Assignment 3

Don Wijesinghe

June 4, 2024

1 Problem 1: Quantum Harmonic Oscillator

We calculate the wave functions $\psi_n(x)$ and corresponding energies E_n of the quantum harmonic oscillator for $n = 0, 1, 2, 3...$ using the Numerov-Cooley method.

The potential for the quantum harmonic oscillator is given by:

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

For simplicity, we take $\omega = 1$ in Hartree atomic units.

The Schrödinger equation for a quantum harmonic oscillator is:

$$-\frac{1}{2}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x)$$

To solve this equation numerically, we use the Numerov method, which is a finite difference method that approximates the second derivative of the wave function. The Numerov method is more accurate than simpler finite difference methods because it includes higher-order correction terms.

1.1 Discretisation Using the Numerov Method

Numerov's method is a more accurate finite difference scheme to solve differential equations of the form

$$\frac{d^2 f(x)}{dx^2} = g(x)f(x).$$

where the Numerov iteration scheme is given by:

$$f_{i+1} = \left(2\left[1 + \frac{5\delta x^2}{12}g_i\right]f_i - \left[1 - \frac{\delta x^2}{12}g_{i-1}\right]f_{i-1}\right) / \left(1 - \frac{\delta x^2}{12}g_{i+1}\right)$$

In the context of the Schrödinger equation, we have:

$$\frac{d^2\psi(x)}{dx^2} = 2(V(x) - E)\psi(x).$$

Here, $g(x) = 2(V(x) - E)$. Substituting this into Numerov's iteration scheme, we get:

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

1.2 Forward and Backward Iteration Schemes

The backward iteration scheme can be derived by rearranging the forward scheme. Starting from the forward Numerov scheme:

$$\psi_{i+1} = \left(2\left[1 + \frac{5\delta x^2}{12} g_i\right]\psi_i - \left[1 - \frac{\delta x^2}{12} g_{i-1}\right]\psi_{i-1}\right) / \left(1 - \frac{\delta x^2}{12} g_{i+1}\right) \tag{2}$$

Rearranging for ψ_{i-1}:

$$\psi_{i-1} = \left(2\left[1 + \frac{5\delta x^2}{12} g_i\right]\psi_i - \left(1 - \frac{\delta x^2}{12} g_{i+1}\right)\psi_{i+1}\right) / \left(1 - \frac{\delta x^2}{12} g_{i-1}\right) \tag{3}$$

This backward scheme is used to propagate the wave function from the right boundary.

1.3 Normalisation

$$\langle\psi|\psi\rangle = \int_{-\infty}^{\infty} |\psi(x)|^2 dx = 1$$

The wave function must be normalised in the discrete case using Simpson's rule for numerical integration:

$$\langle\psi|\psi\rangle = \sum_{i=1}^{N} |\psi_i|^2 w_i = 1 \tag{4}$$

where w_i are the weights for Simpson's rule. Then we square root this and divide by it to get the normalised wavefunction.

$$\psi_{norm} = \psi / \sqrt{\langle\psi|\psi\rangle} \tag{5}$$

1.4 Implementation

1.5 Initialisation

Define grid parameters, potential, and arrays. Allocate arrays for x, V, weights, and wave functions (ψ_L, ψ_R, ψ). N is taken as the size of the x grid from $x = -5.0$ to $x = 5.0$ in this case. We also ensure N is an odd number, since this is a requirement for simpson's rule later needed for integration.

```
1  program main
2      implicit none
3      real*8, parameter :: xmin = -5.0d0, xmax = 5.0d0, s=1.0e-4, eps
           =1.0e-6
4      real*8, allocatable :: x(:), V(:), weights(:), psiR(:), psiL(:),
           psi(:)
5      real*8 :: dx, E, Emax, Emin, deltaE, integral
6      integer :: i, nodes, N, iter, max_iter, node_count
7      logical :: converged
8
9      N = 21
10     max_iter = 10000
11
12     if (mod(N, 2) == 0) N = N + 1
13     dx = (xmax - xmin) / (N - 1)
```

```
    allocate(x(N), V(N), weights(N), psiL(N), psiR(N), psi(N))

    ! Calculate weights for Simpson's rule
    weights(1) = 1.0d0
    do i = 2, N-1
        weights(i) = 2.0d0 + 2.0d0 * mod(i + 1, 2)
    end do
    weights(N) = 1.0d0
    weights(:) = weights(:) * dx / 3.0d0

    ! Initialize grid and potential
    do i = 1, N
        x(i) = xmin + (i - 1) * dx
        V(i) = 0.5 * x(i)**2
    end do
```

