Assignment 2

Don Wijesinghe

Q1

The Born-Oppenheimer approximation allows us to separate the electronic and nuclear motions in molecules. For the H_2^+ molecule, we can write the molecular Hamiltonian as:

$$H = H_{elec} + H_{nucl} \tag{1}$$

Where H_{elec} and H_{nucl} are the electronic and nuclear Hamiltonians respectively. The Born-Oppenheimer approximation assumes that the nuclei, being much heavier than the electrons, move much more slowly. Thus, for a given nuclear configuration, the electrons can be considered to adjust instantaneously to the positions of the nuclei. This allows us to write the molecular wavefunction as a product of an electronic part and a nuclear part:

$$\Phi_{nvJ}(\boldsymbol{r_1}, \boldsymbol{r_2}, \dots, \boldsymbol{R_1}, \boldsymbol{R_2}, \dots) = \Phi_n(\boldsymbol{r_1}, \boldsymbol{r_2}, \dots; \boldsymbol{R_1}, \boldsymbol{R_2}, \dots)\Phi^{\mathrm{n}}_{vJ}(\boldsymbol{R_1}, \boldsymbol{R_2}, \dots) \tag{2}$$

Where, $\Phi_n(\boldsymbol{r_1}, \boldsymbol{r_2}, \dots; \boldsymbol{R_1}, \boldsymbol{R_2}, \dots)$ is the electronic wavefunction, which depends on the electronic coordinates r_i and parametric dependence on the nuclear coordinates $\boldsymbol{R}_\alpha$ and $\Phi^{\mathrm{n}}_{vJ}(\boldsymbol{R_1}, \boldsymbol{R_2}, \dots)$ is the nucklear wavefunction. n, v and J are the electronic, vibrational, and rotational quantum numbers, respectively.

For a fixed nuclear position, the electronic schrodinger equation is,

$$H_{elec}\Phi_n = \epsilon_{\mathrm{n}}\Phi_{\mathrm{n}} \tag{3}$$

For the H_2^+ we choose a spherical coordinate system where the z axis is aligned with the two protons and taking the origin in the middle, where you can take r as the distance to the electron from the origin and R as the internuclear distance. Hence, the electronic wavefunctions become functions of r and R. Solving (3) for various nuclear configurations R gives the potential energy surfaces $\epsilon_n(R)$.

We can then write the electron Hamiltonian as,

$$H_{elec} = -\frac{1}{2}\nabla^2{}_{\mathrm{r}} + \mathrm{V(r)} + \frac{1}{R} \tag{4}$$

Where,

$$V(r) = -\frac{1}{\left|r + \frac{R}{2}\right|} - \frac{1}{\left|r - \frac{R}{2}\right|} \tag{5}$$

is the electron-nuclei potential and $\frac{1}{R}$ is the nuclear repulsion potential and $\left|r \pm \frac{R}{2}\right|$ are the distances to the electron from each proton. (5) can be expanded using spherical harmonics as,

$$\mathrm{V(r)} = -2\sum_{\lambda\,\mathrm{even}}^{\lambda\,max}\sqrt{\frac{4\pi}{2\lambda+1}}\frac{r_<^\lambda}{r_>^{\lambda+1}}Y_\lambda^0(\hat{r}), \tag{6}$$

Where, $\mathrm{r}_< = \min\left(\mathrm{r}, \frac{\mathrm{R}}{2}\right)$ *and* $\mathrm{r}_> = \max\left(\mathrm{r}, \frac{\mathrm{R}}{2}\right)$. For the V matrix elements we use the following equation,

$$V_{ij} = -2\delta_{m_i m_j}\delta_{\pi_i \pi_j} \sum_{\lambda \text{ even}} \sqrt{\frac{4\pi}{2\lambda+1}} \int_0^\infty \phi_{ki\ell_i}(r) \frac{r_<^\lambda}{r_>^{\lambda+1}} \phi_{kj\ell_j}(r)\, dr \cdot \left\langle Y_{\ell_i}^{m_i} \middle| Y_\lambda^0 \middle| Y_{\ell_j}^{m_j} \right\rangle \quad (7)$$

Here, $\delta_{\pi_i\pi_j}$, ensures that the electronic parity $\pi = (-1)^l$ is conserved, resulting from selection rules, and we use the same basis functions as assignment 1.

Implementation and results

The basis function for the H_2^+ molecule need to be set up for each pair of magnetic quantum number m and parity π. For a given l_{max} , we include all l values up to l_{max}, subject to selection rules based on m and π.

