Computational Assignment 1: Using the Laguerre Basis to get Hydrogen Energies and Radial Wavefuctions

May 9, 2024

1   Problem 1

Solutions to the Schrodinger equation for the Hydrogen atom come in the separable form:

$$\Phi_{nlm}(\mathbf{r}) = \Phi_{nl}(r) * Y_l^m(\hat{\mathbf{r}}) \tag{1}$$

Where $\Phi_{nl}(r)$ are the spherically symmetric radially dependent parts of the wavefunction and $Y_l^m(\hat{\mathbf{r}})$ are the Spherical Harmonics, for the quantum numbers n,l and m, representing the principal, angular and magnetic quantum numbers.

These wavefunctions can be represented as:

$$\Phi_{nlm}(\mathbf{r}) = \Phi_{nl}(r) * Y_l^m(\hat{\mathbf{r}}) \tag{2}$$

Analytical solutions to the bound state radial part of the hydrogen atom are completely known, the first few relevant ones for the rest of this report follow:

n	l	$\Phi_{nl}(r)$
1	0	$2re^{-r}$
2	0	$\frac{r}{\sqrt{2}}(1 - \frac{r}{2})e^{-r/2}$
2	1	$\frac{r^2}{\sqrt{24}}(1 - \frac{2r}{3} + \frac{2r^2}{27})e^{-r/3}$
3	0	$\frac{2r}{\sqrt{27}}(1 - \frac{2r}{3} + \frac{2r^2}{27})e^{-r/3}$
3	1	$\frac{8r^2}{27\sqrt{6}}(1 - \frac{r}{6})e^{-r/3}$
4	1	$\frac{r^2}{64\sqrt{15}}(\frac{r^2}{4} - 5r + 20)e^{-r/4}$

If we choose a set of basis functions ϕ_j for $k = 1, 2, \ ... \ , \infty$ which form a complete basis on the Hilbert space, defined as:

$$\langle \mathbf{r}|\phi_j\rangle = \frac{1}{r}\phi_{k_j,l_j}(r)Y_{l_j}^{m_j}(\hat{\mathbf{r}}) \tag{3}$$

These basis function can be used to recover high order approximations to the true radial wavefunction through a sum over a finite number of the basis functions in the following way:

$$|\Phi_i\rangle = \sum_j^N c_j i\,|\phi_j\rangle \tag{4}$$

We need to generate the basis functions for an arbitrary sized basis N. The non-orthonormal basis functions can be given by:

$$\phi_{kl}(r) = \sqrt{\frac{\alpha_l(k-1)!}{(k+l)(k+2l)!}}(2\alpha_l r)^{l+1} e^{-\alpha_l r} L_{k-1}^{2l+1}(2\alpha_l r) \tag{5}$$

We do not want to store arbitrarily large Laguerre basis functions, so we use a recurrence relation for the Laguerre polynomials which gives the following recurrence relation:

$$\tilde{\phi}_{kl}(r) = \frac{2(k-1+l-\alpha_l r)\tilde{\phi}_{k-1,l}(r) - (k+2l-1)\tilde{\phi}_{k-2,l}(r)}{k-1} \tag{6}$$

Where we start with the $k = 1, 2$ basis, for a chosen parameter α:

$$\tilde{\phi}_{1l}(r) = (2\alpha r)^{l+1} e^{-\alpha r} \tag{7}$$

$$\tilde{\phi}_{2l}(r) = 2(l+1-\alpha r)(2\alpha r)^{l+1} e^{-\alpha r} \tag{8}$$

Which can be used to reconstruct any of the basis functions through:

$$\phi_{kl}(r) = \sqrt{\frac{\alpha_l(k-1)!}{(k+l)(k+2l)!}}\tilde{\phi}_{kl}(r) \tag{9}$$

To allow real type variables of FORTRAN to handle the factorials, we can simplify the normalisation factor. First we consider the behaviour of the fraction $\frac{(k-1)!}{(k+2l)!}$

$$\frac{(k-1)!}{(k+2l)!} = \frac{(k-1)!}{(k+2l)(k+2l-1)(k+2l-2)...(k+2l-x)!} \tag{10}$$

If we continue in this way eventually $2l - x = -1$ and we get $(k-1)!$, which cancels out with the numerator, leaving us with:

$$\frac{1}{\Pi_{x=0}^{2l}(k+2l-x)} \tag{11}$$

We can substitute this into our normalisation coefficient to get:

$$\sqrt{\frac{\alpha_l(k-1)!}{(k+l)(k+2l)!}} = \sqrt{\frac{\alpha_l}{(k+l)\Pi_{x=0}^{2l}(k+2l-x)}} \tag{12}$$

Implemented in the code with the following way:

```
do i = 1, N
        p=1.0
        do j = 0, 2*l
                p = real(p*(i+2*l-j))
                !Print *, p, j
        end do
    normalise = sqrt(alpha /((i+l)* p))
    !Print *, "Norm:: ", normalise
    basis(:,i) = normalise*basis(:,i)
    end do
```

To test the bounds of this method, if we compute using the repeated multiplication method and use the Gamma function in FORTRAN to write the same normalisation coefficient directly from the formula; for $\alpha=1$ $l=1$, the k limit for the gamma method is 32. This test was run with only real data types, so the k+2l value was expected to be much lower before the function could not hold accurate values. After this point, the gamma formula returns 0.0, and subsequently NaN. However, the method described above continues to return non-zero results even for k=150, but the accuracy of these values has only been verified up to k=40. This sets an upper safe limit of k+2l $\leq$ 41, without further testing.