1.6 Numerov Forward and Backward Subroutines

Using (2) and (3) we implement forward and backward Numerov methods to calculate the wave functions ψ_L, ψ_R from the boundaries $x = -5.0$ and $x = 5.0$. To ensure correct boundary conditions, both forwards and backwards we set ψ_1=ψ_N to 0. The second point for ψ_L is set to $s(-1)^n$, for some small number s. For ψ_R we set ψ_{N-1} to s. The subroutines then calculate $g(x)$ given $V(x)$ and E as inputs to calculate ψ_L or ψ_R.

```
subroutine numerov_forward(psi, V, E, N, s, nodes, dx)
    integer, intent(in) :: N, nodes
    real*8, intent(in) :: V(N), E, s, dx
    real*8, intent(out) :: psi(N)
    real*8, allocatable :: g(:)
    integer :: i

    allocate(g(N))
    psi(1) = 0.0
    psi(2) = s * (-1.0)**nodes

    do i = 1, N
        g(i) = 2.0 * (V(i) - E)
    end do

    do i = 3, N
        psi(i) = ((2.0 + 5.0 * dx**2 / 6.0 * g(i - 1)) * psi(i - 1)
           - &
                  (1.0 - dx**2 / 12.0 * g(i - 2)) * psi(i - 2)) / &
                 (1.0 - dx**2 / 12.0 * g(i))
    end do
    deallocate(g)
end subroutine numerov_forward



subroutine numerov_backward(psi, V, E, N, s, dx)
```

```
    integer, intent(in) :: N
    real*8, intent(in) :: V(N), E, s, dx
    real*8, intent(out) :: psi(N)
    real*8, allocatable :: g(:)
    integer :: i

    allocate(g(N))
    psi(N) = 0.0
    psi(N - 1) = s

    do i = 1, N
        g(i) = 2.0 * (V(i) - E)
    end do

    do i = N - 2, 1, -1
        psi(i) = ((2.0 + 5.0 * dx**2 / 6.0 * g(i + 1)) * psi(i + 1)
           - &
                  (1.0 - dx**2 / 12.0 * g(i + 2)) * psi(i + 2)) / &
                 (1.0 - dx**2 / 12.0 * g(i))
    end do
    deallocate(g)
end subroutine numerov_backward
```