For example, with l_{max}=3:

For $\pi = +1$

- $l = 0$ $(\pi = +1)$
- $l = 2$ $(\pi = +1)$

For $\pi = -1$

- $l = 1$ $(\pi = -1)$
- $l = 3$ $(\pi = -1)$

The magnetic quantum number, m must satisfy $m \leq l$. For a given pair (m, π), the basis functions are constructed for each valid combination of l and k, where k is the index for our radial basis functions from last assignment.

Example for $l_{max} = 3$:

1. $m = 0, \pi = +1$:
   - Valid l values: 0,2
   - Basis functions:

$$\phi_{k0}(r) \text{ for } l = 0, k = 1,2, \dots, N$$

$$\phi_{k2}(r) \text{ for } l = 2, k = 1,2, \dots, N$$

2. $m = 0, \pi = -1$:
   - Valid l values: 1,3
   - Basis functions:

$$\phi_{k1}(r) \text{ for } l = 1, k = 1,2, \dots, N$$

$$\phi_{k3}(r) \text{ for } l = 3, k = 1,2, \dots, N$$

3. $m = 1, \pi = +1$:
   - Valid l values: 2

- Basis functions:

$$\phi_{k2}(r) \text{ for } l = 2, k = 1,2, \dots, N$$

4. $m = 1, \pi = -1$:
   - Valid l values: 1,3
   - Basis functions:

$$\phi_{k1}(r) \text{ for } l = 1, k = 1,2, \dots, N$$

$$\phi_{k3}(r) \text{ for } l = 3, k = 1,2, \dots, N$$

The program begins by reading parameters $\alpha, l_{max}, dr, r_{max}, N$ and R. The radial coordinates and Simpson's rule weights are set up for numerical integration and we also calculate the maximum value of m and the number of states as seen in line 26.

```
1.      !read input file
2.      open(unit=1, file='input.txt', status='old', action='read')
3.      read(1,*) alpha, lmax, N, dr, rmax, Rn
4.      close(1)
5.
6.      !allocate memory for r
7.      nr = int(rmax/dr) +1
8.      if (mod(nr,2) == 0) nr = nr + 1
9.      print*, "nr=", nr
10.     allocate(r(nr), weights(nr),f(nr))
11.
12.     !create radial coordinates
13.     do i = 1, nr
14.             r(i) = dr*real(i-1)
15.     end do
16.
17.     !weights
18.     weights(1) = 1.0d0
19.     do i=2, nr-1
20.             weights(i) = 2.0d0 + 2.0d0*mod(i+1,2)
21.     end do
22.     weights(nr) = 1.0d0
23.     weights(:) = weights(:)*dr/3.0d0
24.
25.     m_max = lmax
26.     nstates = 2*m_max +1
27.     allocate(energies(nstates))
28.     nstate = 0
```

```
1.          !loop over symmetry pairs
2.          do par=-1, 1, 2
3.                  do m=0, m_max
4.                          write(*,'(A,I2,A,I2)') "par =", par, ", m =", m
5.                         !calculate number of basis functions for each symmetry pair
6.                          num_func = 0
7.                          do l=0, lmax
8.                                  if((-1)**l /=par .or. l<m) cycle
9.                                  num_func = num_func + N
10.                                 write(*,'(A,I1)') "l =",l
11.                         end do
12.
13.                         write(*,'(A,I2)') "num_func=",num_func
14.
15.                         if (num_func == 0) then
16.                                 write(*,'(A, I1)') "skipping m=", m
17.                                 cycle        !for cases when num_func == 0 cycle to next m
18.                         end if
19.
20.                         !keep track of indexes in basis function matrix
21.                         allocate(basis_fns(nr,num_func), k_list(num_func), l_list(num_func))
22.                         i=0
23.                         do l=0, lmax
24.                                 if((-1)**l /= par .or. l < m) cycle
25.                                 do k=1, N
26.                                         i = i + 1
27.                                         k_list(i) = k
28.                                         l_list(i) = l
29.                                 enddo
30.                         enddo
31.                         !calculate basis functions
32.                         do i=0, num_func/N-1
33.                                 call calc_phikl (alpha, l_list(N*i+1), N, r, basis_fns(:,N*i+1:N*i+N))
34.                         end do
```

We iterate through each symmetry pair to calculate our basis functions and calculates the Hamiltonian matrix elements for each pair to solve the eigenvalue problem similar to what we did in assignment1. We note that the negative values of m are ignored since these come in degenerate pairs with their positive value, hence giving us the same energies for these states as these correspond to same orbitals but opposite orientation in space. The program generates basis functions for different l values that belong to a given symmetry pair as shown before. It calculates the number of basis functions for each symmetry pair and keeps track of indexes in the basis function matrix by storing indexes in l_list and k_list so they can be accessed later. If we encounter values of l that do not belong to to a symmetry pair ie, num_func is equal to 0, we cycle to the next symmetry pair. We then generate our basis functions for each l and store it in our basis function matrix using the same subroutine we used in assignment 1 as seen in line 33.

