Computational Assignment 4: Potential Scattering

Ryan Craft

June 4, 2024

As per last few reports, the latest version of the code can be found on https://github.com/Ryan-Craft/ACQM2.git in the folder *comp4/template_1*. Code listings at the end. Made an attempt at problem 2, but in the interest of completing the bulk of assignment 5 problems 3/4 are probably not included, unless I was able to come back to this report and finish them at some point. Most of the code listings are the missing code components and any surrounding 'context', as I figure there is no need for you to see code you wrote. You've seen us make rgrids and stuff like that a lot so im not going to put that in here, but anything else that is new will be listed at the end of the document.

1 Problem 1: Potential Scattering

We consider the 1D scattering problem of a projectile electron/positron on a 1s state hydrogen atom. We assume that the atomic state is fixed in the 1s state, and ignore exchange terms of close-coupling, and consider different angular momentum of the projectile and the direct V matrix elements.

The direct V-matrix 1s elements become

$$\langle \mathbf{k}_f \phi_{1s} | \, \frac{z}{r_1} - \frac{z}{|\mathbf{r_1} - \mathbf{r_2}|} \, | \phi_{1s} \mathbf{k}_f \rangle \tag{1}$$

for projectile charge z.

Integrating over the target space we get the simple potential function operating on the right

$$\langle \mathbf{k}_f | \, z(1 + \frac{1}{r}) e^{-2r} \, | \mathbf{k}_f \rangle \tag{2}$$

$$V(r) = z(1 + \frac{1}{r}) e^{-2r} \tag{3}$$

The potential function is implemented as a function of r in code placed in listing 2.1.1.

Far from the potential, the particle acts as a plane wave with the ordinary Schrodinger equation

$$H_0 \, |\mathbf{k}\rangle = \frac{k^2}{2} \, |\mathbf{k}\rangle \tag{4}$$

The main goal here is to determine the cross section and the differential cross section of the scattering situation set up here. To determine this we will need to solve the 3D Lippman Schwinger equation for the transition matrix and the K matrix.

The 3d LS equation is given below

$$\langle \mathbf{k}_f | T | \mathbf{k}_i \rangle = \langle \mathbf{k}_f | V | \mathbf{k}_i \rangle + \int \frac{\langle \mathbf{k}_f | V | \mathbf{k} \rangle \, \langle \mathbf{k} | T | \mathbf{k}_i \rangle}{E + i0 - k^2/2} d\mathbf{k} \tag{5}$$

for $d\mathbf{k} = k^2 dk d\Omega$ integral over all space.

To simplify this 3D LS equation, apply partial wave expansions for the plane waves

$$\langle \mathbf{r}|\mathbf{k}\rangle = \sqrt{\frac{2}{\pi}}\frac{1}{kr}\sum_{lm} i^l u_l(r;k) Y_l^m(\hat{\mathbf{r}}) Y_l^{m*}(\hat{\mathbf{r}}) \tag{6}$$

Where u_l are free particle continuum waves which obey

$$\left[-\frac{1}{2}\frac{d^2}{dr^2} + \frac{l(l+1)}{2r^2} - \epsilon_k\right] u_l(r;k) = 0 \tag{7}$$

for $\epsilon_k = k^2/2$. The continuum waves are generated is per listing 2.1.2.

Which via problem 2's solution will simplify to the LS equation into

$$T_l(k_f, k_l) = V_l(k_f, k_l) + \int_0^\infty \frac{V_l(k_f,k) T_l(k,k_i)}{E + i0 - k^2/2} dk \tag{8}$$

Through theoretical lectures and assessments we know this can be simplified into a real valued integral defined in terms of a K matrix

$$K_l(k_f, k_i) = V_l(k_f, k_i) + P.V. \int_0^\infty \frac{V_l(k_f,k) K_l(k,k_i)}{E - k^2/2} dk \tag{9}$$

Which can be discretion over an r-domain for computing

$$K_l(k_f, k_i) = V_l(k_f, k_i) + \sum_n w_n V_l(k_f, k_n) K_l(k_n, k_i) \tag{10}$$

in this case the function $G_n = 1/(E - k_n^2/2)$ greens function is stored in the w_n integration weights. The entire system can be converted to a matrix

$$K_{fi} = V_{fi} + \sum_n w_n V_{fn} K_{ni} \tag{11}$$

The fi terms are on-shell terms, and the fn, ni elements are half-on shell. To get the transmission matrix and compute the differential and integrated cross sections we need to calculate the on shell T matrix, which requires that we solve for both off shell K matrix such that we can get the on-shell K matrix.

To do this we solve the following system of matrix equations

$$\sum_n \left[\delta_{fn} - w_n V_{fn}\right] K_{ni} = V_{fi} \tag{12}$$

$$K_{fi} = V_{fi} + \sum_n w_n V_{fn} K_{ni} \tag{13}$$

There is a need to generate the off-shell K matrix K_{ni} using the *dgesv* subroutine provided in the assignment, via *call dgesv(nkmax-1, 1, A, nkmax-1, ipiv, Koff, nkmax-1, info)*. After generating this the above equations can be solved, where the first one can be solved using the matrix solving subroutine of listing 2.1.4 (provided with the assignment), to get K_{ni}, which is used to get K_{fi} in the second. The V matrix is computed separately using listing 2.1.3.