1.7 Main part of the program

```
    ! Loop over eigenstates
    do nodes = 0, 3
        ! Estimate energies
        Emin = 0.0d0
        Emax = 10.0d0
        converged = .false.
        iter = 0

        do while (.not. converged .and. iter < max_iter)
            ! Calculate the new energy
            E = (Emin + Emax) / 2.0d0
            print*, 'E', E

            ! Shoot from left boundary to right boundary to generate
                wf
            call numerov_forward(psiL, V, E, N, s, nodes, dx)
            call numerov_backward(psiR, V, E, N, s, dx)
            call match_wavefunctions(psiL, psiR, psi, N)

            ! Count nodes
            node_count = count_nodes(psi, N)
            print*, 'nodes = ', node_count

            ! Adjust energies to get correct nodes
            if (node_count < nodes) then
                Emin = E
```

```
            else if (node_count > nodes) then
                Emax = E
            else
                print*, '**calculating Cooleys energies'
                do while (.not. converged)
                    call numerov_forward(psiL, V, E, N, s, nodes, dx
                       )
                    call numerov_backward(psiR, V, E, N, s, dx)
                    call match_wavefunctions(psiL, psiR, psi, N)
                    call cooley_correction(psiL, psiR, V, E, psi,
                       deltaE, N, dx)

                    ! Check for convergence
                    if (abs(deltaE) > eps) then
                        E = E + deltaE
                        print*, 'Energy correction:', deltaE, 'New
                           E:', E
                    else if (abs(deltaE <= eps) then
                        converged = .true.
                        print*, 'Energy correction:', deltaE, 'New
                           E:', E
                        print*, '**converged**'
                    end if
                    iter = iter + 1
                    if (iter >= max_iter) exit
                end do
            end if
            iter = iter + 1
        end do

        ! Normalize wave function
        write(*,'(A, I0)') '#Iterations till convergence: ', iter
        integral = sum((psi(:)**2) * weights(:))
        psi(:) = psi(:) / sqrt(integral)

        ! Write wave function to file
        call write_wave_function(nodes, x, psi, E)
        print*, '-----------------------------------'
    end do

    ! Write potential to file
    open(unit=1, file='output.txt', status='replace', action='write'
       )
    do i = 1, N
        write(1, '(F10.4, 1X, F10.4)') x(i), V(i)
    end do
    close(1)

    deallocate(x, V, weights, psiL, psiR, psi)
end program main
```

The main program iterates through each eigenstate, guesses the eigenenergy between E_{min}

and E_{max} and calculates the wavefunction using the Numerov forward and backward methods using the subroutines in the previous section. It then joins the wavefunctions at a middle point using the matching subroutine shown below. It takes in ψ_L and ψ_R, Calculates the index for some middle point x_m to join the two wavefunctions at. If either wave function is zero at this point, the index is adjusted to a different point to avoid this. We then match the wavefunction into ψ.

```
subroutine match_wavefunctions(psiL, psiR, psi, N)
    integer, intent(in) :: N
    real*8, intent(in) :: psiL(N), psiR(N)
    real*8, intent(out) :: psi(N)
    integer :: i, m

    m = N / 2
    do while((psiL(m) == 0.0 .or. psiR(m) == 0.0) .and. m < N)
        m = m + 1
    end do

    do i = 1, m
        psi(i) = psiL(i) / psiL(m)
    end do
    do i = m + 1, N
        psi(i) = psiR(i) / psiR(m)
    end do
end subroutine match_wavefunctions
```

Once the matched wavefunction ψ is obtained, we begin counting the number of nodes in the wavefunction by using the following subroutine:

```
integer function count_nodes(psi, N)
    real*8, intent(in) :: psi(N)
    integer, intent(in) :: N
    integer :: i

    count_nodes = 0
    do i = 2, N
        if (psi(i - 1) * psi(i) < 0.0) then
            count_nodes = count_nodes + 1
        end if
    end do
end function count_nodes
```

If the number of nodes does not match n, the loop continues and narrows down the energy range further. Once the correct number of nodes is found, Cooley's method is used to get a more accurate value for the energy as this guessed value may not be exact, leading to deviations from the true wave function, especially near the boundaries.

1.8 Cooley's Correction

Cooley's energy correction is a technique used to refine the energy estimate of a wave function in the process of solving the Schrödinger equation numerically. The main idea behind Cooley's correction is to use the wave function obtained from the Numerov method and correct the energy estimate to achieve better accuracy and faster convergence.