In the following page we see the code for calculating the Hamiltonian, overlap, and potential matrices. We first calculate the overlap matrices, however now the B matrix is block tridiagonal with each block corresponding to a different value of l. we utilise the our k_list and l_list arrays to calculate the diagonal, super and sub diagonals for each block. We then calculate our V matrix using (7). We first iterate through i and j and then iterate through even values of λ choosing a maximum value of $\lambda_{max} = 2 * l_{max}$ to calculate the sum of integrals. Within this loop we evaluate the integrand in (7) and then in the following step we use simpson's rule to numerically integrate it over all values of our r grid. The V matrix element is calculated using the integral and the angular part evaluated by the YINT function, which calculates the spherical harmonic integral. We then combine the overlap and potential matrices to form the Hamiltonian matrix (lines 51 -59)

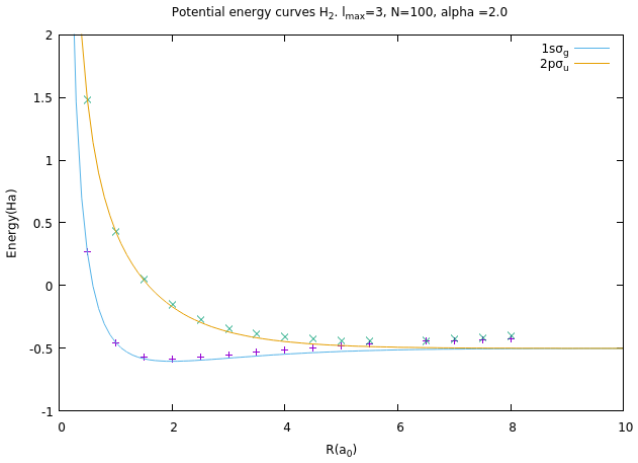
```
                        !allocate H, V, B matrices
                        allocate(H(num_func,num_func), V(num_func,num_func), B(num_func, num_func))

                        B = 0.0d0
                        do i=1, num_func-1
                                B(i,i) = 1.0d0
                                l = l_list(i)
                                k = k_list(i)
                                if(l_list(i+1) /= l) cycle
                                B(i,i+1) = -0.5*SQRT(1.0-l*(l+1.0)/((i+l)*(i+l+1.0)))
                                B(i+1,i) = B(i,i+1)
                        end do
                        B(num_func,num_func) = 1.0d0

                        print*, "B"
                        do i=1, num_func
                                do j=1,num_func
                                        write(*, '(F10.4, 1X)', advance='no') B(i,j)
                                end do
                                print*
                                print*
                        end do

                       !V matrix calculation
                       V = 0.0d0
                       do i=1, num_func
                                do j=1, num_func
                                        li = l_list(i)
                                        lj = l_list(j)

                                        do lamd=0, 2*lmax, 2

                                                f = basis_fns(:,j) * min(r(:), Rn/2.0d0)**lamd/max(r(:),Rn/2.0d0)**(lamd+1) * basis_fns(:,i)
                                                integral = sum(f(:)*weights(:))
                                                V(i,j) = -2.0d0*integral*yint(dble(li),dble(m),dble(lamd),0.0d0,dble(lj),dble(m)) +V(i,j)
                                        end do

                                end do
                        end do


                        print*, "V"
                        do i=1, num_func
                                do j=1,num_func
                                        write(*, '(F10.4, 1X)', advance='no') V(i,j)
                                end do
                                print*
                        end do


                        !initialise H matrix
                        H = -0.5*alpha**2 * B


                        do i=1, num_func
                               H(i,i) =  alpha**2 + H(i,i)
                        end do

                        H = H+V


                        print*, "H"
                        do i=1, num_func
                                do j=1,num_func
                                        write(*, '(F10.4, 1X)', advance='no') H(i,j)
                                end do
                                print*
                        end do

                        !calculate energies for each symmetry
                        allocate(z(num_func,num_func),w(num_func))

                        !solve eigenvalue problem

                        call rsg(num_func,num_func,H,B,w,1,z,ierr)
                        nstate = nstate +1
                        energies(nstate) = w(1)


                        deallocate(k_list, l_list, basis_fns, H, B, V,z,w)
                        print*, "--------------------------------------------------------------------------------"

                end do
        end do
```

