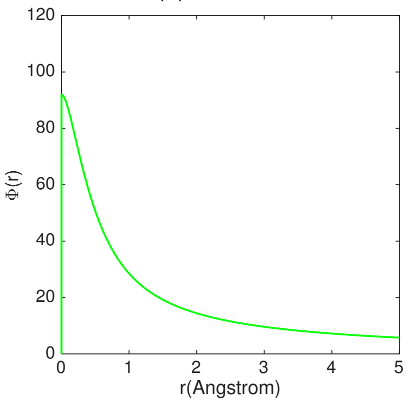
(f) i=15



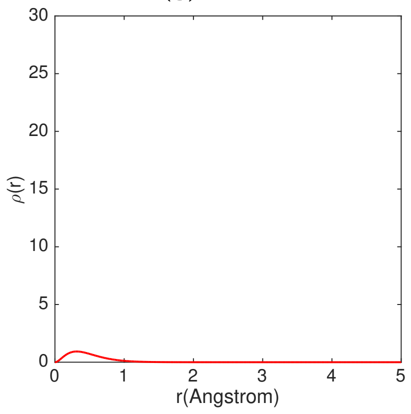
(g) i=0



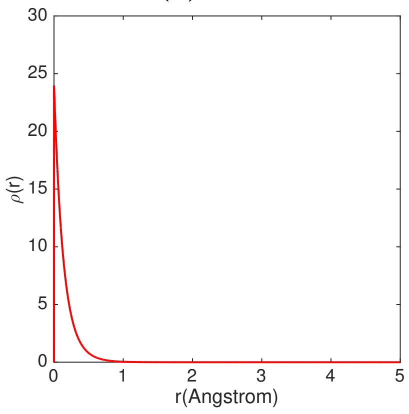
(h) i=5



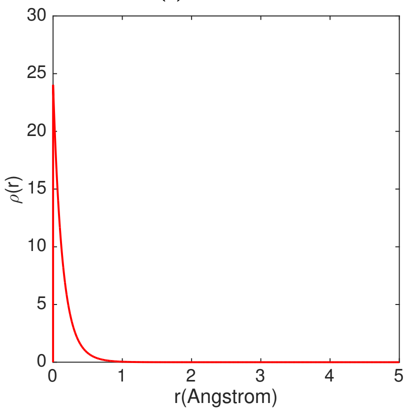
(i) i=15



(j) i=0



(k) i=5



(l) i=15

Figure 3

Table 1: Iteration of ground state energies of Helium atom

Iteration	Total E (R_1)	Total E (R_2)
0	-78.0318	-28.108
1	-77.8295	-60.0584
2	-77.8688	-71.7212
3	-77.9359	-75.8577
4	-77.9854	-77.3006
5	-78.0155	-77.7979
6	-78.0322	-77.9672
7	-78.0411	-78.0239
8	-78.0457	-78.0424
9	-78.0481	-78.0483
10	-78.0493	-78.05
11	-78.0498	-78.0504
12	-78.0501	-78.0505
13	-78.0503	-78.0505
14	N/A	-78.0504

3.3 Discussion

From Koonin,[3] we see that the experimental value of the ground state energy of the Helium atom is

$$E_{helium} = -78.88eV \tag{35}$$

From our results, we get that $E_{helium} = -78.05eV$. Our results generally agree with the experimental values but is not exact. Nevertheless, this demonstrates that the Hartree Fock method is able to get a convergent solution for $n = 2$ electrons. It is most interesting to note that even upon changing the trial wavefunction, the ground state energy converges to the same value. As expected, most of the changes in the energy comes from the first few iterations. The deviations from experimental values are postulated by the fact that our assumptions involving the Born-Oppenheimer approximation and mean-field approximation provide limitations to our model and are not adequate to fully account for physical phenomena. Furthermore, we neglect spin-orbit coupling, etc. so that will also affect our answer.

In this assignment, we implement a very simplistic version of what could be a much more complex Hartree fock method. There were many parameters that we could have greater control in particular: R_{MAX}, w. Although qualitatively, changing the value of R_{MAX} from ± 2 will not result in significant change in our result, it would be ideal if we could automate a search that would use the appropriate R_{MAX} for an arbitrary system. Furthermore, it would be ideal to determine the optimal relaxation parameter w so that the solution converges the fastest. In Koonin, a method is proposed to solve the eigenvalue equation (18) by integrating both forward AND backwards to ensure proper behavior at the boundaries. Furthermore, we could have determined an optimal step size h, but instead chose an arbitrary small step size of $h = 5 * 10^{-4}$. It is possible to implement a $\log(r)$ grid where the step size is smaller closer to the origin and larger farther away. However, this adds complexity to our solution because then Numerov solution would have to be revised.