Generating the basis functions was done by the subroutine *LaguerreSub*, which can be found at the top of the *main.f90* program in the code repository. The code generates N number of basis functions based on the recurrence relation outlined above in the manner shown in the below code listing. Please note that lines have been wrapped for readability here are not wrapped in the code.

```fortran
subroutine LaguerreSub(alpha, l, nr, N, rgrid, basis)
        implicit none

        ! initialise alpha, l, dr, rmax, N and others
        integer :: i
        integer, intent(in) :: nr
        integer, INTENT(IN) :: N
        real, INTENT(IN) :: alpha, l
        real, dimension(nr), INTENT(IN) :: rgrid
        real, dimension(nr,N) :: basis

        basis(:,1) = (2.0d0*alpha*rgrid(:))**(l+1) *exp(-alpha*rgrid(:))

        basis(:,2) = 2.0d0*(l+1-alpha*rgrid(:)) * (2.0d0*alpha*rgrid(:))**(l+1)
                                                *exp(-alpha*rgrid(:))

        !generate N laguerre basis using recurrence relation
        do i = 3, N
                basis(:,i) = (2*(i-1+l-alpha*rgrid(:))*basis(:,i-1) -
                                        (i+2*l-1)*basis(:,i-2) )  / (i-1)
        end do

        return
end subroutine LaguerreSub
```

LaguerreSub reads in the alpha, and l parameters, the N number of basis functions to create, the corresponding r values through rgrid, and nr, which represents the number of steps over the range of r. Basis, is a 2D array of dimension nr x N, which holds the basis functions as column vectors.

The first two basis functions are hard-coded, with flexibility for changes in alpha and r. The subsequent basis are generated by the do loop, which uses the recurrence relation to generate the next basis function, and then uses that newly generated basis function to make the next one, and so on until the array is full.

This subroutine is called by *main.f90* after calculating the input arrays and variables. The input parameters for *LaguerreSub* and the basis array are created by main in the following way:

```fortran
program main
        implicit none

        real :: normalise, p
        real :: alpha, l
        real :: dr, rmax
        integer :: N, nr, ier
        integer :: i,j
        real, dimension(:), allocatable :: rgrid
```

```fortran
      real, dimension(:,:), allocatable :: basis

      !add in overlap and hamiltonian
      real, dimension(:,:), allocatable :: H
      real, dimension(:,:), allocatable :: B
      ! energies and expansion coefficients
      real, dimension(:,:), allocatable :: w
      real, dimension(:,:), allocatable :: z
      real, dimension(:,:), allocatable :: V
      ! create array for wavefunctions
      real,dimension(:,:), allocatable :: wf

      !open file location: hard coded for now but could become flexible
      !read stored values into relevent variables

      open(unit=1, file="LaguerreParams.txt", action="read")
      read(1,*) alpha, N, l, dr, rmax
      Print *, alpha, N, l, dr, rmax

      !calculate rgrid params
      nr = rmax/dr + 1
      Print *, nr

      ! based on options from file, \
               !allocate appropriate memory to arrays
      allocate(rgrid(nr))
      allocate(basis(nr,N))
      allocate(H(N,N))
      allocate(B(N,N))
      allocate(V(N,N))
      allocate(w(N,1))
      allocate(z(N,N))
      allocate (wf(nr,N))

      !allocate values to the rgrid
      do i = 1, nr
              rgrid(i) = (i-1)*dr
      end do


      !use recurrence relation to compute the basis functions
      CALL LaguerreSub(alpha, l, nr, N, rgrid, basis)

      ... More ...
```

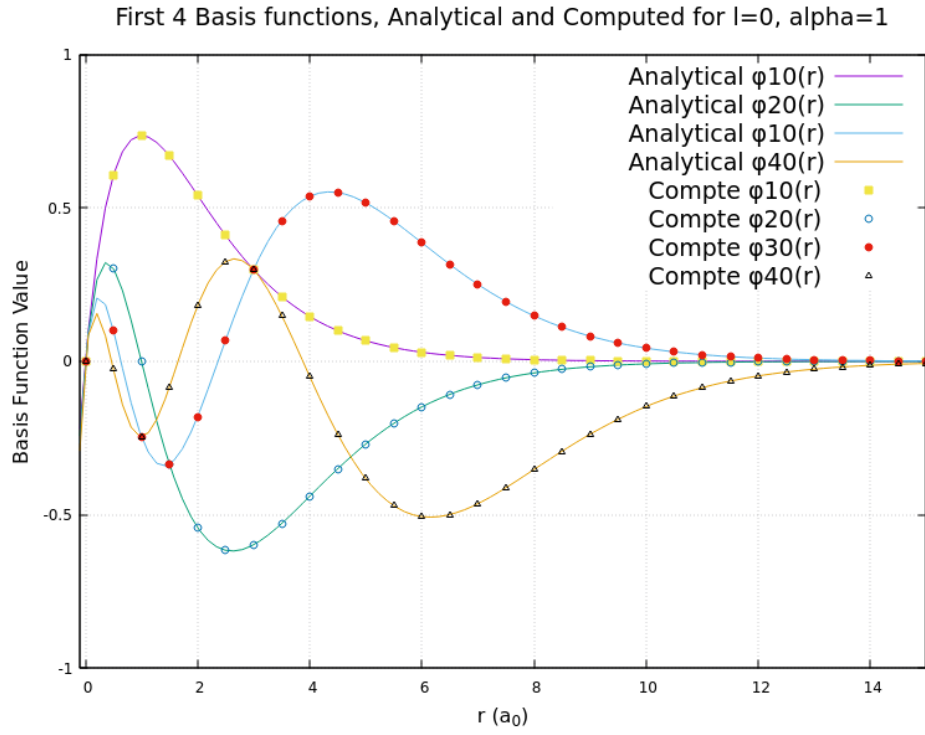Testing the Basis functions generated by the code vs the analytical basis functions using:

Figure 1: First four analytical Laguerre basis functions vs discrete computational Laguerre basis functions, for l=0, α=1
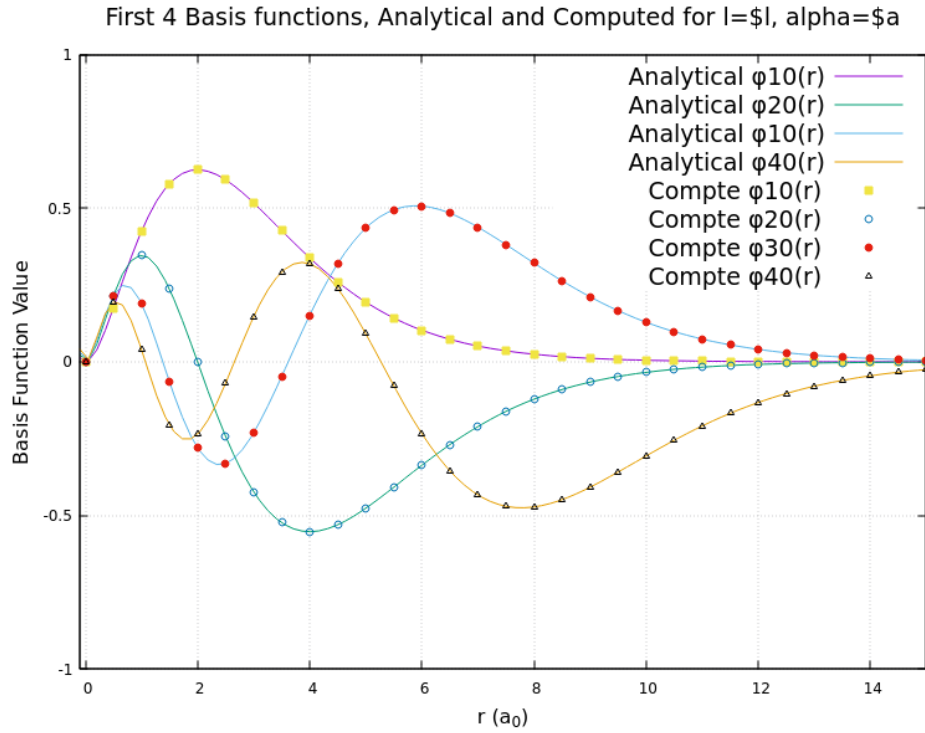

