Computational Assignment 2: Solving Molecular Vibrational Wavefunctions

Ryan Craft

May 28, 2024

I put all the code listings for this one at the end. Also, completed destroyed the commit history of the previous github, so the new repo is: https://github.com/Ryan-Craft/ACQM2.git, with these code snippets inside of *comp3*.

1 Problem 1: Quantum Harmonic Oscillator

Note: There is some kind of issue with dr=0.5 in the code now. Its quite happy to perform all the calculations at 0.1 or lower but gets stuck on 0.5. I did most of the tests before leaving and coming back to finish them of a week or so later, and while writing this report up have found this bug. I dont know if it was some subtle change I made that I forgot to write down or what, but I have other assessments that I need to finish off so my table for the convergence will have dr=0.1, 0.01, 0.001 instead so that I can still try and understand the trend. My bad.

Particles trapped in the harmonic potential

$$V(r) = \frac{1}{2}\omega^2 r^2 \tag{1}$$

can be expressed with the Schrodinger equation in the ordinary way

$$\frac{1}{2}\frac{d^2}{dr^2}\psi(r) + V(r)\psi(r) = E\psi(r) \tag{2}$$

We can solve the schrodinger equation using finite difference through Numerovs method where

$$\frac{d^2\psi(r)}{dr^2} = g(r)\psi(r) \tag{3}$$

Then set $g(r) = 2(V(r) - E)$, the discrete representation of the wavefunction can be given by the recurrence relation

$$\psi_{i+1} = \frac{2(1 + \frac{5\delta x^2}{12}g_i)\psi_i - (1 - \frac{\delta x^2}{12}g_{i-1})\psi_{i-1}}{1 - \frac{\delta x^2}{12}g_{i+1}} \tag{4}$$

The implementation of this iteration is both forwards and backwards, and has been defined in 2.1 and 2.2 respectively. This section is implemented in *NumerovsQHO.f90*.

This method requires that the energy of the wavefunction is known. In the QHO we know exactly the energy levels; $E_n = \hbar\omega(1/2 + n)$, but in this exercise the energy will be recursively determined recursively with the following procedure:

1. Guess a pair of energies, E_{min} and E_{max}
2. Use $E = (E_{min} + E_{max})/2$

3. apply Numerov approximation from the left and right side of the range of r

4. count the number of nodes in the wavefunction

   if nodes $<$ n, $E_{min} = E$ and perform approximation again

   if nodes $>$ n, $E_{max} = E$ and perform approximation again

   if nodes = n, E is in the ballpark of the true energy

This procedure is not accurately predicting the energy, rather its "narrowing it down" and forcing the wavefunction to have the right number of nodes, its implementation is seen in 2.3.

After each generation of the wavefunction via Numerov's method, a suitable midpoint is selected (one that has non-zero wavefunction), and the wavefunctions are divided by their value at the common midpoint, allowing them to be joined continuously there. The midpoint is labelled x_m in the code.

The wavefunctions are joined at the midpoint, which should represent the wavefunction across the whole range of r values.

When applying the Numerov approximation, we set the boundary conditions for the forwards and backwards version as $\psi = 0$ at either side of the r-axis and the elements $\psi_1 = s, \psi_{N-1} = s$. Where s is some small value. I chose $s = 0.0001$ as suggested.

Once the correct number of nodes has been generated, the numerov forwards and backwards is applied again. This time after every generation, we apply the Cooley energy correction using the midpoint:

$$\Delta E \approx \frac{\psi_m}{\sum_{i=1}^{N} |\psi_i|^2} \left[-\frac{1}{2} \frac{Y_{m+1} - 2Y_m + Y_{m-1}}{\delta x^2} + (V_m - E)\psi_m \right] \tag{5}$$

The cooley correction is added to the energy and the wavefunction recalculated, this process is repeated until the cooley correction is smaller than some predetermined value. In this case $e_{lim} = 10^{-12}$ is the margin. The same midpoint corrections are made and single function is formed (see 2.4 for implementation).