The transmission matrix is then generated by calculating

$$T_{fi} = \frac{K_{fi}}{1 + \frac{i\pi}{k_f} K_{fi}} \tag{14}$$

We benefit from the elastic situation here because $k_f = k_i$.

Then, having the transmission matrix, the scattering amplitude can be expressed as

$$f(\mathbf{k_f}, \mathbf{k_i}) = -\frac{\pi}{k_i^2} \sum_l (2l+1) T_l(k_i, k_i) P_l(\cos\theta) \tag{15}$$

Which is calculated in listing 2.1.5 (which is also where the DCS is generated).

The differential cross section comes from this directly as ($k_f = k_i$).

$$\frac{d\sigma}{d\Omega} = |f(\mathbf{k_f}, \mathbf{k_i})|^2 \tag{16}$$

Then it is possible to calculate partial-wave integrated cross sections, which are defined as

$$\sigma_l = \frac{4\pi^3}{k_i^4} (2l+1) |T_l(k_i, k_i)|^2 \tag{17}$$

and subsequently the total cross section for a given maximum l is

$$\sigma = \sum_l \sigma_l \tag{18}$$

Both total and partial integrated cross sections are implemented in listing 2.1.6.

1.1 Results: Electron

The differential cross section of the $l_{max} = 3$ electron for energies between 5 and 35 eV, with steps of five is given in figure 1.

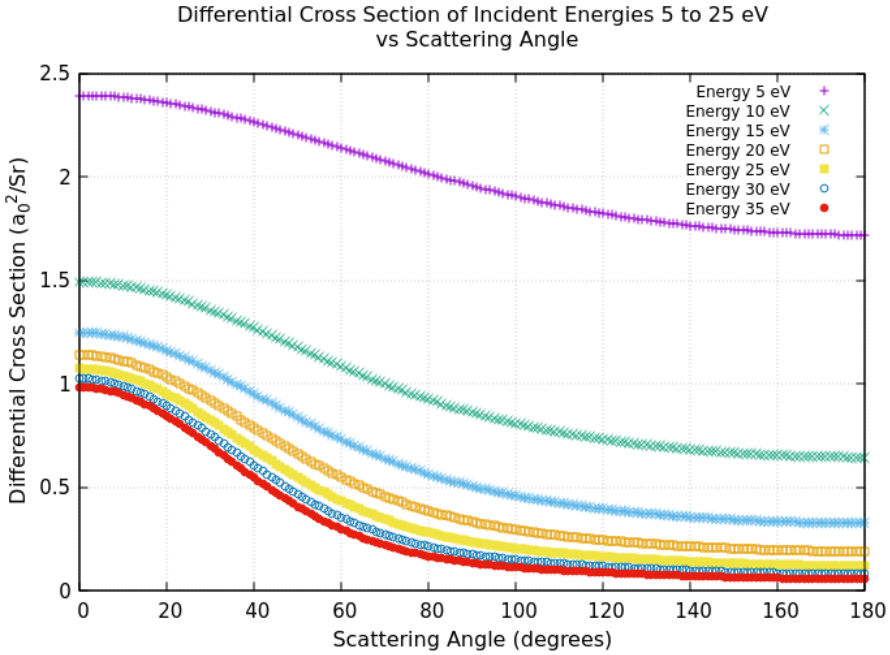

Figure 1: DCS of $l_{max} = 3$ electron scattering off of hydrogen 1s state with incident energies between 5 - 35 eV.

Over this range of energies it is clear that the DCS is approaching what is qualitatively similar to a gaussian distribution.

The variation of the DCS with constant energy and varied angular momentum is studied in figure 2.

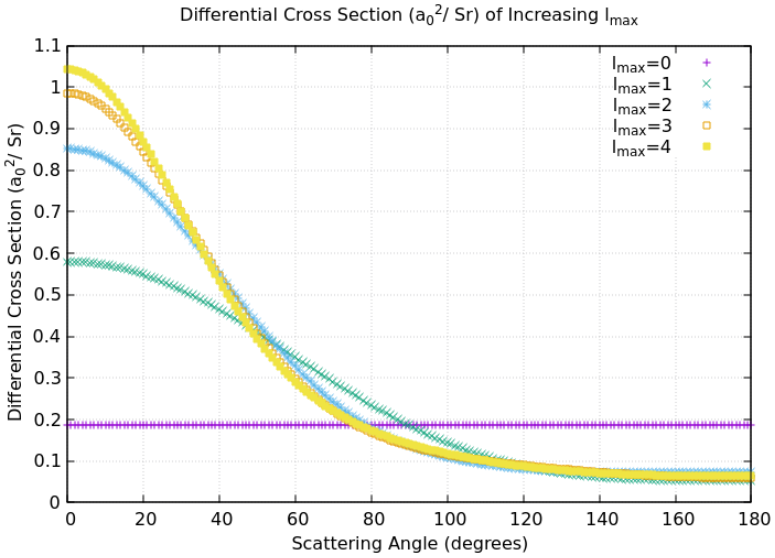

Figure 2: DCS of 35 eV incident electron on Hydrogen 1s state with lmax from 0 - 4.