However, there is nothing physically requiring the Hartree Fock method to ever converge to a specific value. It is only by happenstance that the Helium atom converges but for larger atoms, this may not be the case. In that case, we would have to rely on other methods like Density Functional Theory.

4 Conclusions

In conclusion, Hartree Fock is an insightful but limited approximation method to calculating the ground state (and excited state) wavefunction and energies of atomic systems. It is possible to extend the Hartree Fock model to systems with more than 1 nucleus and/or more than 2 electrons. In that case, we would then need to include the additional terms in equation (2): $\hat{V}_{NN}, \hat{V}_{ee}$. The latter would involve a much more complicated exchange-integral. Furthermore, it is possible to assume that the angular dependence is not a spherical harmonic, and in that case, the integration would need to be done as a 6-dimensional integral at worst.

Nevertheless, the Helium atom system is a good starting point to test the most basic Hartree Fock implementations. It combines many methods learned throughout the semester like bisection for root-finding, trapezoid method for integration, Numerov method for solving differential equations, and the shooting method for solving eigenvalue equations. Our results give us an acceptable value for the ground state energy estimate, but is not much better than the standard variational principle. However, where the Hartree Fock will shine more is when dealing with more complicated systems. Alternatively, there are a wide array of other methods besides Hartree Fock like Density Functional Theory (DFT) that seeks to solve the same problem – finding the energies and wavefunctions. DFT methods is is an active area of research where the approximation to the exchange interaction will vary by the flavor of the "functional" that you use.

References

[1] Pablo Echenique and J. L. Alonso. A mathematical and computational review of hartree–fock scf methods in quantum chemistry. *Molecular Physics*, 105(23-24):3057–3098, 2007.

[2] D.J. Griffiths. *Introduction to Quantum Mechanics*. Pearson international edition. Pearson Prentice Hall, 2005.

[3] Steven E. Koonin and Dawn Meredith. *Computational Physics (FORTRAN Version); Disk Enclosed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1990.