As this represents our final approximation to the wavefunction, it needs to be normalised. The normalisation function for the wavefunction is

$$\psi_{norm} = \frac{1}{\sqrt{\langle \psi | \psi \rangle}} \psi \tag{6}$$

Normalisation is calculated as

```
norm = 1/sqrt(sum(psi(:)**2 * weights(:)))
Print *, norm
psi = psi*norm
```

Producing the wavefunction in this way gives the following outcomes for the energy levels n=0,1,2,3,4, the wavefunctions in the potential are illustrated in that order for figures 1 to 4.

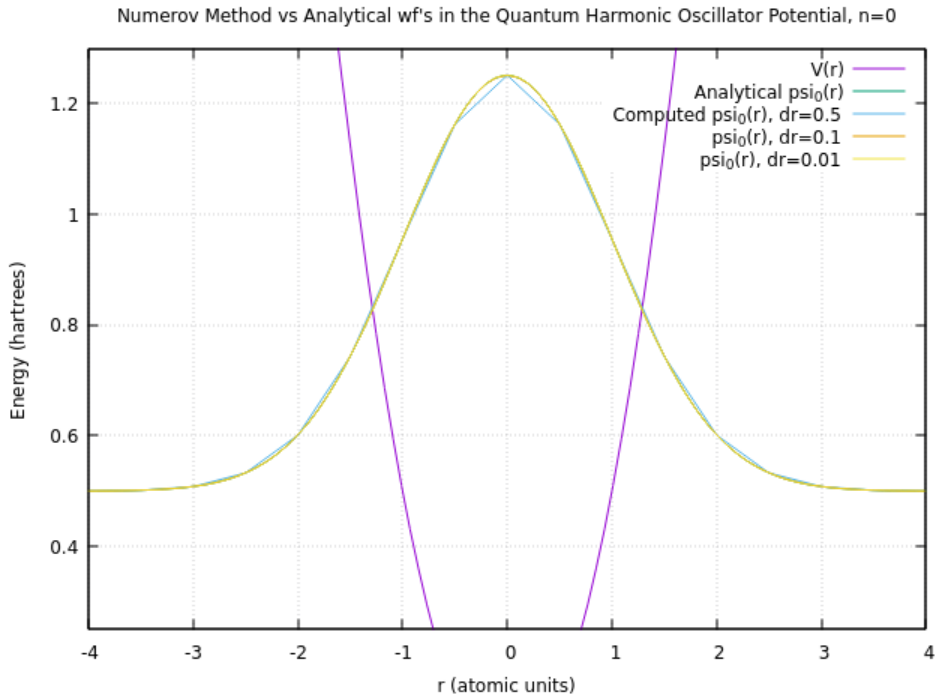

Figure 1: n=0 QHO wavefunction analytical vs dr=0.5,0.1,0.01 grid spacing for the computational values.