Curiously the $l_{max} = 0$ DCS is a constant value over all scattering angles. This effect is very unusual, as a classical analogy of two solid objects colliding somewhat precludes the idea that the projectile could transmit *through* the target. With no angular momentum the projectile is equally as likely to scatter in any direction. In this case the 1s potential symmetric wrt the scattering angle, which has resulted in an equal likelihood that the incident electron will deflect at a given angle. Observing how this changes with different angular momentum of the target state would be very interesting.

Partial ICS were tested from the projectile energy ranges over 1-50 eV in steps of 0.5eV. The 0 eV point is not plotted because there is no result for an incident energy of zero, rather to probe energies near zero we require finer steps in the kgrid. 0.5 eV steps provided qualitatively smooth partial ICS curves. The partial-ICS curves for electron Hydrogen scattering are given in figure 3.

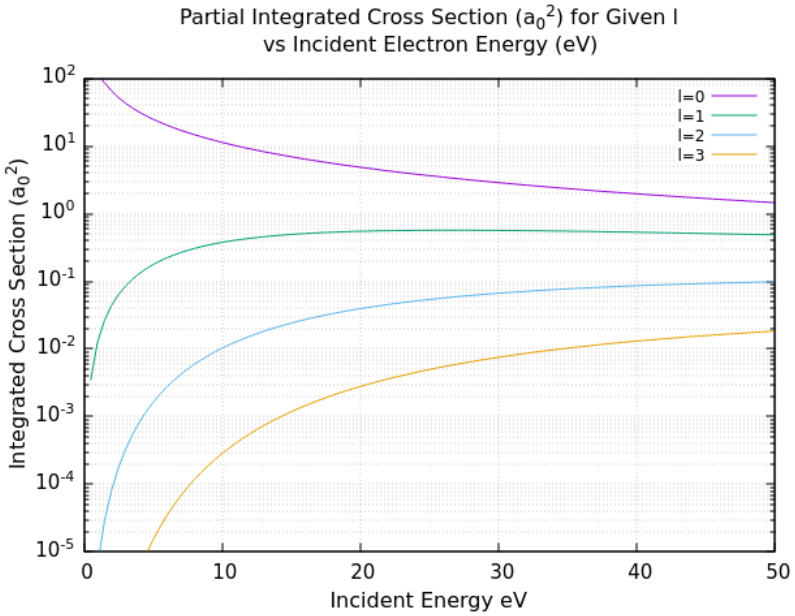

Figure 3: Partial ICS for different l values (0-3) for electron scattering on Hydrogen over an energy range 0.5-50 eV.

The total integrated cross sections for increasing $l_{max} = 0 - 3$ were also calculated, demonstrating the convergence to a single function of the cross section.

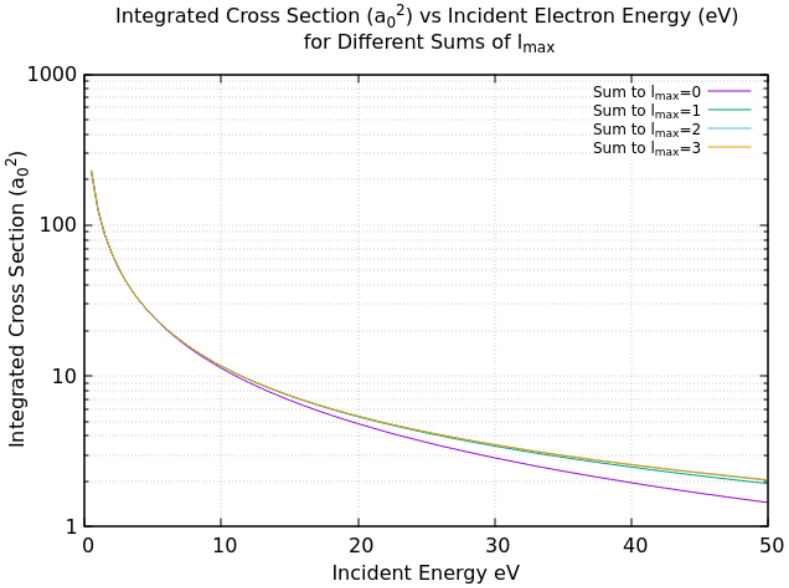

Figure 4: Partial ICS for different l values (0-3) for electron scattering on Hydrogen over an energy range 0-50 eV.

