Rutherford's Scattering: A Fortran Simulation

Antonio Ortiz Castro

Department of Physics
Universidad Anahuac
Huixquilucan 52786, Mexico.
antorca59@gmail.com

keywords: Rutherford scattering, Fortran
PACS: 34.50.Cx

(Submitted 16 Nov 2015)

Abstract

A simulation of Rutherford's experiments of 1911 about the scattering of α particles is presented using Fortran. The number of collisions (number of particles scattered) were considered per unit solid angle and the curve obtained was fitted with a new function.

1 Introduction

A hundred years ago Lord Ernest Rutherford conducted a series of experiments performed by Geiger and his student Marsden that lead him to discover atomic nucleus[1]. Bombarding a thin Au foil with α-particles emitted by RaC (Bi^{214}), they measured the scattering angle, and Rutherford deduced an expression for a coulomb field that was later confirmed quantum mechanically by N. Bohr. Rutherford recognized that the energy and direction of the emitted particle was a random event and a consequence of a single encounter particle-nucleus and not a result of multiple scatterings. The formula he deduced for the dfferential scattering cross section $\sigma(\theta)$ is well known and can be found in any Classical Mechanics text book [2, 3]. The α-particle is subject to a coulombian central force field whose potential is

$$V(r) = \frac{K}{r}, \qquad K = \frac{1}{4\pi\epsilon_0} q_1 q_2 \tag{1}$$

The impact parameter s and the scattering angle θ are related by:

$$s = \kappa \cot(\frac{\theta}{2}) \qquad (2)$$

where $\kappa = K/(2T'_0)$, and T'_0 is the α-particle's energy in the center of mass system [2, 3]. Goldstein [3] defines the differential scattering cross section as

$$\sigma(\theta)d\Omega = \frac{n}{I} = \frac{s}{\sin\theta}\left|\frac{ds}{d\theta}\right|$$

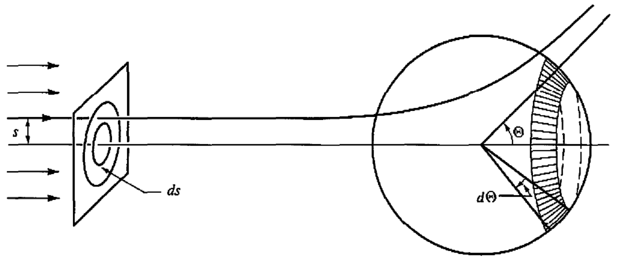
where n is the number of particles scattered within the solid angle $d\Omega$ per unit time and I is the intensity of the incident particles. Substituting s from eq (2) we obtain Rutherford's law:

$$\sigma(\theta) = \left(\frac{K}{4T'_0}\right)^2 \cdot \sin\left(\frac{\theta}{2}\right)^{-4}. \qquad (3)$$

2 Particle count

The number of particles scattered within unit solid angle $d\Omega$, that is, between θ and $\theta + d\theta$, is determined by[3]:

$$\sigma(\theta)Id\Omega = 2\pi\sigma(\theta)I\sin\theta|d\theta| \qquad (4)$$



The area of the shaded region is $2\pi s\cdot\Delta\theta$. The total area is obtained considering the whole range of θ values which is π rad or 180^o; that is, area $= 2\pi^2 s \cdot 180/\pi = 360\pi s$. ([3], p. 107)

The absolute value is taken because to an increase of θ corresponds a decrease in the scattering angle. The intensity of the incident beam I is calculated as the number of particles per unit area per unit time:

$$\begin{aligned} I &= \frac{\text{number of particles/sec}}{\text{area}} \\ &= = \frac{N}{2\pi R^2 \sin\theta\Delta\theta} = \frac{N}{90sr^2} \end{aligned}$$

Integrating eq (4)

$$\begin{aligned} n &= 4\pi I\sigma_0 \sin(\frac{\theta}{2})^{-2} \\ &= \frac{N\sigma_0}{90sr^2}\sin\left(\frac{\pi\theta}{360}\right)^{-2} \qquad (5) \end{aligned}$$

because θ is expressed in degrees. We found that this new function fits the data of the simulation.