Figure 2: First four analytical Laguerre basis functions vs discrete computational Laguerre basis functions, for l=1, α=1
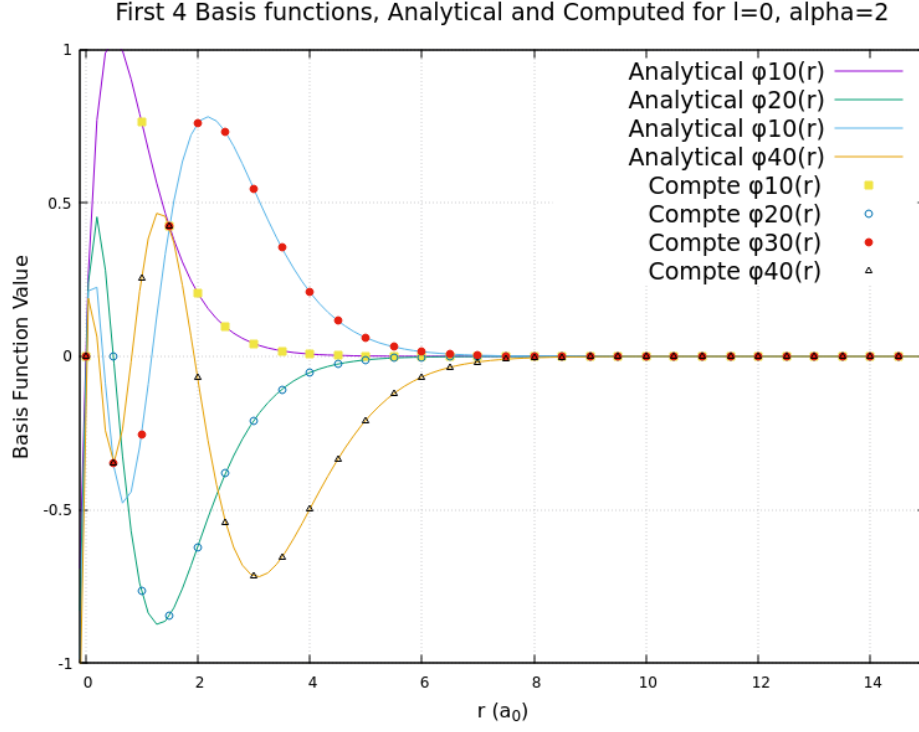
Figure 3: First four analytical Laguerre basis functions vs discrete computational Laguerre basis functions, for l=0, α=2
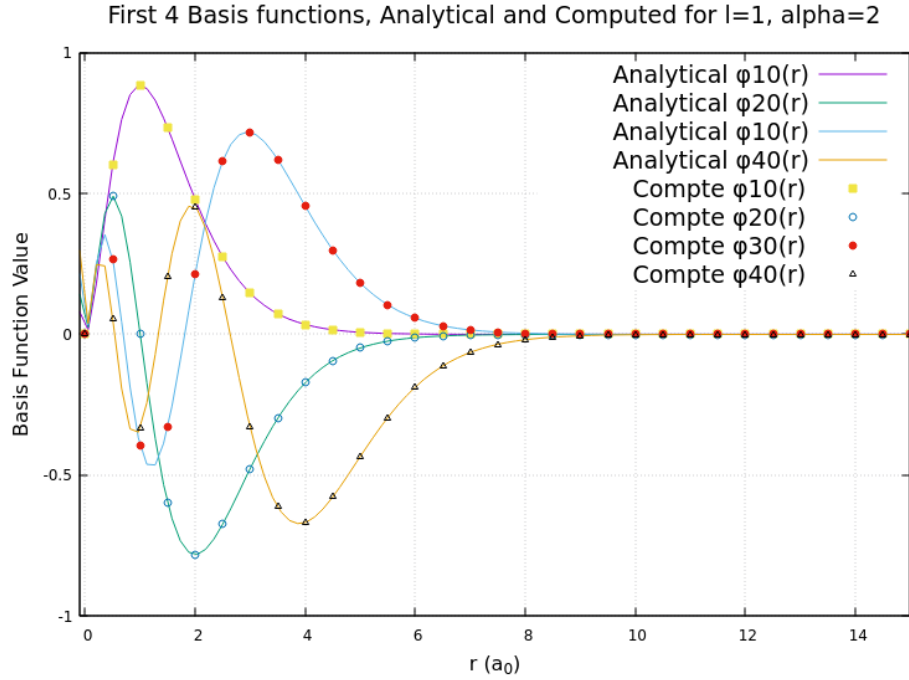

Figure 4: First four analytical Laguerre basis functions vs discrete computational Laguerre basis functions, for l=1, α=2

Increasing α decreases the range of the basis functions such that they asymptotically decay to zero faster, expected as all of the functions $\propto e^{-\alpha r}$ (compare figures 1 and 3). Increasing l for constant α results in a suppression of the

wavefunction at the origin (see figures 1, 3 vs 2, 4). The α parameter may produce more for lower N depending on the energy states of interest, if it allows the basis functions to more quickly approximate the fall-off of the true wavefunction. Similar could be said for smaller alpha. In all cases we see a perfect agreement between the analytical basis functions and the vectorised basis stored in the array.

1.1 Problem 3

Section 1 mentions that the radial component of the Hydrogen wavefunction can be determined by the infinite sum of the appropriate Laguerre based basis functions:

$$|\Phi\rangle = \sum_j c_j \, |\phi_j\rangle \tag{13}$$

If we make this substitution into the Schrodinger equation:

$$\sum_j c_j \mathbf{H} \, |\phi_j\rangle = E \sum_j c_j \, |\phi_j\rangle \tag{14}$$

$$\sum_j c_j \, \langle\phi_i| \, \mathbf{H} \, |\phi_j\rangle = E \sum_j c_j \, \langle\phi_i|\phi_j\rangle \tag{15}$$

As a matrix equation then becomes:

$$\sum_j c_{ji}\mathbf{H_{ij}} = E_i \sum_j \mathbf{B_{ij}}c_{ji} \tag{16}$$

Close approximations to the true Schrodinger equation solution can be determined for a finite sum, up to some limit of the N'th basis function:

$$\sum_j^N c_{ji}\mathbf{H_{ij}} = E_i \sum_j^N \mathbf{B_{ij}}c_{ji} \tag{17}$$

As we increase N, functions in the basis ,we will get higher accuracy to the true functions.

In *main.f90* the provided *rsh.f* function is used to solve this problem, and get approximations to the bound and continuum state hydrogen energy levels and recover the coefficients c_{ij} which can be used in the finite sum:

$$\Phi(r) = \sum_j^N c_{ji}\phi_j(r) \tag{18}$$

To recover the radial wavefunctions.