Figure 4 demonstrates the convergence of the integrated cross section for increasing values of l_{max}, which appear to approach a single function. Convergence is rapid, with the $l_{max} = 2$ and $l_{max} = 3$ being nearly identical.

1.2 Results: Positron

Positron scattering results were calculated for identical values to the electron case. In studying the convergence of the DCS at fixed $l_{max} = 3$ and positron incident energies of 5 - 35 eV, a similar convergence behaviour to a singular

scattering function in the high energy limit is identified in figure 5.

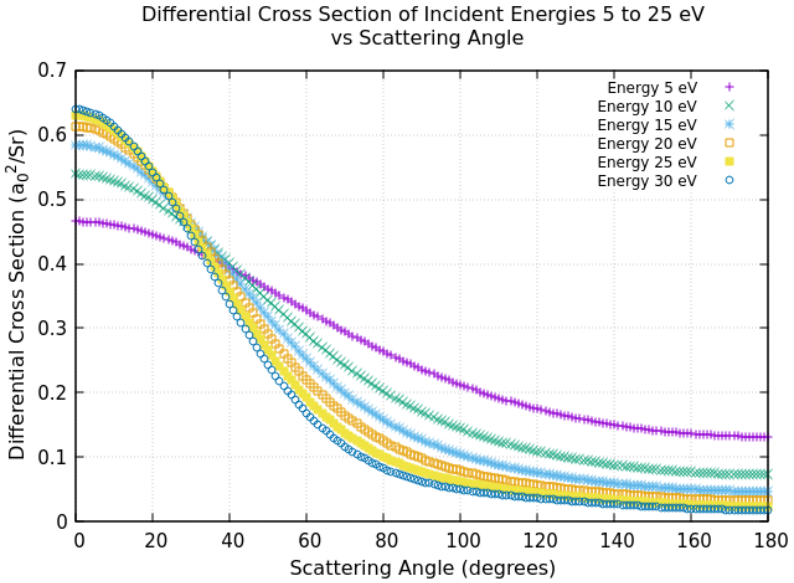

Figure 5: DCS of $l_{max} = 3$ positron scattering off of hydrogen 1s state with incident energies between 5 - 35 eV.

A kind of opposite behaviour is detected between the positron and electron cases when comparing figured 5 and 1. The 5eV projectile in positron scattering results in a lower differential cross section at the zero scattering angle, whereas for electron scattering, the same case has a substantially larger differential cross section. In fact the electron case has a higher differential cross section over the entire range of scattering angles for the every energy. The convergence is from higher scattering angle to lower, whereas for positron scattering the case is somewhat the opposite, with values at low theta being amplified for larger energies.

Curiously this behaviour does not flow into the case of specific energy, varied angular momentum of figure 6.

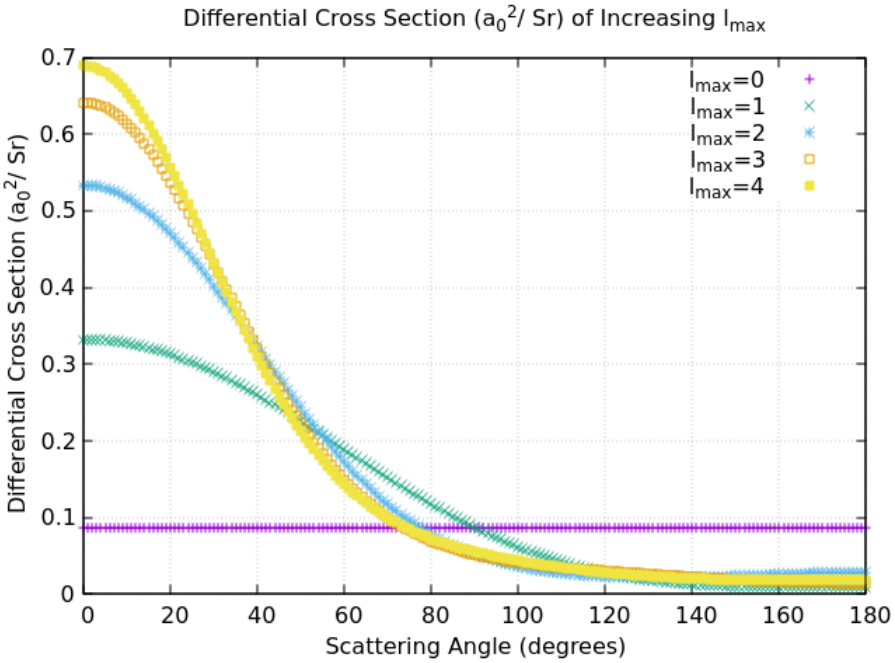

Figure 6: DCS of 35 eV incident positron on Hydrogen 1s state with lmax from 0 - 4.

Increasing the angular momentum seems to converge the values of the DCS towards a similar gaussian curve to that of the electron scattering case. The $l_{max} = 0$ case of uniform DCS is similar in both particles also.