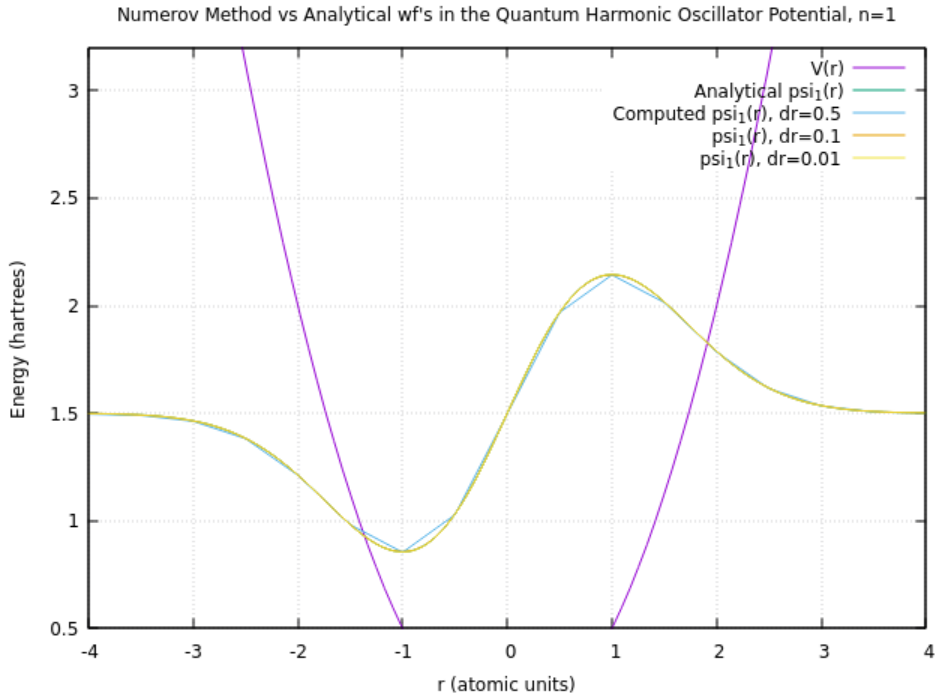

Figure 2: n=1 QHO wavefunction analytical vs dr=0.5,0.1,0.01 grid spacing for the computational values.

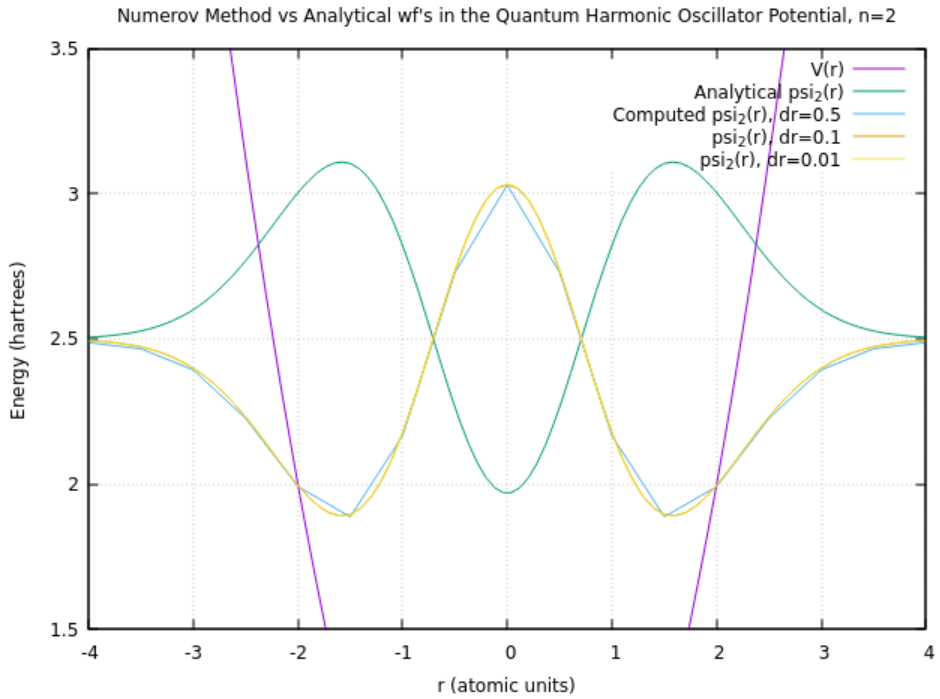

Figure 3: n=2 QHO wavefunction analytical vs dr=0.5,0.1,0.01 grid spacing for the computational values.