To do this, the following definitions of $\mathbf{H}, \mathbf{B_{ij}}$ were used:

$$\mathbf{H_{ij}} = \quad \alpha_{l_i}^2 \delta_{ij} - \tfrac{\alpha_{l_i}}{(k_i+l_i)}\delta_{ij} - \tfrac{\alpha_{l_i}^2}{2}\left\langle\phi_{k_i l_i}\middle|\phi_{k_j l_i}\right\rangle \delta_{l_i l_j}\delta_{m_i m_j}$$

$$\mathbf{B_{ij}} = \qquad\qquad \left\langle\phi_{k_i l_i}\middle|\phi_{k_j l_i}\right\rangle \delta_{l_i l_j}\delta_{m_i m_j}$$

Where we also employ the useful definition:

$$\left\langle\phi_{k_i l}\middle|\phi_{k_j l}\right\rangle = -0.5\sqrt{1 - \frac{l(l+1)}{(k_i+l)(k_i+l+1)}} \tag{19}$$

The code implementation of the calculation of the Hamiltonian and overlap matricies, as well as the use of the rsg function is given below. This is a section of code extracted from *main.f90*, with some definitions removed for brevity, but all can be found in the code repository.

```fortran
... See previous code listing for code prior to this

!use recurrence relation to compute the basis functions
CALL LaguerreSub(alpha, l, nr, N, rgrid, basis)

!implement normalisation condition using simplified factorial
do i = 1, N
        p=1.0
        do j = 0, 2*l
                p = real(p*(i+2*l-j))
                !Print *, p, j
        end do
        normalise = sqrt(alpha /((i+l)* p))
        Print *, "Norm:: ", normalise
        basis(:,i) = normalise*basis(:,i)
end do


!write basis to file for plotting

open(1, file='basisout.txt', action='write')
do i =1,nr
        write(1, '(*(f12.8))'), rgrid(i), basis(i,:)
end do
close(1)

!calculate overlap matrix
B = 0.0d0
do i =1, N-1
        B(i,i) = 1.0d0
        B(i,i+1) = -0.5d0*sqrt(1.0d0-( (l*(l+1.0d0)) / ((i+l)*(i+l+1.0d0))))
        B(i+1,i) = B(i,i+1)
end do
B(N,N) = 1.0d0

!compute H-matrix Elements
H = (-alpha**2/2.0) * B
do i =1,N
        H(i,i) = H(i,i) + alpha**2 - (alpha/(i+l))
end do

Print *, H(i,:)

CALL rsg(N,N,H,B,w,1,z,ier)

!recover wavefunctions:
wf = 0.0d0
do i =1,N
        do j = 1,N
        wf(:,i) = z(j,i)*basis(:,j) + wf(:,i)
        !Print *, wf(:,i)
        end do
```

end do

```
open(1, file='wfout.txt', action='write')
do i =1,nr
       write(1, '(*(f12.8))'), rgrid(i), wf(i,:)
end do
close(1)

open(1, file='wout.txt',action='write', access='append')
do i =1,N
       write(1, '(*(f12.8))'), real(N), w(i,1)
end do
end do
```

... deallocation **and end program** ...

We have made use of the Kronecker delta functions where they can reduce the number of loop iterations for values of matricies that are always zero. For example, when defining the matrix for the Hamiltonian, we do not need to iterate over the whole thing, we need only add missing components to the diagonal after copying a modified overlap matrix.

From this method we ca show that for $\alpha = 1, l = 0$, as we increase the nubmer of basis functions, we get a convergence in the Energies of the hydrogen atom to the analytically derived solutions (figures 5 and 6)
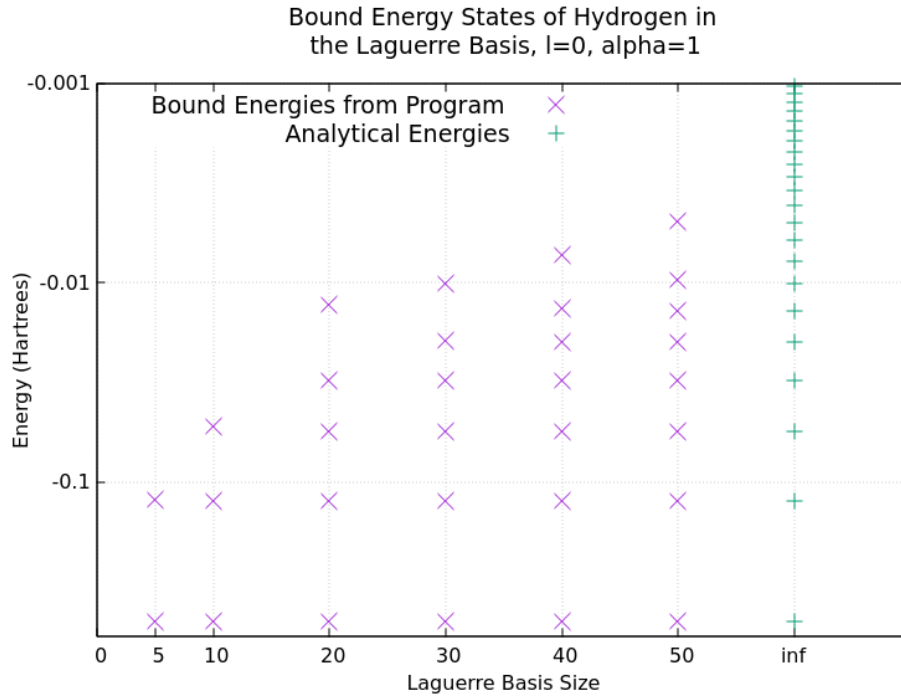