The difference between the positron and electron behaviour seems to be mostly with respect to the energy. Whilst the positron does exhibit reduced values on all points for the DCS, the convergence behaviour is most different for static l and increasing energy.

This difference may arise from the fact that the positron is repelled by the stationary proton, and attracted by a somewhat delocalised electron. The delocalised electron may behave to partially screen the deflection of the positron and proton in a classical analogy. Conversely, with electron projectile, the electron screens the attractive force of the proton to the electron.

Thus we see has energy increases both tend towards a similar scattering behaviour as one another, as higher energies appear to have the effect of disregarding the differences in the internal structure of the hydrogen atom.

The same similarities and differences between l states and energies are witnessed in the partial and total integrated cross sections in figure 7 and 8.

In these plots the l=0 and/or the low energy cross sections are fundamentally different to the electron collision data, owing to a difference in the interaction between the target and the projectile which are most dominant at low energies and angular momenta.

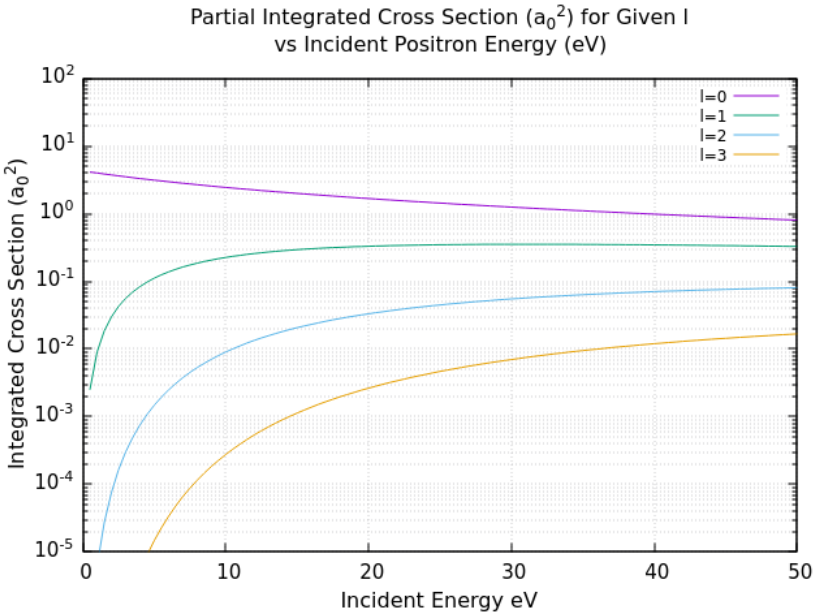

Figure 7: Partial ICS for different l values (0-3) for positron scattering on Hydrogen over an energy range 0.5-50 eV.

The l=0 partial cross section for the positron is substantially different to the electron, when comparing figures 3 and 7 this becomes apparent. The integrated cross section is lower for the positronic case at low energies and l. Convergence behaviour of the total cross section is similar to the electron but the function representing the cross section for the proton tends to be an order of magnitude less for all incident energies. Partial cross sections for l greater than zero tend to be near identical for both cases.

Ultimately, the higher order l values contribute less to the total cross section than the l=0 state. Thus despite convergence for higher l, the total integrated cross section between positron and electron hydrogen collision are substantially different.

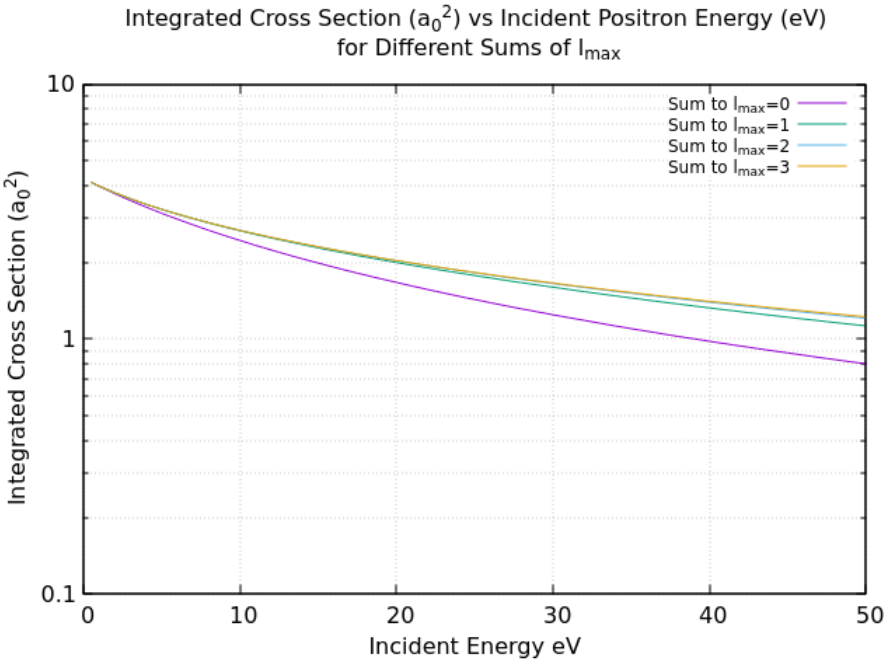

Figure 8: Partial ICS for different l values (0-3) for positron scattering on Hydrogen over an energy range 0-50 eV.