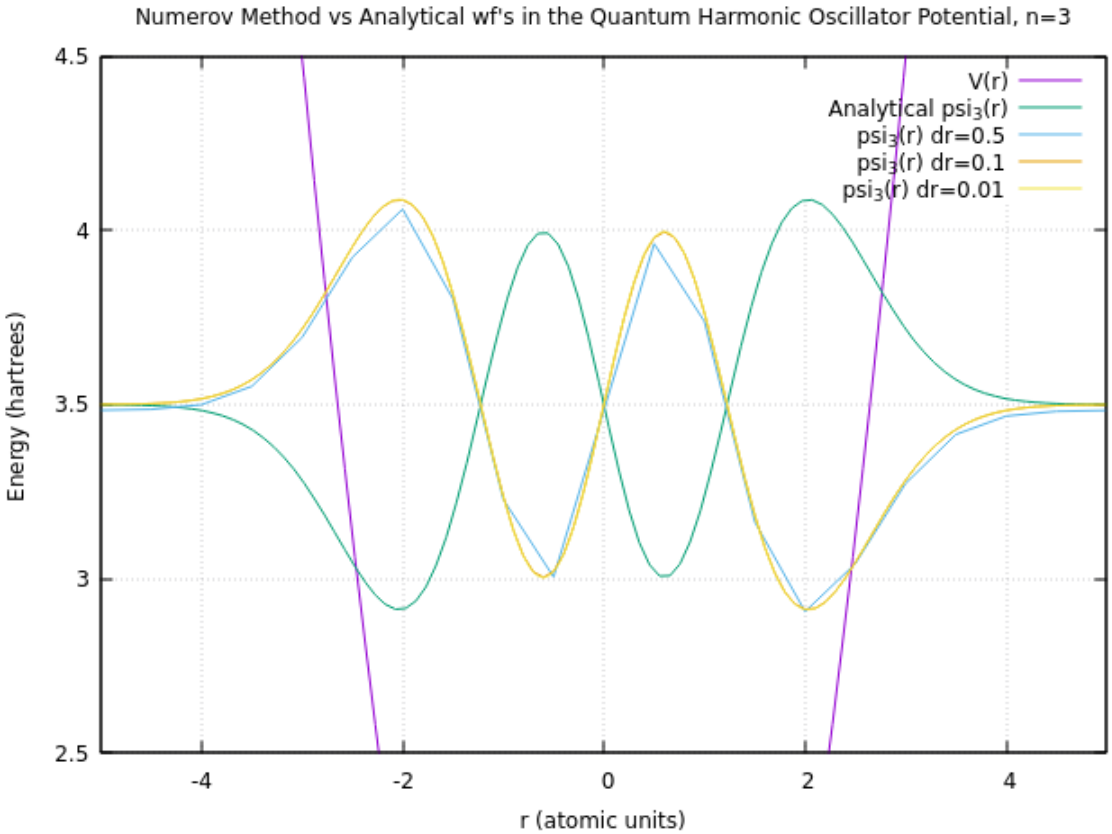

Figure 4: n=3 QHO wavefunction analytical vs dr=0.5,0.1,0.01 grid spacing for the computational values.

The calculated energies of the wavefunctions for different energy levels and grid spacings are given in table 1

Table 1: Calculated energies for $e_{lim} = 10^{-12}$, tested dr, and n=0-3. Given to same precision as the code to see small differences in the output.

	dr=0.5	dr=0.1	dr=0.01	Analytical
n=0	0.49974711028260788	0.49999968333074568	0.50000000419848212	0.5
n=1	1.4982089671345686	1.4999972647164823	1.4999999999057003	1.5
n=2	2.4935052012437571	2.4999901984807384	2.4999999967839908	2.5
n=3	3.4833258359458590	3.4999753371765463	3.4999999986659764	3.5

Examining table 1 shows that the first energy level is accurate to the 4th decimal place, which for an relatively course grain grid of 0.5 seems reasonable. Appreciable differences between the analytical and calculated values at the $dr = 0.5$ level only begin to appear at the n=3 energy state. The same can be said for the figures. In figure 4 the wavefunctions don't line up in the asymptotic regions, meaning that the energy is deviating from the analytical value, and it can be seen somewhat that the values converge to the analytical function value as the grid spacing is decreased.

Also, the computed outputs are negative. This has been discussed in a previous report. Negative solutions to the schrodinger equation are equally valid solutions to the positive ones.

The $dr = 0.5$ output wavefunctions are also plot extremely inaccurately with gnuplot due to the liner interpolation between data points. It is surprising how close the energies are reported considering how poorly it seems to fit the analytical solution.