Figure 5: For l=0, α=1, and increasing N, the predicted bound states of hydrogen energy levels slowly converge to the true analytical distribution.
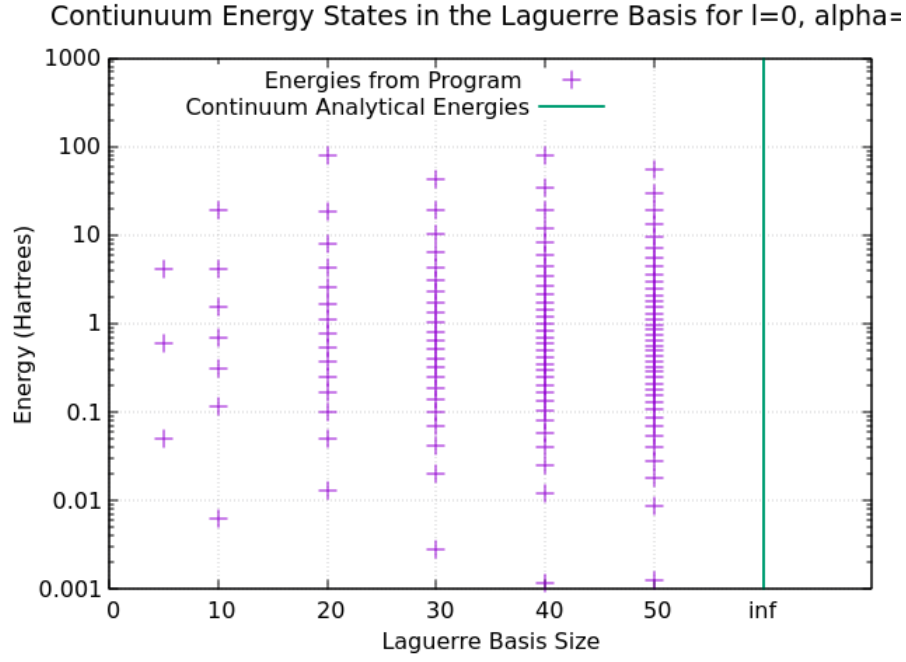
Figure 6: For l=0, α=1, and increasing N, the continuum states of the hydrogen atom slowly tend towards a true continuum.