2 Problem 2

We start with the equation

$$\langle \mathbf{k_f}|T|\mathbf{k_i}\rangle = \langle \mathbf{k_f}|V|\mathbf{k_i}\rangle + \int_0^\infty k^2 dk \int_\Omega d\Omega \frac{\langle \mathbf{k_f}|V|\mathbf{k}\rangle \langle \mathbf{k}|T|\mathbf{k_i}\rangle}{E + i0 - k^2/2} \tag{19}$$

And we substitute the partial wave expansions of for the V and T matricies

$$\langle \mathbf{k_f}|T|\mathbf{k_i}\rangle = \frac{1}{k_i k_f} \sum_{lm} T_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \tag{20}$$

$$\langle \mathbf{k_f}|V|\mathbf{k_i}\rangle = \frac{1}{k_i k_f} \sum_{lm} V_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \tag{21}$$

The substitution

$$\frac{1}{k_i k_f} \sum_{lm} T_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) = \frac{1}{k_i k_f} \sum_{lm} V_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \tag{22}$$

$$+ \int_0^\infty k^2 dk \int_\Omega d\Omega \left[\frac{1}{k k_f} \sum_{lm} V_l(k_f, k) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \frac{1}{k_i k} \sum_{lm} T_l(k, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \right] / (E + i0 - k^2/2) \tag{23}$$

You can cancel the k^2 and multiply by $k_i k_f$ to remove some extra k terms.

$$\sum_{lm} T_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) = \sum_{lm} V_l(k_f, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \tag{24}$$

$$+ \int_0^\infty dk \int_\Omega d\Omega \left[\sum_{lm} V_l(k_f, k) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \sum_{lm} T_l(k, k_i) Y_{lm}(\hat{\mathbf{r}}) Y^*_{lm}(\hat{\mathbf{r}}) \right] / \left(E + i0 - k^2/2\right) \tag{25}$$

On the rhs lets integrate over the solid angle for a pair of the spherical harmonics to get some delta functions

$$\sum_{lm} T_l(k_f,k_i)Y_{lm}(\hat{\mathbf{r}})Y^*_{lm}(\hat{\mathbf{r}}) = \sum_{lm} V_l(k_f,k_i)Y_{lm}(\hat{\mathbf{r}})Y^*_{lm}(\hat{\mathbf{r}}) \tag{26}$$

$$+\int_0^\infty dk \left[\sum_{lm} V_l(k_f,k)Y_{lm}(\hat{\mathbf{r}})Y^*_{lm}(\hat{\mathbf{r}})T_l(k,k_i)\delta_{l_f l_i}\delta_{m_f m_i}\right] / \left(E+i0-k^2/2\right) \tag{27}$$

We can multiply again by $d\Omega$ and integrate over the solid angle to remove remaining spherical harmonics

$$\sum_{lm} T_l(k_f,k_i)\delta_{l_f l_i}\delta_{m_f m_i} = \sum_{lm} V_l(k_f,k_i)\delta_{l_f l_i}\delta_{m_f m_i} \tag{28}$$

$$+\int_0^\infty dk \left[\sum_{lm} V_l(k_f,k)T_l(k,k_i)\delta_{l_f l_i}\delta_{m_f m_i}\right] / \left(E+i0-k^2/2\right) \tag{29}$$

Which simplifies to (unsure about this part actually (or the other parts), this is just what came to mind)

$$T_l(k_f,k_i) = V_l(k_f,k_i) + \int_0^\infty dk \left[V_l(k_f,k)T_l(k,k_i)\right] / \left(E+i0-k^2/2\right) \tag{30}$$

It would have the effect of diagonalising the matricies, which Im unsure is true.

2.1 Listings

2.1.1 Generating V(r)

```
open(1, file="Scattering-Potential.txt", action="write")
do i=1,nrmax
        V(i) = zproj*(1.0d0+(1.0d0/rgrid(i)))*exp(-2.0d0*rgrid(i))
        write(1, *) rgrid(i), V(i)
end do
close(1)
```

2.1.2 Continuum Function Generation

In the l loop, the call to *setup_contwaves(...)* is made to generate continuum waves which are stored in the contwaves(:,:) matrix. For a reference on how the *calculate_Vmatrix* subroutine see the listing 2.1.3.

```
do l=lmin, lmax
        !populate contwaves matrix with a continuum wave for each off-shell k
        !RC: implemented but all cont waves normalised to unity
        contwaves = 0.0d0
        call setup_contwaves(nkmax,kgrid,l,nrmax,rgrid,contwaves)

        !evaluate the V-matrix elements
        !RC :: Implemented, checking now. Checked it, looks pretty good
        call calculate_Vmatrix(nkmax,kgrid,contwaves,nrmax,rgrid,rweights,V,Vmat)

        open(1, file="Vmat-halfonshell.txt", action="write")
        do i=1,nkmax
                write(1, *) kgrid(i), Vmat(i,1)
        end do
        close(1)
```

```
        !solve the Lippman-Schwinger equation for the on-shell T-matrix
        call tmatrix_solver(nkmax,kgrid,kweights,Vmat,Ton(l))
enddo
```