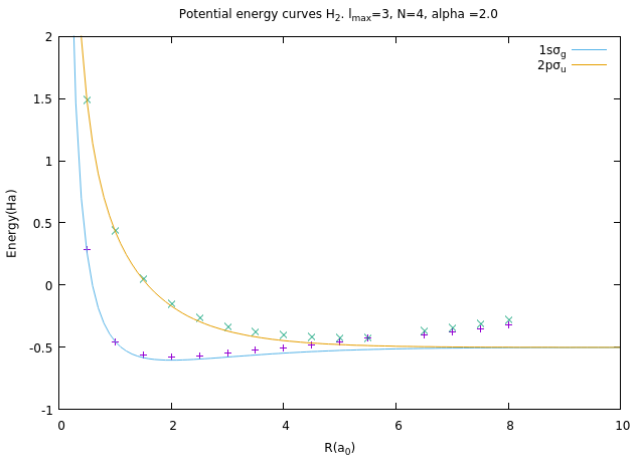
Similar to assignment we solve the eigenvalue problem to obtain the eigen energies and eigenvectors using the *rsg* subroutine. The number of energies obtained will be equal to num_func for each symmetry. We will store the lowest energies into the energies array, as we don't require all energies. After the program iterates through each

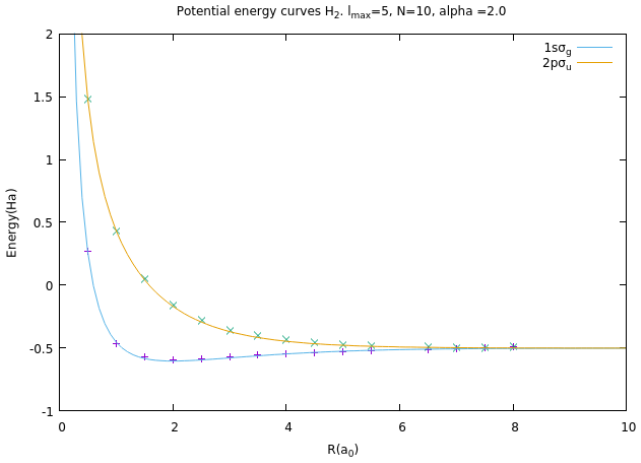
symmetry, we would have obtained *nstates* number of energies. We then sort these energies and write these to a file called *Energies.txt* keeping in mind that we need to add the 1/R nuclear repulsion energy. This term is included as a constant shift to the potential energy (see (4)), which does not affect the electronic wave functions but shifts the electronic energy levels.

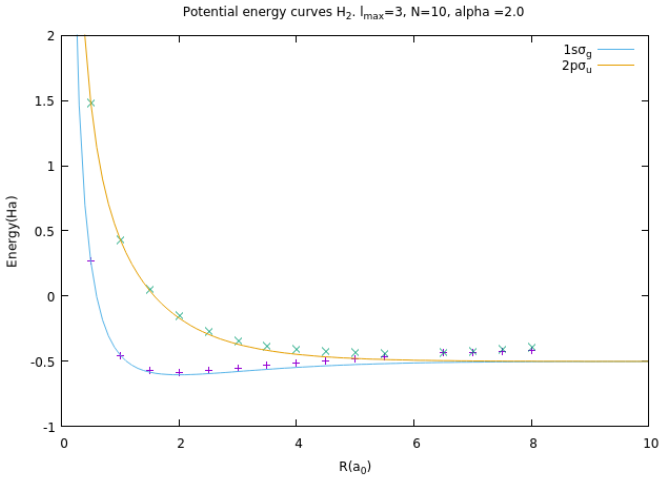
```
1.          sorted = .false.
2.          do while(.not.sorted)
3.                  sorted = .true.
4.                  do i=1, nstates-1
5.                          E1 = energies(i)
6.                          E2 = energies(i+1)
7.                          if(E2 < E1) then
8.                              sorted = .false.
9.                              temp = energies(i)
10.                             energies(i) = energies(i+1)
11.                             energies(i+1) = temp
12.                         end if
13.                 end do
14.         end do
15.
16.         file_name = "Energies.txt"
17.
18.        open(unit=11, file= file_name, position='APPEND')
19.        !write energy states to file
20.        write(11,'(F10.4)',advance='no') Rn
21.        do i=1, nstates
22.
23.                 write(11, '(1X, F10.7)',advance='no')  energies(i)+1/Rn
24.
25.        end do
26.        close(11)
```

We then use a script to calculate energies for different values of R to obtain our potential energy curves to be used for q2. The potential energies for the first two electronic states are shown below comparing it to a more accurate result shown by the line curves.









