1997

# Electromagnetic reaction cross sections in relativistic heavy ion collisions

Jack Dostal
*University of Northern Iowa*

# Electromagnetic Reaction Cross Sections in Relativistic Heavy Ion Collisions

**Jack Dostal**

**Presidential Scholars**
**Research Report**

## ABSTRACT

The goal of this project was to produce a computer program to predict the cross section of relativistic heavy ion collisions between given projectile and target nuclei. An explanation of the usefulness of relativistic heavy ion collision analysis is presented. The use of the Weiszacker-Williams method of virtual quanta for enhancing reaction cross sections of this type is discussed. Quantum-mechanical corrections to the semiclassical Weiszacker-Williams method are introduced. C++ computer code has been developed to calculate both semiclassical (as a check) and quantum-mechanical cross sections for these types of reactions. Preliminary results are given for the heavy ion reaction ($^{12}C$, $^{197}Au$), comparing the semiclassically generated and quantum mechanically generated cross sections with published values.

## SOLAR NEUTRINO PHYSICS

The use of heavy ion collisions to understand more about neutrino physics got its impetus from the Solar Neutrino Problem. Our sun produces energy via nuclear reactions inside its core. The chain of reactions which burns hydrogen (and thereby produces helium) is referred to as the proton-proton chain.[1] This chain consists of a series of successive reactions which produce energy in the form of photons, electrons, positrons, and neutrinos. Photons, electrons, and positrons travel only a very short distance before they interact electromagnetically with other particles in the sun. Thus, they do not reach the sun's surface. However, neutrinos only interact with other particles through the weak interaction. Because the neutrinos do not interact electromagnetically, these neutrinos escape from the sun's surface and reach the Earth. This is what makes neutrinos interesting to study; they are the only particles which can provide direct information about the sun's core.

Several experimental sites have been developed to measure the neutrino flux from the sun. One of the first was located deep inside the Homestake Mine in South Dakota. The Homestake detector (which remains in operation today) consists of a large underground tank containing $C_2Cl_4$, a common cleaning fluid. When a neutrino passes through the fluid, it will occasionally interact with one of the Cl nuclei. Each reaction produces an electron and an argon atom, which can be counted by the detector.[2]

The neutrino flux detected by the Homestake detector turns out to be only approximately one-third of the prediction by the standard solar model. This surprising result is the Solar Neutrino Problem. Results from other detectors[3] show similar discrepancies in their neutrino cross-section measurements. There are three general ways in which these discrepancies can be explained. The first possibility is that the thermodynamic stellar model of the sun is flawed. However, this is an unlikely explanation, as these neutrino discrepancies are much too large to be accounted for in the stellar model. A second option is that the particle physics used to interpret these reactions is incomplete. One explanation is a theoretical process called neutrino oscillation. This process would allow one type of neutrino to transform into another type. (There are three different types of neutrinos: electron neutrinos, muon neutrinos, and tau neutrinos.) A detector designed to detect electron neutrinos would then produce lower than expected results, due to some of the predicted electron neutrinos transforming into muon neutrinos. Although preliminary evidence for these neutrino oscillations has been found at LAMPF in Los Alamos,[4] that will not be addressed in this paper. The third possibility, which this paper focuses on, deals with the physics of the nuclear reactions taking place. The cross sections which determine the reaction rates in the sun may be incorrect.

## NUCLEAR REACTIONS

For the Homestake detector, the reaction of interest is:

$$^{7}\mathrm{Be} + \mathrm{p} \rightarrow {}^{8}\mathrm{B} + \gamma$$

The subsequent β+ decay of the radioactive $^{8}$B produces high energy neutrinos which are detected experimentally at Homestake. In order to gain a better understanding of this reaction, it is useful to study it in the laboratory. However, attempting to exactly reproduce the ($^{7}$Be,p) reaction is not possible. The above reaction takes place in the sun at a very low energy, approximately 20 keV. At that low an energy, no measurable cross section can be produced in the laboratory.

We can still learn about this reaction cross section if we consider its quantum-mechanically related time-reversed reaction:

$$\gamma + {}^{8}\mathrm{B} \rightarrow {}^{7}\mathrm{Be} + \mathrm{p}$$

This reaction, called $^{8}$B breakup, is a good candidate for study. However, the reaction presents two problems. First, $^{8}$B is a radioactive element with a half-life of approximately one second. Thus, it is not possible to create a $^{8}$B target at which photons could be directed. Second, attempting to collide beams of radioactive $^{8}$B and photons is impractical and would not produce a measurable cross section. These problems must be solved by using another method to measure these cross sections. For electromagnetic interactions between photons and nuclei, greater cross sections can be produced using the method of heavy ion reactions.