Table 2: Calculated energies for $e_{lim} = 10^{-12}$, tested dr, and n=0-3. Given to same precision as the code to see small differences in the output.

	dr=0.1	dr=0.01	dr=0.001
n=0	7	6	7
n=1	7	6	7
n=2	7	6	7
n=3	7	6	7

Table 2 shows the number of node recounts + Cooley corrections before reaching convergence, ie cooleys correction was less than the limit set. Out of these, it was expected that the node counting would take more steps, but in fact it seems that the node counding for the first four levels takes only one step, which seems to indicate that the initial guess and numerov approximation begins immediately with the correct form.

Then the cooley corrections seem to take very little time. They reach the limit very fast, within usually 5 steps. It was noted that limits below 10^{-15} would tend to cause the loop to go infinitely, as the energy correction seemed to be only accurate down to that level, so it would never report below a certain threshold. Even so, the accuracy gets very high.

1.1 Problem 2: Dissociative Wavefunctions

Dissociation wavefunctions of the H_2^+ molecule have an energy $E = E_k + D$, where D is the energy of the potential energy curve created by the electron. In this case the $D = \epsilon(\infty)$, which is -0.5 Hartrees. E_k represents the kinetic energy of the proton released from the molecule. Our Schrodinger equation we want to solve is:

$$\left[-\frac{1}{2\mu}\frac{d^2}{dR^2} + \epsilon(R) - E\right]\nu(R) = 0 \tag{7}$$

In the asymptotic limit, the free particle will have plane wave solutions and a normalisation coefficient $\sqrt{(2\mu/k\pi)}$, where $k = \sqrt{(2\mu E_k)}$. Before this normalisation factor is applied, the waves in the asymptotic region are given unit amplitude, so that the normalisation gives them the correct asymptotic amplitude.

The dissociation event we investigate is the result of the electronic transition from the 1ssg state to the 2psu state. This change in potential causes immediately dissociation of the molecule because the 2psu potential has no bound wavefunctions.

Using the code from previous assignments, the 1ssg and 2psu states were read in and the cross section as a function of the kinetic energy release is calculated. The code calculated the bound states in the 1ssg potential and their overlap with the free particles in the 2psu state using the Frank Condon Approximation:

$$\frac{d\sigma_{f,iv_i}}{dE_k} \approx |\langle v_{fE_k}|v_{iv_i}\rangle|^2 \tag{8}$$

The Frank-Condon approximation is calculated numerically.

The first few dissociative wavefunctions and their energies are on figure 5.