Cooley's energy correction provides a way to iteratively improve the guessed energy E based on the current wave function ψ. The correction ΔE is calculated by considering the deviations of the wave function from the expected behavior and using them to adjust E and is given by:

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

for some matching point x_m, where,

$$Y_i = \left(1 - \frac{\delta x^2}{12} g_i\right) \psi_i.$$

Cooley's energy correction is implemented in the program using the following subroutine:

```
subroutine cooley_correction(psiL, psiR, V, E, psi, deltaE, N, dx)
    integer, intent(in) :: N
    real*8, intent(in) :: psiL(N), psiR(N), V(N), E, dx
    real*8, intent(out) :: deltaE
    real*8, intent(inout) :: psi(N)
    real*8 :: numerator, denominator, Ym, Ym_minus1, Ym_plus1
    integer :: i, m

    m = N / 2
    do while ((psiL(m) == 0.0 .or. psiR(m) == 0.0) .and. m < N)
        m = m + 1
    end do

    Ym = (1.0 - dx**2 / 12.0 * 2.0 * (V(m) - E)) * psi(m)
    Ym_minus1 = (1.0 - dx**2 / 12.0 * 2.0 * (V(m - 1) - E)) * psi(m
       - 1)
    Ym_plus1 = (1.0 - dx**2 / 12.0 * 2.0 * (V(m + 1) - E)) * psi(m +
        1)

    numerator = psi(m) * (-0.5 * (Ym_plus1 - 2.0 * Ym + Ym_minus1) /
        dx**2 + (V(m) - E) * psi(m))
    denominator = sum(psi**2)

    deltaE = numerator / denominator
end subroutine cooley_correction
```

Once the cooley's energy correction is called, we check whether the energy correction is smaller than some value $\epsilon << 1$ for convergence, and if not the energy correction is added to E, the wavefunction is re-calculated with the new energy until the above condition is met for convergence. Once convergence is achieved, both while loops break and we normalise the wavefunction using (5) and print the wavefunction shifted up by it's energies to file for plotting.

1.9 Results

The calculated wavefunctions $\psi_n(x)$ for $n = 0, 1, 2, 3$ are compared with analytical solutions, As the grid spacing δx decreases, the numerical solutions converge to the analytical ones. Also the energy convergence also deviates from the true value for very high δx as seen in the left image of Figure 1. Perfect convergence with the analytical solution is seen on the right as $deltax$ is decreased.

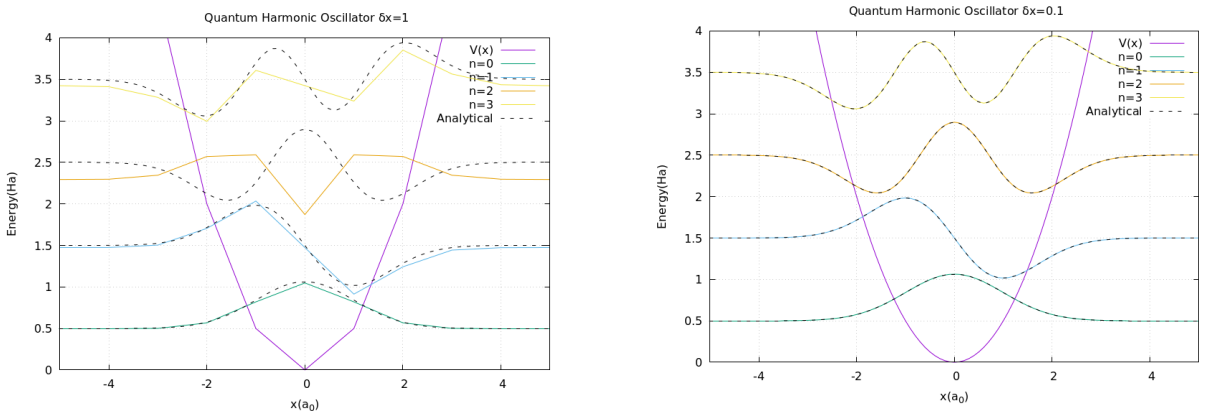

Figure 1: Solutions to the SHO $\psi_n(x)$ for $n = 0, 1, 2, 3$ comparing convergence to analytical solutions. Both analytical and numerical solutions are scaled by 0.75 to avoid overlap and are shifted up by their energies.

n	$\delta x = 1$	$\delta x = 0.5$	$\delta x = 1$
0	10	9	10
1	11	10	10
2	11	7	4
3	19	11	10