Source Code Listing

```
//
//  main.cpp
//  HartreeFock
//
//  Created by Junlan Lu on 2018/11/18.
//  Run on a core i7 with XCode 9.4.1 on MacOS 10.14
//

#include <iostream>
```

```
#include <cmath>
#include "bigarrayt.hpp"
#include <string>
#include <sstream>
#include <fstream>
#include <iomanip>
using namespace std;
/* Defines constant RMAX to be the maximum radius of the lattice, ABhor to be bohr radius, etc.
 */
double RMAX=5, e2 =14.409, h1,h2m=3.81795,ABhor = 0.5299,zstar=2.0-5.0/16.0;

/*Implements the trapezoid integration technique to integrate an integrand dat from 0 to RMAX
 */
double integrate_trapezoid(arrayt<double> dat){
    int i,n=dat.n1();
    double dr = RMAX/n;
    double I = 0.5*dr*(dat(0)+dat(n-1));
    for(i = 1;i<n; i++){
        I = I + dat(i)*dr;
    }
    return I;
}

/* normalizes the R wavefunction
 */
void normalize(arrayt<double> &R){
    int i,n=R.n1();
    double I;
    arrayt<double> R2(n);
    for(i=0;i<n;i++){
        R2(i) = R(i)*R(i);
    }
    I = 1/sqrt(integrate_trapezoid(R2));
    for(i=0;i<n;i++){
        R(i) = R(i)*I;
    }
}

/* solves for the density given the radial wavefunction
 */
void calculateRho(arrayt<double> &Rho, arrayt<double> &R){
    int i,n = R.n1();
    double r,dr = RMAX/n;
    Rho(0)=0;
    for(i=1;i<n;i++){
        r = i*dr;
        Rho(i)=R(i)*R(i)/(2*M_PI*r*r);
    }
}

/* solves for the potential term given the density by solving Poisson's equation
 */
void calculatePhi(arrayt<double> &Rho,arrayt<double> &Phi){
    int i, n=Rho.n1();
    double r,dr = RMAX/n, dr2=dr*dr, m;
    arrayt<double> phi(n), integrand(n);
    for(i=0;i<n;i++){
        r = i*dr;
        integrand(i) = r*Rho(i);
    }
    phi(0) = 0;
    phi(1) = 4*M_PI*e2*dr*integrate_trapezoid(integrand);
    for(i=2;i<n;i++){
        r = i*dr;
        phi(i) = (dr2/12)*(-4*M_PI*e2*r)*(Rho(i)+10*Rho(i-1)+Rho(i-2))-phi(i-2)+2*phi(i-1);
    }

    Phi(0)=0;
    for(i=1;i<n;i++){
        r = i*dr;
        Phi(i) = phi(i)/r;
    }

    m=(phi(n-1)-phi(n-11))/(10*dr);
    for(i=1;i<n;i++){
        r = i*dr;
        Phi(i) = Phi(i)-m;
    }
}


/* calculates the total energy of the 2-electron system, given the density, potential,
  and radial wave function
 */
double calculateE(arrayt<double> &Phi, arrayt<double> &Rho, arrayt<double> &R, double z){
    int i, n = R.n1();
    double int1=0,int2=0,int3=0,E,r,dr = RMAX/n,DR,KEN,VEN,VEE;
    arrayt<double> integrand(n),integrand1(n-1), integrand2(n-1),integrand3(n-1);
    for(i=0;i<n-1;i++){
        r=i*dr;
        DR=(R(i+1)-R(i))/dr;
        integrand1(i)= DR*DR;
        integrand2(i)= Rho(i)*r;
        integrand3(i)= Phi(i)*Rho(i)*r*r;
    }
    int1=integrate_trapezoid(integrand1);
    KEN=2*h2m*int1;
    int2=integrate_trapezoid(integrand2);
    VEN=-4*M_PI*z*e2*int2;
    int3=integrate_trapezoid(integrand3);
    VEE=M_PI*int3;
    E = KEN+VEE+VEN;
    return E;
}

/* given an eigenvalue epsilon, calculate R(RMAX) by using the numerov method
 */
double calculateRMax(arrayt<double> &Phi, arrayt<double> &R, double epsilon, double z){
    int n=Phi.n1(),i;
    double Rmax, r, dr = RMAX/n, const1=dr*dr/(12*h2m), term1, term2, term3;
    R(0) = 0;
    R(1) = 1;
    for(i=2;i<n;i++){
        r = i*dr;
        term1 = 2*(1-5*const1*(epsilon+z*e2/(r-dr)-Phi(i-1)/2))*R(i-1);
        term2 = (1+const1*(epsilon+z*e2/(r-2*dr)-Phi(i-2)/2))*R(i-2);
        if(i==2) term2 = 0;
        term3 = (1+const1*(epsilon+z*e2/r-Phi(i)/2));
        R(i) = (term1 - term2)/ term3;
    }
    normalize(R);
    Rmax=R(n-1);
    return Rmax;
}

/*bisect function takes in a specified function, a bracket x1-x2, and a tolerance value. It returns
 -1 if a root is not bracketed and the calculated root within the tolerance specifation if
 the root is found. Uses the bisection method.
 */
double bisect(double(*f)(arrayt<double> &, arrayt<double> &, double, double),
 double x1, double x2, double tol, arrayt<double> &Phi, arrayt<double> &R, double z){
    double f1, f2, f3 = -1, x3;
    f1 = f(Phi, R, x1, z);
    f2 = f(Phi, R, x2, z);
    if((f1 * f2 > 0)){
        return 1;
    }
    do {
        x3 = 0.5 * (x1 + x2);
        f3 = f(Phi, R, x3, z);
        if(f1 * f3 > 0){
            x1 = x3;
            f1 = f3;
        }
        else if (f2 * f3 > 0){
            x2 = x3;
            f2 = f3;
        }
    } while (abs(f3) > tol);

    return x3;
}

/* implements the shooting method to solve for R(r)
 */
double shoot(arrayt<double> &Phi, arrayt<double> &R, double z){
    int count=0, maxcount = 1e5;