Figures 5 and 6 demonstrate the convergence of the bound and continuum state energies for increased basis size. Figure 5 shows that the bound state predictions for the energies converge to the true values fairly quickly for the lowest energy levels. For the continuum the predictions tend to fill out faster around $E = 1$, and very large basis size is would be needed to make a good approximation to the continuum for the lowest energies.
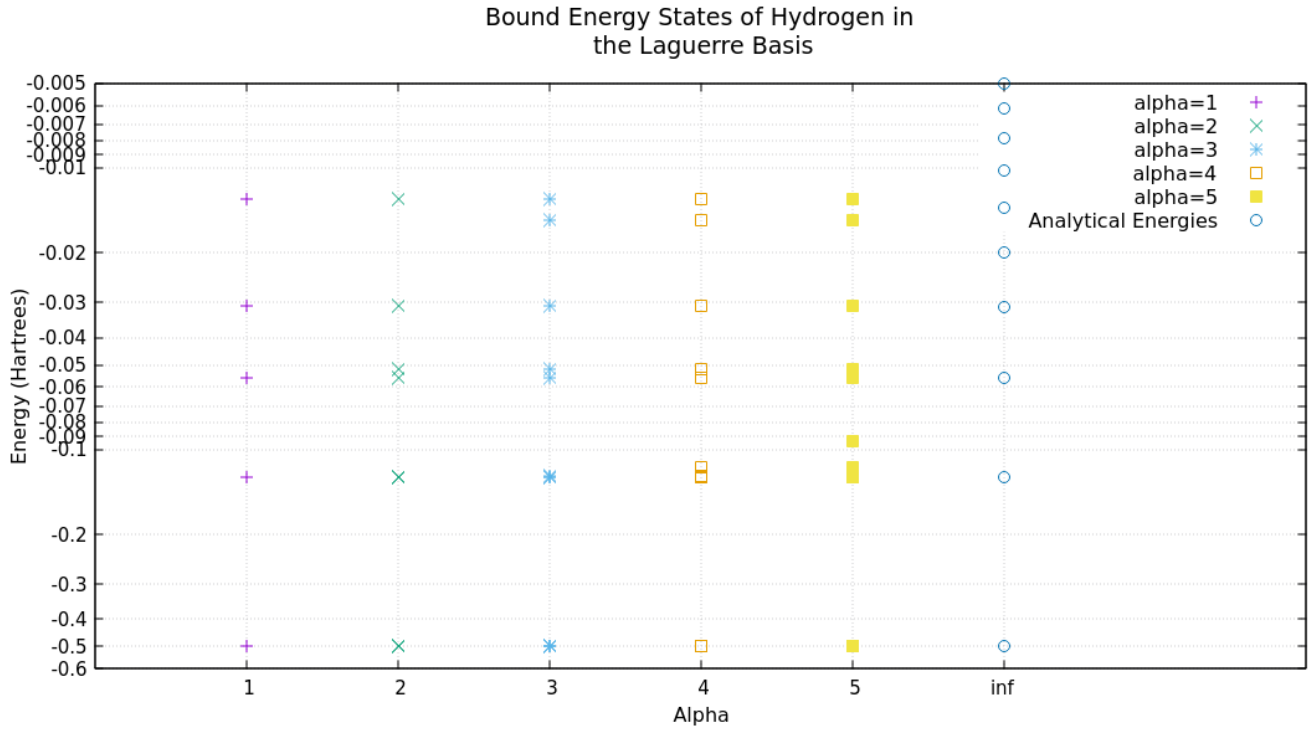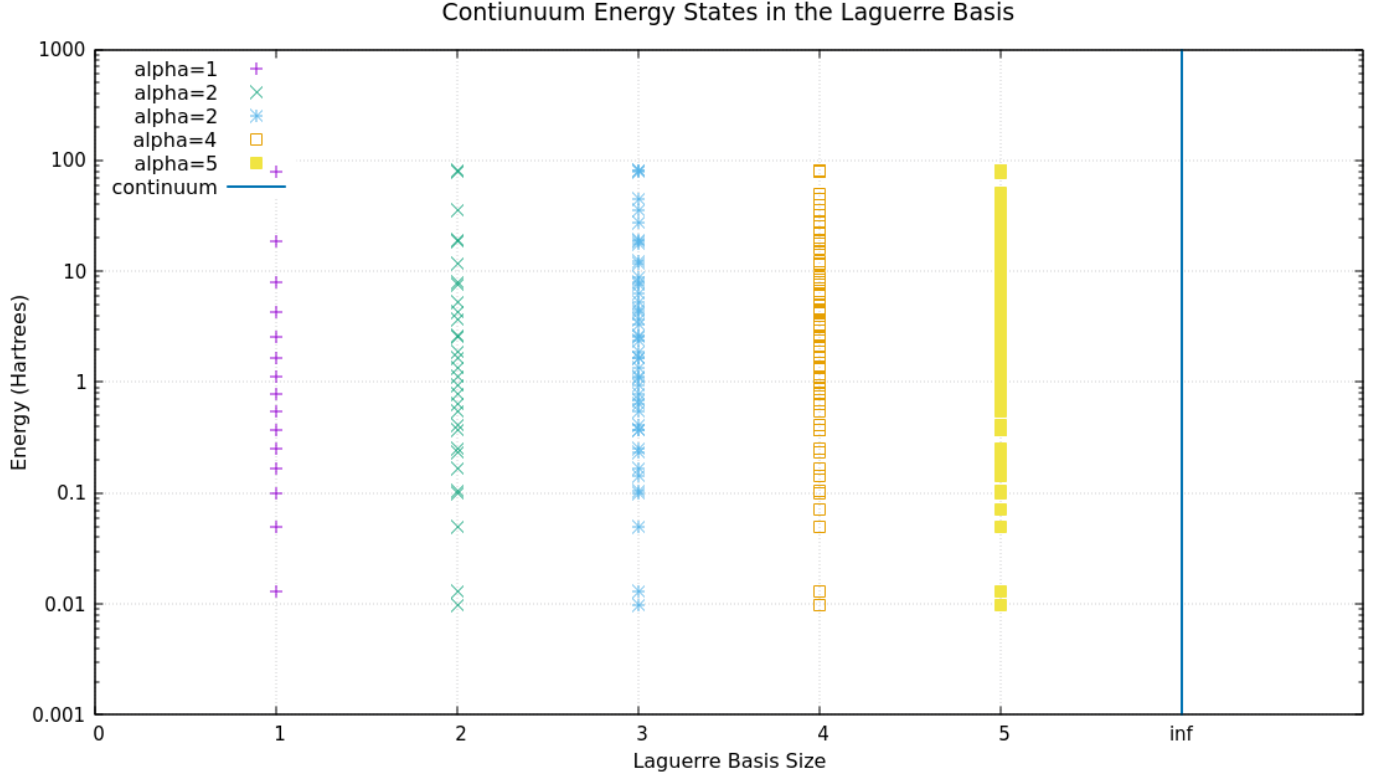

Figure 7: N=20 bound states for varied α.

Figure 8: N=20 continuum states for varied α.

From figures 7 and 8 the increase in α appears to increase the rate that the continuum states are populated but decreases the accuracy of the bound states. For the bound states and limited N=20, the higher the alpha, the more erroneous bound state energy values there are. The choice of α will strongly influence scattering calculations. For collision where the lowest states are expected to be excited, a small α would converge faster, requiring smaller basis arrays, but for collisions where electrons are expected to enter a continuum state, higher alpha might be preferable because it covers the continuum faster for the same basis size.

For N=500, we calculate the first three bound state s-wave and p-wave radial wavefunctions using the finite summation of the basis functions with their coefficients, as previously outlined. These calculated wavefunction vectors are compared to the complete analytical solutions in figures 9 and 10.