Table 1: Number of iterations taken for convergence for different values of δx for $\epsilon = 1.0e-6$

n	$\delta x = 1$	$\delta x = 0.5$	$\delta x = 0.1$	Analytical Energies
0	0.4950434	0.49974737	0.4999997	0.5
1	1.4738126	1.4982089	1.4999972	1.5
2	2.2943326	2.4935057	2.4999903	2.5
3	3.4230551	3.4833271	3.4999765	3.5

Table 2: Convereged energies vs analytical energies for different values of δx for $\epsilon = 1.0e-6$

The provided tables illustrate the energies for various n and δx values, along with the number of iterations required for convergence. As the δx value decreases, the calculated energy approaches the true eigenenergy value. Interestingly, the number of iterations required for convergence remains relatively constant despite further reductions in δx. Conversely, when δx is large, the number of iterations until convergence is significantly higher.

2 Problem 2: Dissociative wave functions for H_2^+

The Schrödinger equation for the dissociative wave functions is given by:

$$-\frac{1}{2\mu}\frac{d^2\nu(R)}{dR^2} + \epsilon(R)\nu(R) = E\nu(R)$$

where $\epsilon(R)$ is the potential energy curve (PEC) of the $2p\sigma_u$ state, μ is the reduced mass of the molecule, and E is the total energy.

For energies E above the dissociation limit of the PEC, the solutions represent dissociative states where the nuclei fly apart. The energy E is decomposed into:

$$E = E_k + D$$

where E_k is the kinetic energy release and D is the dissociation energy (asymptotic energy of the PEC at $R \to \infty$ equal to -0.5Ha).

2.1 Franck-Condon Approximation

The Franck-Condon approximation is used to estimate the KER distributions. The cross section for excitation to a dissociative state is proportional to the Franck-Condon factor:

$$|\langle \nu_f(E_k)|\nu_i\rangle|^2$$

where $\nu_f(E_k)$ is the final dissociative wave function and ν_i is the initial bound vibrational wave function.

2.2 Implementation

The potential energy curve (PEC) data for the $2p\sigma_u$ state is read from a file and interpolated to fit the grid points. The Dissociative wave functions are calculated for a range of kinetic energy releases using the Numerov method. The Numerov method is used to propagate the wave function from $R = 0$ to the desired R values. The continuum wave functions are normalised such that their amplitude tends to $\sqrt{\frac{2\mu}{k\pi}}$ as $R \to \infty$, where $k = \sqrt{2\mu E_k}$. Using this idea we first find the asymptotic amplitude of each wavefunction we calculated in the previous step, divide the wavefunction by it and multiply by this amplitude. The Franck-Condon factors are calculated by taking the overlap integral of the dissociative wave functions with the initial bound vibrational wave functions. The KER distributions are then computed and written to a file.

```
! read and interpolate data for PEC.2psu
file_name = 'PEC.2psu'
Vout=0.0
call read_and_interpolate(file_name, rn, r, Vout)
print*, 'interpolated data for PEC.2psu'
D = -0.5d0

!convert eV to Ha
Ekmax = Ekmax / 27.21136
Ekmin = Ekmin / 27.21136
num_Ek = 200
```

```
s=1.0e-5

allocate(psi2psu(rn, num_Ek), Ek(num_Ek))
psi2psu = 0.0
!calculate Ek grid

do i=1, num_Ek
    Ek(i) = Ekmin + (Ekmax - Ekmin) *(i-1) / (num_Ek - 1)
end do

!loop over Energies
do i=1, num_Ek
    E = Ek(i) + D
    call numerov_forward(psi2psu(:,i), Vout, E, rn, s , 1, dr, mu)
    !calculate asymptotic amplitude of the wf
    startidx = rn - rn / 10
    amp = psi2psu(startidx,i)
    do j=startidx, rn
        if (abs(psi2psu(j,i)) > amp) amp = abs(psi2psu(j,i))
    end do

    print*, 'amp', amp
    k = sqrt(2.0*mu*Ek(i))
    psi2psu(:,i) = psi2psu(:,i)*sqrt(2*mu / (k*pi)) / amp
end do

!write continuum wavefunctions to file
file_name = 'contwfs.txt'
open(unit=4, file=file_name, status='replace')
do i=1, rn
    write(4,*) r(i), 0.002*psi2psu(i,:) + D + Ek(:)
end do
close(4)

!franck condon approximation
allocate(KED(rn, 4))
vi=1
do j= 1,10,3
    do i=1, num_Ek
        integrand = psi2psu(:,i) * wf(:,j)
        KED(i,vi) = (abs(sum(integrand(:)*weights(:))))**2
    end do
    vi=vi+1
end do
file_name = 'KED.txt'
open(unit=4, file=file_name, status='replace')
do i=1, num_Ek
    write(4,*) 27.21136*Ek(i), KED(i,:)
end do
close(4)
```