    double epsilon = 1, trial1, trial2, increment = 1, tol = 1e-5;
    trial1 = -200;
    trial2 = trial1+increment;
    while(epsilon > 0){
        epsilon = bisect(calculateRMax, trial1, trial2, tol, Phi, R, z);
        trial1 = trial1 + increment;
        trial2 = trial2 + increment;
        count++;
        if(count > maxcount){
            cout << "Cannot find any eigenvalues" << endl;
            break;
        }
    }
    calculateRMax(Phi, R, epsilon, z);
    return epsilon;
}

/* test function to test the CalculateE() method
 */
void testCalculateE(){
    int i,n=1e4;
    double const1=2*sqrt(zstar/ABhor)*zstar/ABhor,dr=RMAX/n,r,E, z=2.0;
    arrayt<double> R(n),Phi(n),Rho(n);
    for(i=0;i<n;i++){
        r=i*dr;
        R(i) = const1*r*exp(-zstar*r/ABhor);
    }
    normalize(R);
    calculateRho(Rho, R);
    calculatePhi(Rho, Phi);
    E=calculateE(Phi,Rho,R,z);

    cout<<"Calculated ground state energy of Helium: "<< E << endl;
    cout<<"Analytical value of ground state energy of Helium: "<<-e2*(z*z-5*z/8+25.0/256)/ABhor<<endl;

    ofstream fp1;
    fp1.open("testCalculateE.dat");
    if( fp1.fail() ) {
        cout << "cannot open file" << endl;
    }
    for(i=0;i<n;i++){
        r=i*dr;
        fp1 << setw(15)<< r << setw(15) << R(i)*R(i) << setw(15) << Rho(i) << setw(15) << Phi(i) << endl;
    }
    fp1.close();
}
```

```
/* test function to test the CalculateRMAX() method for an arbitrary epsilon
 */
void testCalculateRMax(){
    int i,n = 1e4;
    double dr = RMAX/n,r,z=2.0;
    arrayt<double> Phi(n), R(n), Rho(n);
    for(i=0; i < n ;i++){
        Phi(i)=0;
        R(i)=2;
    }
    calculateRMax(Phi, R, -20,z);
    ofstream fp1;
    fp1.open("testRMax.dat");
    if( fp1.fail() ) {
        cout << "cannot open file" << endl;
    }
    for(i=0;i<n;i++){
        r=i*dr;
        fp1 << setw(15)<< r << setw(15) << R(i)*R(i) << endl;
    }
    fp1.close();
}

/* test function to test the shoot() method for the 1 electron case
 */
void testShoot(){
    int i,n = 1e4;
    double epsilon, E, dr = RMAX/n,r;
    arrayt<double> Phi(n), R(n), Rho(n);
    for(i=0; i < n ;i++){
        Phi(i)=0;
    }
    epsilon = shoot(Phi, R, 2);

    calculateRho(Rho, R);
    E = calculateE(Phi, Rho, R, 2);
    cout << "Energy eigenvalue for z = 2: " <<epsilon << endl;
    ofstream fp1;
    fp1.open("testShoot.dat");
    if( fp1.fail() ) {
        cout << "cannot open file" << endl;
    }
    for(i=0;i<n;i++){
        r = i*dr;
        fp1 << setw(15) << r<< setw(15) << R(i)*R(i) << setw(15) << Rho(i) << setw(15) << Phi(i) << endl;
    }
    fp1.close();
}
/* print the arrays into an output .dat file
 */
void print(arrayt<double> &R, arrayt<double> &Rho, arrayt<double> &Phi, int count){
    int i, n=R.n1();
    double dr=RMAX/n, r;
    ofstream fp;
    string str ="sol2_"+to_string(count)+".dat";
    fp.open(str);
    if( fp.fail() ) {
        cout << "cannot open file" << endl;
    }
    for(i=0;i<n;i++){
        r=i*dr;
        fp << setw(15)<< r << setw(15) << R(i)*R(i) <<setw(15)<<Rho(i)<<setw(15)<<Phi(i)<< endl;
    }
    fp.close();
}
/* main function. intializes R(r) to a trial wavefunction and uses the
 shooting method to solve for the R(r) until the total energy convergese to a
 specified tolerance.
 */
int main(int argc, const char * argv[]) {
    int n=1e4, i, count = 0;
    double w = 0.5, tol = 1e-5, r, dr = RMAX/n, enew, eold, E, z=2.0;
    arrayt<double> Rho(n), RhoNew(n), RhoOld(n), Phi(n), R1(n);
    ofstream fp; fp.open("energy2.dat");
    if( fp.fail() ) {
        cout << "cannot open file" << endl;
    }
    cout << "---------Begin Testing--------"<<endl;
    testCalculateE();
    testCalculateRMax();
    testShoot();
    cout << "---------End Testing--------"<<endl;
    //initializes the trial wavefunction R1 and R2
    for(i=0;i<n;i++){
        r = i*dr;
        R1(i) = 2*sqrt(zstar/ABhor)*(zstar/ABhor)*r*r*exp(-zstar*r/ABhor);
    }
    normalize(R1);
    calculateRho(RhoOld, R1);
    calculatePhi(RhoOld, Phi);
    print(R1, RhoOld, Phi, count);
    E = calculateE(Phi, Rho, R1, z);
    enew = shoot(Phi, R1, z);
    do{
        calculateRho(RhoNew, R1);
        for(i = 0;i < n;i++){
            Rho(i) = RhoNew(i)*w + (1-w)*RhoOld(i);
            RhoOld(i) = Rho(i);
        }
        calculatePhi(Rho, Phi);
        fp <<setw(15)<< count << setw(15) << calculateE(Phi, Rho, R1, 2) << endl;
        eold = enew;
        enew = shoot(Phi,R1,z);
        count++;
        print(R1, Rho, Phi, count);
    }while(abs(eold - enew)>tol);
    fp.close();
    return EXIT_SUCCESS;
}
```