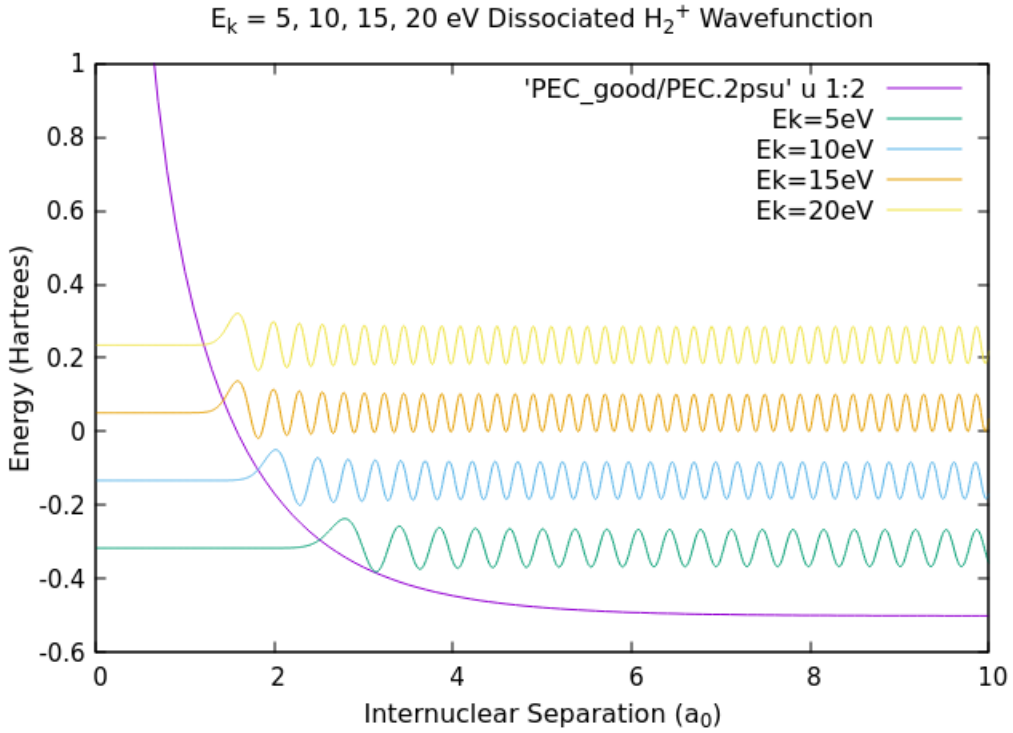

Figure 5: 4 dissociated waves in the 2psu potential. Note that they all become continuum waves when far enough away from the potential.

The above figure is relatively intuitive even if the position on the y axis of the wavefunctions doesn't change, because the wavelength is shorter for higher energies. That is to be expected for the particles De Broglie wavelength as energy increases.

The KER distribution is given below.