2.3 Results

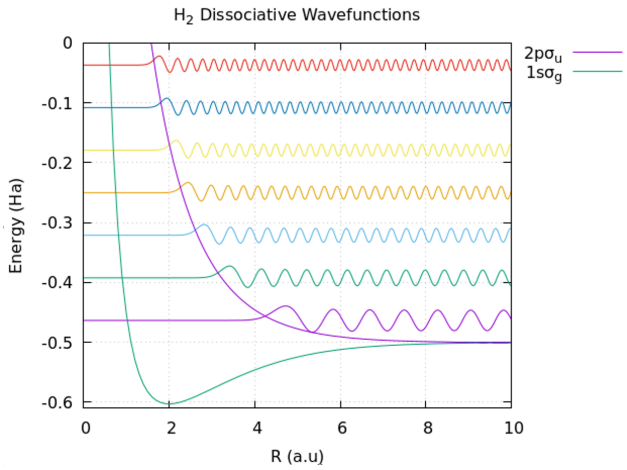

Figure 2: Dissociative wave fucntions in $2p\sigma_g$ scaled and shifted up by their energies.

The dissociative wave functions exhibit the expected oscillatory behavior, with higher kinetic energies resulting in more oscillations. This is consistent with the nature of dissociative states, where higher energy corresponds to greater momentum and thus more oscillations per unit distance.

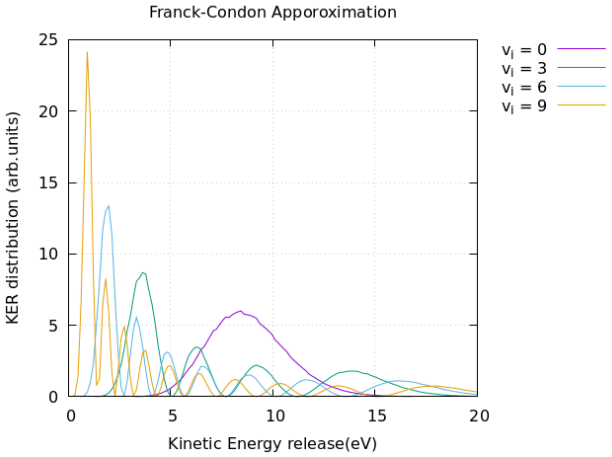

Figure 3: Kinetic energy release distributions for scattering on $v_i = 0, 3, 6, 9$ levels of the $1s\sigma_g$

The peaks in the KER distributions correspond to the energies at which the Franck-Condon overlap is maximised. We see that the number of peaks of the distribution increases with increasing initial vibrational states. These peaks indicate the most probable kinetic energies for the dissociation process. For low initial vibrational states, the peaks for these states are more spread out over higher kinetic energy values. This is because the initial vibrational energy is low, and the transition distributes energy more into kinetic forms. High inititial vibrational states have more vibrational energy which reduces the amount of energy that needs to be converted into kinetic energy for dissociation, making lower kinetic energy releases more probable as the peaks are higher and skewed towards lower kinetic energy values.