The function for generating the continuum waves and the Numerov Approximation used to generate an individual continuum wave are listen together below.

```
subroutine setup_contwaves(nkmax, kgrid, l, nrmax, rgrid, contwaves)
        implicit none
        integer, intent(in) :: nkmax, l, nrmax
        real*8, intent(in) :: kgrid(nkmax), rgrid(nrmax)
        real*8, intent(out) :: contwaves(nkmax,nrmax)
        real*8 :: ncontwaves(nkmax,nrmax)
        real*8 :: g(nrmax) ! RC: I added this one
        integer :: nk, nr, nodes !indices to loop over k and r
        real*8 :: E
        logical :: pass_condition
        !>>> iterate over k, populating the contwaves matrix
        !RC : We need to use forwards numerov to get the continuum waves

        do nk=1,nkmax
                E = kgrid(nk)**2/2
                g = 2*( l*(l+1)/(2*rgrid**2)  - E)
                call NumerovForwards(nrmax, rgrid, nkmax,
                                kgrid(nk), contwaves(nk,:), g, l)
        end do

        ! These need to made unit valued at their asymptotic region and then normalise

        Print *, "Writing Numerovl0vsSin.txt,
                contains rgrid, first continuum wave and sin(kr)"
        open(1, file="Numerovl0vsSin.txt", action="write")
        do nr=1,nrmax
                write(1, *) rgrid(nr), contwaves(3,nr), sin(kgrid(3)*rgrid(nr))
        end do
        close(1)

end subroutine setup_contwaves

subroutine NumerovForwards(nrmax, rgrid, nkmax, kval, psi, g, l)
        implicit none
        integer, intent(in) :: nrmax, l, nkmax
        real*8, intent(in) :: g(nrmax), rgrid(nrmax), kval
        real*8, intent(inout):: psi(nrmax)
        real*8 :: psi_ip1, psi_ip2, denom, dr
        integer*8 :: dfactorial, i, j
        ! RC : High precision integers becaues of the factorial

        dr = rgrid(1)

        !RC : Well we need to make a code to calculate a double factorial now
        dfactorial=1
        do i = (2*l+1), 0, -2
                if(i==0 .or. i==1 .or. i<0) then
                        dfactorial=dfactorial
                else
```

```
                dfactorial = dfactorial * i
            end if
        end do
        !Print *, "Factorial, l", dfactorial, i

        !RC : because of page 72 of the lectures
        psi(1) = (rgrid(1)*kval)**(l+1) / dfactorial
        psi(2) = (rgrid(2)*kval)**(l+1) / dfactorial
        !Print *, "NUMEROV LEFT BOUNDARY ::"
        !Print *, psi(1), psi(2)

        do i=3, nrmax
            denom = 1-(dr**2/12)*g(i)
            psi_ip1 = (1 + (5*dr**2/12)*g(i-1) )*psi(i-1)
            psi_ip2 = (1 - (dr**2/12)*g(i-2) )*psi(i-2)
            psi(i) = (1/denom) * ( 2*psi_ip1 - psi_ip2)
        end do
end subroutine NumerovForwards
```

2.1.3 Calculating the V-matrix

Other than the potential, a kgrid V matrix needs to be generated for later use working on creating the K and V matricies and scattering amplitudes. The generation of the 2D V matrix is as below. To see where this function is called, refer to the first listing in this section.

```
subroutine calculate_Vmatrix(nkmax,kgrid,contwaves,nrmax,rgrid,rweights,V,Vmat)
        use constants
        implicit none
        integer, intent(in) :: nkmax, nrmax
        real*8, intent(in) :: kgrid(nkmax), contwaves(nkmax,nrmax),
                    rgrid(nrmax), rweights(nrmax), V(nrmax)
        real*8, intent(out) :: Vmat(nkmax,nkmax)
        integer :: nkf,nki, i !indices for looping over on- and off-shell k

        !>>> evaluate the V-matrix elements and store in the Vmat matrix
        !    note: the V-matrix is symmetric, make use of this fact to reduce the
        !          amount of time spent in this subroutine
        Vmat=0.0d0
        do nkf =1, nkmax
            do nki =nkf,nkmax
                Vmat(nkf,nki) = (2/pi)*sum(contwaves(nkf,:)
                                *V(:)*contwaves(nki,:)*rweights(:))
                Vmat(nki,nkf) = Vmat(nkf,nki)
            end do
        end do

end subroutine calculate_Vmatrix
```