3 Data

The next table shows the energies of the α-particles emitted by Bi^{214} and their relative intensities [4]:

Energy (keV)	Intensity
4941	5.3E-5
5023	4.4E-5
5184	1.28E-4
5273	0.00122
5452	0.0113
5516	0.0082

T_0' was taken as the expected value of these energies:

$$\text{Total Energy} = \text{Energy}_1 \cdot p_1 + \text{Energy}_2 \cdot p_2 + \cdots = 5.4628 \text{ MeV}$$

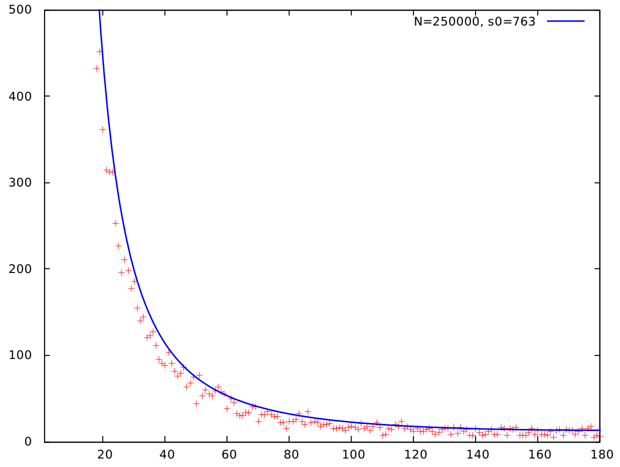
where p_i = relative intensity. Nuclear radius was calculated according to[5]:

$$r = (r_0 + \frac{r_1}{A^{2/3}} + \frac{r_2}{A^{4/3}}) \cdot A^{1/3} = 5.4499 \times 10^{-15} m$$

being A the atomic number and the parameters take the values: r_0 = 0.9071 fm, r_1 = 1.105 fm y r_2 = -0.548 fm, and 1 fm = 10^{-15} m. κ and s in ec(2) are expressed as multiples of the nuclear radius r ($\kappa = 3.8210\ r = 2.0824 \times 10^{-14}$ m).

4 Fortran

We vary two quantities: l_0 and N. The impact parameter s is generated at random between 0 and l_0, being l_0 an arbitrary value (l_0 = distance from the scattering nucleus expressed as a multiple of the nuclear radius). Rutherford reports that Geiger made some N = 250 000 counts at each mesurement session, so we decided to generate this number of "events". Next, the scattering angle θ given in eq (2) is generated, and then, expressed in degrees (the factor 2 in eq (2) becomes: 2*180/Pi = 114.59.) Each scattering angle is rounded to the nearest integer because we are interested in the number of particles scattered degree by degree. Afterwards we count the particles scattered in each degree. Note that since $s < l_0$, the scattered angles satisfy: $\theta > 2 * \arctan[\kappa/l_0] * 180/\pi$. The data is dumped in the file "dataC.dat" and read by Gnuplot to obtain the plot shown in fig 4. The code of the Fortran program is in appendix A.



Graph of n vs θ made with Gnuplot of the data generated by Fortran. θ is expressed in degrees and in the vertical axis is n. The continuous line corresponds to eq (5) while the data generated with Fortran are represented with crosses. In this case, s_0 = 763 nuclear radii, and 250 000 events were simulated. The code of this graph is in appendix B.

5 Conclusions

According to the results, the new expression eq (5), correctly explains the number of particles found in each degree, since, as we can see, it fits well the data generated. The results also show that it is possible to use Fortran to reproduce Rutherford's experiments of the scattering of α-particles by a nucleus, by a simple-code program. We would like to add that the random number generator[6] can be considered a good generator because it permitted us to obtain data that resembles the data observed of a phenomenon (emission of particles) which, by its very nature, we know it is completely aleatory.