The above graphs illustrate the behaviour of the potential energy curves when changing l_{max} and N values. Increasing l_{max}, which includes higher angular momentum states, improves the flexibility of the basis set in describing the electron distribution around the nuclei, leading to more accurate potential energy curves. Increasing N the number of radial basis functions, improves the resolution of the radial part of the wavefunction, however increasing it to even higher values does not increase accuracy of the potential energy functions and only exhibit accuracy at shorter internuclear distances.

Q2

Once we obtain the potential energy curves from solving the electronic schrodinger equation, we can then use the nuclear Schrodinger equation for the vibrational wave functions to solve for the bound vibrational wavefunctions and their energies, where μ is the reduced mass of the molecule.

$$\left[-\frac{1}{2\mu}\frac{d^2}{dR^2}+\epsilon_n(R)\right]\Phi_v^n(R)=\epsilon_v^n\Phi_v^n(R) \tag{8}$$

Implementation and results

We use the accurate potential energy curve for the first electronic state of the H_2^+molecule to calculate the vibrational wavefunctions and molecular energies using (8).

The accurate $1s\sigma_g$ potential-energy curve read from the `PEC.1ssg` file is interpolated into the r grid in the code from assignment 1 using the `intrpl` subroutine.

```
1.
2. !read data and interpolate data
3.         file_name = 'PEC.1ssg'
4.         num_points=0
5.         open(unit=10, file=file_name, status='old', action='read', iostat=ierr)
6.         if (ierr /= 0) then
7.                 print *, 'Error opening file ', file_name
8.                 stop
9.         endif
10.        do
11.                read(10, '(A)', iostat=ierr) line
12.                if (ierr /= 0) exit
13.                !count number of data
14.                num_points = num_points + 1
15.        end do
16.        close(10)
17.
18.        allocate(Vin(num_points), Rin(num_points), Vout(rn),weights(rn))
```

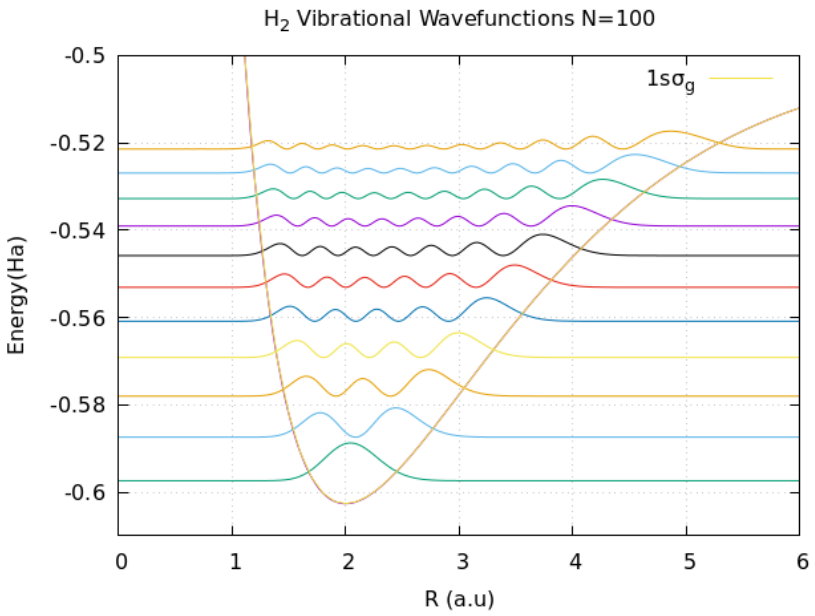
```
        i=0
        open(unit=10, file=file_name, status='old', action='read', iostat=ierr)
        if (ierr /= 0) then
                print *, 'Error opening file ', file_name
                stop
        endif
        do
                read(10, '(A)', iostat=ierr) line
                if (ierr /= 0) exit
                i = i + 1
                read(line, *) Rin(i), Vin(i)
        end do
        close(10)

        !interpolate data
        Vout=0.0d0
        call intrpl(num_points, Rin, Vin, rn, r, Vout)
        deallocate(Vin, Rin)
```

The major change done to the code is that we calculate the V matrix using the interpolated potential energy curve using the equation below, to calculate the Hamiltonian so that the eigenvalue problem can be solved to obtain wavefunctions and energies.