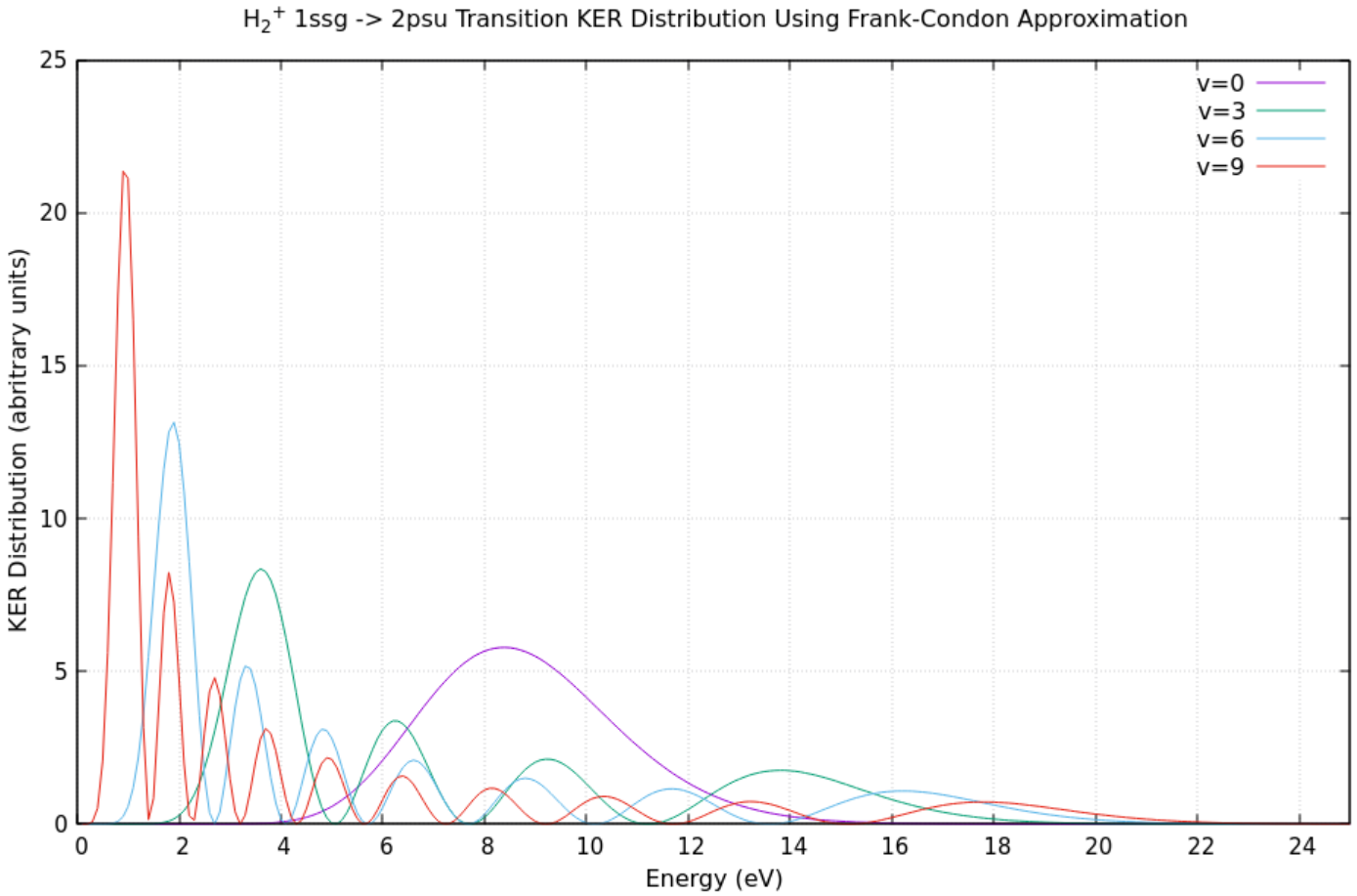

Figure 6: Kinetic Energy Release distribution using the Frank-Condon Approximation.

In the KER the distribution of kinetic energy is released based on different vibrational states is observed. The ground state vibrational mode has a gaussian appearance almost, with a relatively limited energy range. Curiously it seems that as the vibrational state increases, the more likely it is for the energy release to be smaller, but at the same time the probability of releases at high energy is present where it is not for the ground state.

The KER was calculated with a modified version of previous code. Please look at the github in *dissociative.f90* towards the end for the main edits. They are a little too lengthy for me to place inside a code listing. However, th subroutine *NumerovAppxMu.f90* is not and the forwards version which is implemented in that code is given in section 2.5.

2 Code Listings

2.1 Forwards Numerov

```
subroutine NumerovForwards(psi_L, V, nr, E, n, s, dr)

implicit none

! initialise local and inbound variables
integer*8 :: i, j
real *8, intent(in) :: s, E, dr, n
integer*8, intent(in) :: nr
! array init

real* 8, dimension(nr), intent(in) :: V
real* 8, dimension(nr):: psi_L
real*8, dimension(nr) :: g
real*8 :: psi_ip1, psi_ip2, denom

g = 2*(V-E)

! for the recurrence relation we need to initialise two values of psi
psi_L(1) = 0.0d0
psi_L(2) = (-1)**n * s
Print *,  psi_L(2)

! this is essentially a recurrence relation like the Laguerre situation
do i=3,nr
denom = 1-(dr**2/12)*g(i)
psi_ip1 = (1 + (5*dr**2/12)*g(i-1) )*psi_L(i-1)
psi_ip2 = (1 - (dr**2/12)*g(i-2) )*psi_L(i-2)
psi_L(i) = (1/denom) * ( 2*psi_ip1 - psi_ip2)

end do

end subroutine NumerovForwards
```

2.2 Backwards Numerov

```
subroutine NumerovBackwards(psi_R, V, nr, E, n, s, dr)

implicit none

! initialise local and inbound variables
integer*8 :: i, j
real *8, intent(in) :: s, E, dr, n
integer*8, intent(in) :: nr

real*8 :: psi_ip1, psi_ip2, denom

! array init

real* 8, dimension(nr), intent(in) :: V
```

```
real* 8, dimension(nr) :: psi_R
real*8, dimension(nr) :: g

g = 2*(V-E)


! for the recurrence relation we need to initialise two values of psi
psi_R(nr) = 0.0d0
psi_R(nr-1) = s

!numerov but hes backwards
do i= nr-2,1,-1
denom = (1-((dr**2.0)/12.0)*g(i))
psi_ip1 = (1.0 + (5.0*(dr**2.0)/12.0)*g(i+1) )*psi_R(i+1)
psi_ip2 = (1.0 - ((dr**2.0)/12.0)*g(i+2) )*psi_R(i+2)
psi_R(i) = (1.0/denom) * (2.0*psi_ip1-psi_ip2)

end do
end subroutine NumerovBackwards
```