We note high agreement between the analytical and calculated solutions, with a key anomaly. For the s-state solutions, the program returns a 2s wavefunction which is the negative of the normal analytical solution, the same is true for the 2p state. These negative solutions are also true solutions of the Schrodinger equation and deliver the same observable outcomes, due to the fact that observables are associated with the square of the wavefunction.
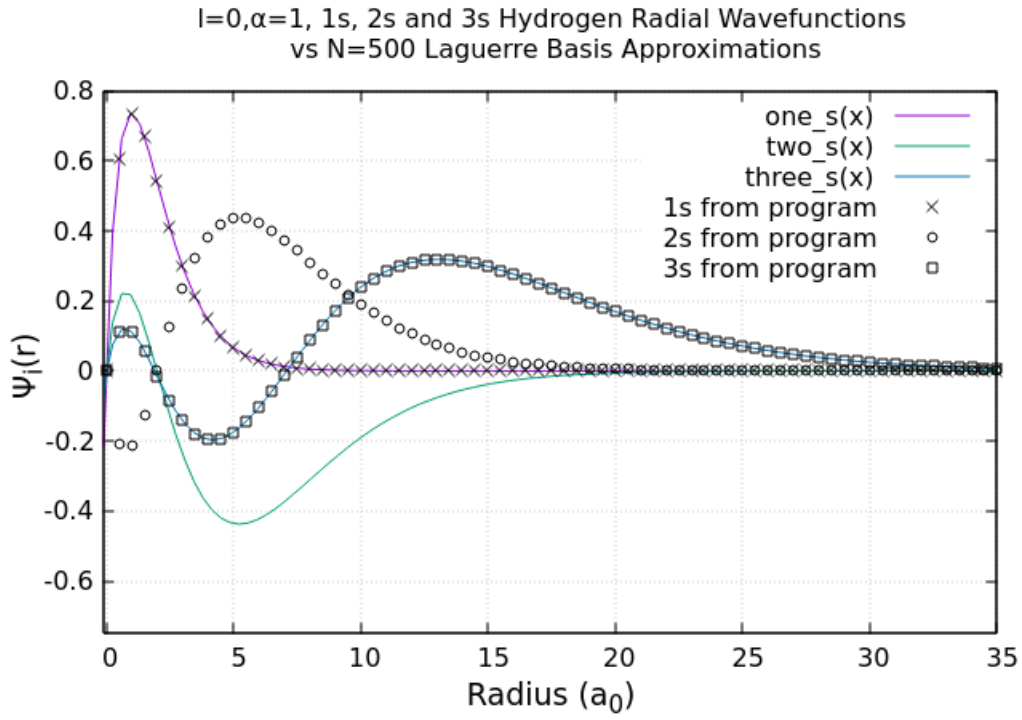
Figure 9: N=500 first three bound state, s-wave, radial wavefunctions of Hydrogen.
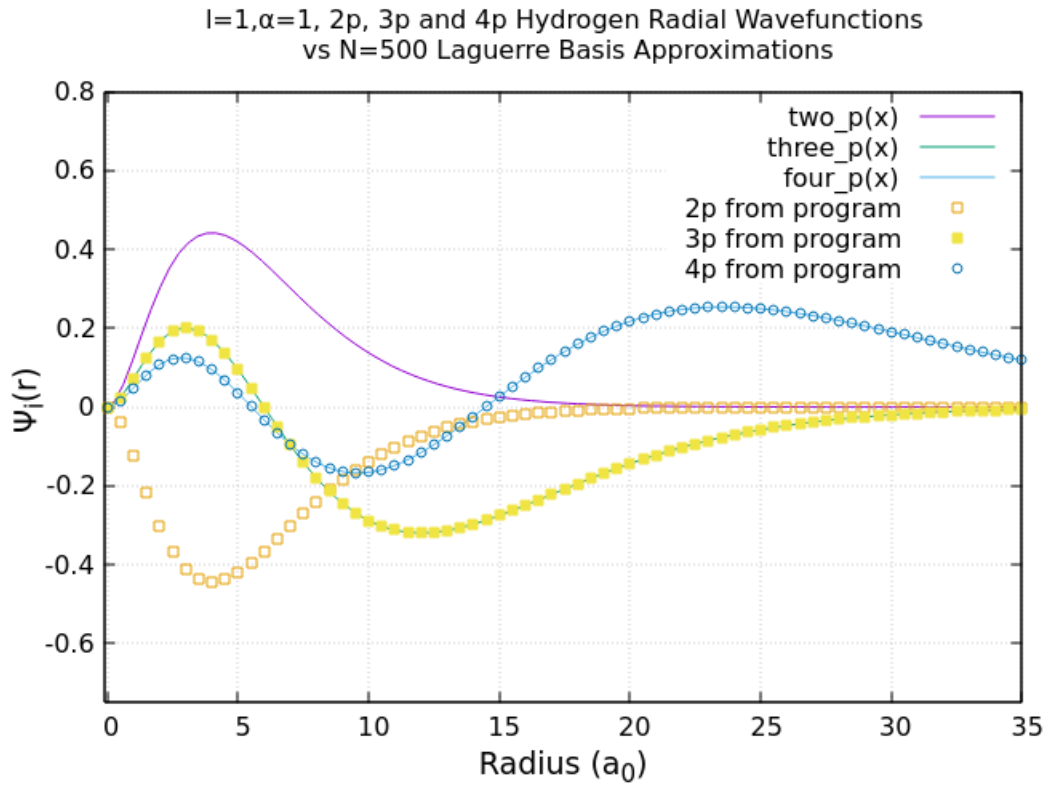

Figure 10: N=500 first three bound state, p-wave, radial wavefunctions of Hydrogen.