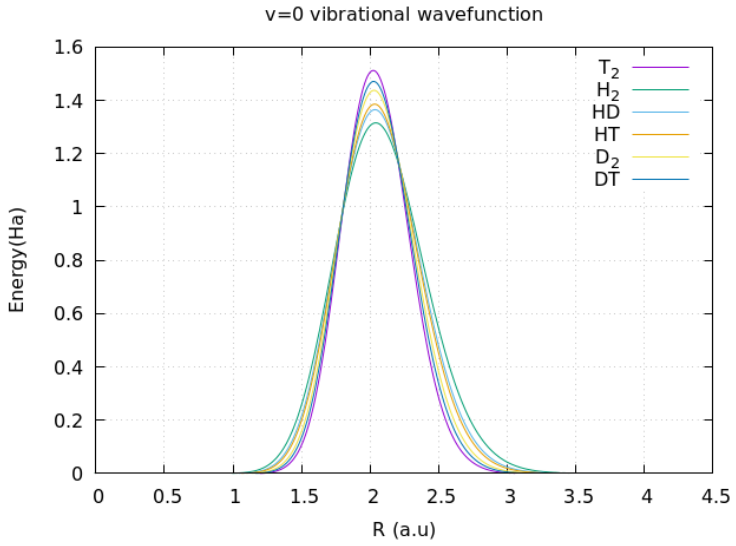
$$V_{ij} = \int \varphi_{k_i l_i}(R)\epsilon(R)\varphi_{k_j l_j}(R)\, dR$$

This is shown by the code below where we use simpson's rule for integration to evaluate the V matrix. We also have to divide the kinetic energy terms by the extra μ term.

```
allocate(B(N,N),H(N,N),V(N,N),w(N),wf(rn,N),z(N,N))

        B = 0.0d0


        do i=1, N-1
                B(i,i)=1.0d0
                B(i,i+1) = -0.5*SQRT(1.0-l*(l+1.0)/((i+l)*(i+l+1.0)))
                B(i+1,i) = B(i,i+1)
        end do
        B(N,N) = 1.0d0

        !calculate V matrix
        V = 0.0d0
        allocate(integrand(rn))
        integrand = 0.0d0
        do i=1, N
                do j=1, N
                        integrand = basis_fns(:,i) * Vout(:) * basis_fns(:,j)
                        V(i,j)= sum(integrand(:)*weights(:))
                end do
        end do
        !print Vmatrix
        print*, 'V'
        do i=1, N
                do j=1, N
                        write(*, '(F10.4, 1X)', advance='no') V(i,j)
                end do
                print*
        end do

        H = -0.5*alpha**2 * B / mu

        do i=1, N
                H(i,i) = alpha**2 / mu + H(i,i)
        end do

        H = H+V
```



The scaled bound vibrational wave functions are plotted by shifting them vertically with their energies. The potential energy curve is also plotted. The restoring force that drives the oscillatory motion is a consequence of the molecular bond, which acts like a spring. As the number of nodes increases with higher vibrational states, there are more points along the internuclear distance where the probability density is zero. Higher vibrational states correspond to higher energy levels. As energy increases, the nuclei move more vigorously and explore a larger range of positions. The wavefunctions spread out over a wider range of internuclear distances. The nodes can be thought of as analogous to the points where a classical oscillator momentarily comes to rest before changing direction. The increased number of nodes indicates that the nuclei spend more time oscillating between these high-probability regions.



We also compare the first vibrational state of different Isotopologues, where H_2^+ has the least reduce mass. All the wavefunctions peak around the same bond distance (approximately 2 a.u.), indicating that the equilibrium bond length is nearly the same for all Isotopologues. The heavier Isotopologues have narrower wavefunctions with higher peaks compared to the lighter ones. This broader distribution for lighter reduced mass means that the probability density is spread over a larger range of internuclear distances. The lighter molecules have higher vibrational frequencies and the zero point energies for the lighter molecules are higher compared to the heavier

ones in the same state resulting the nuclei to explore larger distances making heavier molecules more tightly bound.