2.1.4 Matrix Equation Solver

This matrix solver is not my code, but for completeness so that it is understood how the matrix equations are solved, the subroutine is listed here:

```
subroutine tmatrix_solver(nkmax,kgrid,kweights,Vmat,Ton)
        use constants
```

```
        implicit none
        integer, intent(in) :: nkmax
        real*8, intent(in) :: kgrid(nkmax), kweights(nkmax), Vmat(nkmax,nkmax)
        complex*16, intent(out) :: Ton !on-shell T-matrix element
        complex*16 :: denom
        real*8 :: &
        Koff(nkmax-1), & !half-off-shell K-matrix elements
        Kon,           & !on-shell K-matrix element
        Von,           & !on-shell V-matrix element
        A(nkmax-1,nkmax-1) !Coefficient matrix for the linear system Ax=b
        integer :: f,n,j, ipiv(nkmax-1), info

        !>>> store the on-shell V-matrix element in Von
        Von = Vmat(1,1)

        !>>> populate the matrix A according to Eq (142) in the slides

        do f=1, nkmax-1
                do n=1, nkmax-1
                        if(f==n) then
                                A(f,n) = 1 - kweights(n+1)*Vmat(f+1,n+1)
                        else
                                A(f,n) = - kweights(n+1)*Vmat(f+1,n+1)
                        end if
                end do
        end do


        !>>> populate the vector Koff with the
        ! half-on-shell V-matrix elements (RHS of Eq (141))
        do n=1, nkmax-1
                Koff(n) = Vmat(n+1,1)
        end do

        !Here is the call to DGESV
        call dgesv(nkmax-1, 1, A, nkmax-1, ipiv, Koff, nkmax-1, info )
        if(info /= 0) then
                print*, 'ERROR in dgesv: info = ', info
        endif

        !>>> Now use the half-on-shell K matrix which has been
        ! stored in Koff to get the on-shell K-matrix element Kon

        Kon = Vmat(1,1) + sum(kweights(2:)*Vmat(1,2:)*Koff)

        !>>> And then use Kon to get the on-shell T-matrix element Ton

        denom = complex(1, (pi/kgrid(1))*Kon)

        Ton = Kon/denom

        Print *, "Complex declaration :::"
        Print *, "Kon", Kon
        Print *, Ton

end subroutine tmatrix_solver
```

The call to this function is again in the do loop of the first listing.

2.1.5 Generating the Scattering Amplitude and Computing the DCS

The generation of the scattering amplitudes is done when the DCS is being generated. In the loop where *f(ntheta)* is evaluated is where the scattering amplitude function is generated. Seeing as the DCS is the squared absolute magnitude of the scattering amplitude, its calculated immediately after generation of the f vector.

```
subroutine compute_dcs(nthetamax, theta, lmin, lmax, Ton, k, DCS)
        use constants
        implicit none
        integer, intent(in) :: nthetamax, lmin, lmax
        real*8, intent(in) :: theta(nthetamax), k
        complex*16, intent(in) :: Ton(0:lmax)
        real*8, intent(out) :: DCS(nthetamax)
        integer :: l, ntheta !loop indices
        real*8:: PL, PLval!Legendre polynomials - from file plql.f
        real*8 :: costheta !use this to store cos(theta in radians)
        complex*16 :: f(nthetamax) !scattering amplitude

        !>>> calculate the scattering amplitude f(theta) for each theta
        !    by iterating over l and using the partial-wave
        !    expansion of f
        ! RC :: I'll do this using the equation 115 of the lecture slides
        f = 0.0d0
        do ntheta=1,nthetamax
                do l=lmin,lmax
                        costheta = cos(theta(ntheta) * (pi/180))
                        f(ntheta) = f(ntheta) + (-pi/k**2) * (2*l+1)
                                * Ton(l) * PL(l, costheta)
                end do
        end do

        !>>> obtain the DCS from the scattering amplitude
        ! RC :: DCS is obtained by eqn 117, remember to do the
        ! complex conjugate, so exploit the abs functions complex arguments

        DCS = abs(f)**2

end subroutine compute_dcs
```

2.1.6 Partial and Total ICS

We create the partial ICS values in the *compute_ics* function and then when writing to file, add the sum per l as a column during the loop for the file output.

```
subroutine compute_ics(lmin, lmax, Ton, k, ICS)
        use constants
        implicit none
        integer, intent(in) :: lmin, lmax
        complex*16, intent(in) :: Ton(lmin:lmax)
        real*8, intent(in) :: k
        real*8, intent(out) :: ICS(lmin:lmax)
        integer :: l

        !>>> populate the ICS array with the partial-wave ICS per l
```

```
        do l=lmin,lmax
                ICS(l) = (4*pi**3 / k**4) * (2*l+1) * abs(Ton(l))**2
        end do
end subroutine compute_ics
```

File output line here

```
open(1, file="ICSout.txt", action="write")
do l=lmin,lmax
        write(1, "(*(g0.6,:,','))") ICS(l), sum(ICS(lmin:l))
end do
close(1)
```