2.3 Node Counting and E Guess

```
E_min = n
E_max = n + 0.75

pass_condition = .false.
do while(pass_condition .eqv. .false.)
        nodes = 0
        E = (E_min + E_max)/2

        call NumerovForwards(psi_L, V, nr, E, n, 0.00001, dr)
        call NumerovBackwards(psi_R, V, nr, E, n, 0.00001, dr)

        psi_L = psi_L / psi_L(x_m)
        psi_R = psi_R / psi_R(x_m)

        do i=1,nr
                if(i .le. x_m) then
                        psi(i) = psi_L(i)
                else
                        psi(i) = psi_R(i)
                end if
    end do

    do i=1,nr-1
            if(psi(i) < 0 .AND. 0 < psi(i+1)) then
                    nodes = nodes + 1
            elseif(psi(i) > 0 .AND. 0 > psi(i+1)) then
                    nodes = nodes + 1
            end if
    end do
    Print *, "Number_of_Nodes"
    Print *, nodes

    if(nodes==n) then
```

```
                pass_condition = .true.
        else if (nodes < n) then
                E_min = E
        else if (nodes > n) then
                E_max = E
        end if
        Print *, "Energy"
        Print *, E

    end do
```

2.4 Cooley's Energy Correction

```
e_lim = 1E-10
pass_condition = .false.
Ecorr = E
do while (pass_condition .eqv. .false.)

 g = 2*(V-Ecorr)
 call NumerovForwards(psi_L, V, nr, Ecorr, n, 0.00001, dr)
 call NumerovBackwards(psi_R, V, nr, Ecorr, n, 0.00001, dr)

 psi_L = psi_L / psi_L(x_m)
 psi_R = psi_R / psi_R(x_m)

 do i=1,nr
  if(i .le. x_m) then
                psi(i) = psi_L(i)
  else
                psi(i) = psi_R(i)
  end if
 end do


 Yx = (1-(dr**2/12)*g(x_m))*psi(x_m)
 Yx1 = (1-(dr**2/12)*g(x_m+1))*psi(x_m+1)
 Yxm1 = (1-(dr**2/12)*g(x_m-1))*psi(x_m-1)
 fract = (psi(x_m)/sum(psi**2))

 cooley_correct = fract* ( -(0.5/dr**2)*(Yx1 - 2.0*Yx + Yxm1) + (V(x_m)-Ecorr)*psi(x_m))

 if(abs(cooley_correct) > e_lim) then
        Ecorr = Ecorr + cooley_correct
 else if (abs(cooley_correct) <= e_lim) then
        pass_condition = .true.
 end if


 Print *, "cooley_correct:"
 Print *, cooley_correct
 Print *, "Corrected_E"
 Print *, Ecorr

end do
```

2.5 Numerov Subroutine for dissociative.f90

```
subroutine NumerovForwardsMu(psi_L, nr, n, s, dr, mu, g)

        implicit none

        ! initialise local and inbound variables
        integer*8 :: i, j
        real *8, intent(in) :: s, dr, n, mu
        integer*8, intent(in) :: nr
        ! array init

        !    real* 8, dimension(nr), intent(in) :: V
        real* 8, dimension(nr):: psi_L
        real*8, dimension(nr), intent(in) :: g
        real*8 :: psi_ip1, psi_ip2, denom

        ! for the recurrence relation we need to initialise two values of psi
        psi_L(1) = 0.0d0
        psi_L(2) = (-1)**n * s
        Print *,  psi_L(2)
        Print *, size(g), g(1), nr



        ! this is essentially a recurrence relation like the Laguerre situation
        do i=3,nr

                denom = 1-(dr**2/12)*g(i)
                psi_ip1 = (1 + (5*dr**2/12)*g(i-1) )*psi_L(i-1)
                psi_ip2 = (1 - (dr**2/12)*g(i-2) )*psi_L(i-2)
                psi_L(i) = (1/denom) * ( 2*psi_ip1 - psi_ip2)

        end do

        Print *,  psi_L(2)

end subroutine NumerovForwardsMu
```