A Fortran Code

```
Program scattering

implicit none
integer, dimension(180)  ::  counter
                real(8)  ::  e, epsilon0, k
                real(8)  ::  r0, r1, r2, AA, r
                real(8)  ::  s0
                real(8)  ::  T0, kappa, sigma0
                real(8)  ::  rand
                real(8)  ::  s, theta, sigma
     real(8), parameter  ::  pi  = 3.141592653589793d0
                integer  ::  error, i, n, v, d, Z


!----------------------[   Parameters of the problem   ]----------------------80
e         =   1.6022d-19
epsilon0  =   8.8542d-12
Z         =   79.
k         =   .5 * Z * e / (pi * epsilon0)

AA   =   1.9697d2
r    =  (0.898 + 1.376/AA**(.6666666) - 2.262/AA**(1.333333) ) * 10 **(-15.) &
         * AA**(.3333333)
```

```
T0       =   5.4628d6
kappa    =   .5 * k / (T0 * r)
sigma0   =   (  .25 * k / T0  )**2.0

!-----------------------------[   data   ]-----------------------------------80
print*,  "# of events 'N':"
 read*,   n
print*,  "Choose a value 's0':"
 read*,   s0

open(unit=20,file="dataC.dat",status="replace",action="write")

write(unit=20,fmt="(i8)")     n              !
write(unit=20,fmt="(f6.0)")   s0             !   Write these values
write(unit=20,fmt="(es13.6)") sigma0         !   for Gnuplot
write(unit=20,fmt="(es13.6)") r              !

!------------------------[   Initializing the counter   ]---------------------80

counter = 0

!--------------------------------[   loop   ]---------------------------------80

   call init_random_seed()

do   i  =  1, n

   call random_number(rand)

       s  =  s0 * rand
   theta  =  2.0 * atan( kappa / s )

   d  =  nint(theta * 180.0 / pi)
   if ( d <= 1.0 ) then                        !
     counter(1)    =  counter(1)  +  1.0       !
   else                                        !--------[   counter   ]--------80
     counter(d)    =  counter(d)  +  1.0       !
   endif                                       !
```

```
enddo

!-------------------------[   save the counter   ]--------------------------80
do   i  =  1, 180

  write(unit=20,fmt="(i3,i8)",iostat=errorC) i, counter(i)

enddo

close(unit=20)

!----------------------------[   messages   ]-------------------------------80
print"(a,i4)", "error in 'counter' = ", error

                           !*****************************
                                      contains
                           !*****************************

!--------------------[   subroutine  init_random_seed   ]--------------------80
     subroutine init_random_seed()
!       implicit none
       integer, allocatable :: seed(:)
       integer :: i, n, un, istat, dt(8), pid, t(2), s
       integer(8) :: count, tms

       call random_seed(size = n)
       allocate(seed(n))
       ! First try if the OS provides a random number generator
       open(newunit=un, file="/dev/urandom", access="stream", &
            form="unformatted", action="read", status="old", iostat=istat)
       if (istat == 0) then
          read(un) seed
          close(un)
       else
          ! Fallback to XOR:ing the current time and pid. The PID is
          ! useful in case one launches multiple instances of the same
          ! program in parallel.
          call system_clock(count)
          if (count /= 0) then
             t = transfer(count, t)
          else
             call date_and_time(values=dt)
             tms = (dt(1) - 1970) * 365_8 * 24 * 60 * 60 * 1000 &
                  + dt(2) * 31_8 * 24 * 60 * 60 * 1000 &
                  + dt(3) * 24 * 60 * 60 * 60 * 1000 &
                  + dt(5) * 60 * 60 * 1000 &
                  + dt(6) * 60 * 1000 + dt(7) * 1000 &
                  + dt(8)
             t = transfer(tms, t)
          end if
          s = ieor(t(1), t(2))
          pid = getpid() + 1099279 ! Add a prime
          s = ieor(s, pid)
          if (n >= 3) then
             seed(1) = t(1) + 36269
             seed(2) = t(2) + 72551
             seed(3) = pid
             if (n > 3) then
                seed(4:) = s + 37 * (/ (i, i = 0, n - 4) /)
             end if
          else
             seed = s + 37 * (/ (i, i = 0, n - 1 ) /)
          end if
       end if
       call random_seed(put=seed)
     end subroutine init_random_seed
endprogram scattering
```

B Gnuplot Code

```
reset
N       =   system("awk NR==1 dataC.dat")
s0      =   system("awk NR==2 dataC.dat")
sigma0  =   system("awk NR==3 dataC.dat")
```

```
r       =   system("awk NR==4 dataC.dat")

  f(x)  =  N * sigma0 /(90 * s0 * r**2.)  *  sin(x*pi/360.0)**-2.

plot [1.:180.][0.:500.]  'dataC.dat' every ::6 notitle, \
                         f(x) title sprintf("N=%.0f, s0=%.0f", N+0, s0+0) \
                         lw 1.5 lc 3
```

References

[1] E. Rutherford, *Philosophical Magazine*, Series 6, No. **21**, 669 (1911)

[2] J.B. Marion, *Classical Dynamics of Particles and Systems*, (Academic Press, 1970), pp. 302-309.

[3] H. Goldstein, Ch. Poole, J. Safko, *Classical Mechanics*, (Addison-Wesley, 3rd Ed, 2000), pp. 106-110.

[4] National Nuclear Data Center (Nuclear structure & decay data), *Brookhaven National Laboratory*, 2014.

[5] I. Angeli, K.P. Marinova, *Table of experimental nuclear ground state charge radii: An update*, Atomic Data and Nuclear Data Tables (Elsevier) **99** (2013) 69-95

[6] random_seed, Intrinsic Procedures, *The Gnu Fortran Compiler.*