## HEAVY ION REACTIONS AND FORMALISM

A heavy ion reaction is an interaction between a target nucleus and the electromagnetic field of a heavy ion nucleus which causes transitions in the target nucleus. When the reaction involves nuclei with sufficiently high kinetic energy, the reaction is considered to be a relativistic heavy ion reaction. The probability of the reaction occurring can be calculated in terms of a cross section for the reaction. The simplest way to describe these relativistic heavy ion collisions is given by the equivalent photon method, developed by Weiszacker and Williams (also referred to as the Weiszacker-Williams method of virtual quanta).[5]

It is assumed that the projectile travels in a straight line near the target nucleus with a set velocity and impact parameter. The impact parameter is the shortest distance between the projectile and the target nucleus during the reaction. When $v \cong c$, where c is the speed of light, the electromagnetic field generated by the projectile looks contracted in the direction of motion. Photons also generate electromagnetic fields which are perpendicular to their direction of travel. Thus, in these heavy ion reactions, the large electromagnetic field of the heavy ion is equivalent to a large photon pulse. This is the cause of the increased cross section. As an example, the 8B breakup reaction presented above can be enhanced by passing the $^{8}B$ beam through lead ($^{208}Pb$). the large electromagnetic field of the lead nuclei enhances the reaction rate and creates a measurable cross section.

The cross section between two given nuclei can be calculated by multiplying the photon cross section $\sigma_\gamma(\omega)$by the equivalent photon spectrum $n(\omega)$ for each photon frequency $\omega$.[6] In integral form,

$$\sigma = \int n(\omega)\sigma_\gamma(\omega)d\omega$$

In relativistic heavy ion reactions, there are both electric dipole and electric quadropole contributions to the cross section. The photon cross section and the spectrum can both be divided into separate dipole and quadropole terms.[7] Thus, including these effects, the cross section is calculated as:

$$\sigma = \int [\, n_{E1}(\omega)\sigma_{E1\gamma}(\omega) + n_{E2}(\omega)\sigma_{E2\gamma}(\omega) \,] d\omega$$

where the subscript $_{E1}$ represents the dipole contribution. Likewise, the $_{E2}$ subscript represents the quadropole contribution. The semiclassical forms of the equivalent spectra are used from Bertulani and Baur[6] to calculate this semiclassical heavy ion cross section. They are:

$$n_{E1}(\omega) = \frac{2}{\pi} Z_1^2 \alpha \left(\frac{c}{v}\right)^2 \left[ \xi K_0 K_1 - \frac{v^2 \xi^2}{2c^2} \left(K_1^2 - K_0^2\right) \right]$$

$$n_{E2}(\omega) = \frac{2}{\pi} Z_1^2 \alpha \left(\frac{c}{v}\right)^4 \left[ 2\left(1 - \frac{v^2}{c^2}\right) K_1^2 + \xi \left(2 - \frac{v^2}{c^2}\right)^2 K_0 K_1 - \frac{v^4 \xi^2}{2c^4} \left(K_1^2 - K_0^2\right) \right]$$

where Z is the charge of the projectile, $\alpha$ is the fine-structure constant, K0 and K1 are Bessel functions, and $\xi$ is a function of the frequency $\omega$, velocity v, and the minimum impact parameter b_min.

This semiclassical calculation is used as a check in the computer code. For experimental data that has been analyzed semiclassically, the cross section found should match the semiclassical cross section calculated by the computer code.

There are two drawbacks in using the Weiszacker-Williams method directly. First, this method approximates the electromagnetic field of the heavy ion to be the same as the electromagnetic field of a point charge. However, in these reactions, the target and the projectile nuclei are separated by approximately one nuclear radius. The electromagnetic field is not equivalent to that of a point charge in that case. The second problem is that the two nuclei

interact for only a very short period of time. Because of this, the Heisenberg uncertainty principle has a significant effect on the results. Normal photons have a well-defined energy (E = pc). However, the Heisenberg uncertainty principle states that:

$$\Delta E \Delta t \geq h$$

where h is Planck's constant. Thus, a short lived photon, such as the ones exchanged between the two nuclei in this case, will have a very small uncertainty in lifetime $\Delta t$. Therefore, there is a significant uncertainty in the photon's energy. Because the energy of the photon is uncertain, it is classified as a virtual photon.

In addition, a third, quite serious problem is that the Weiszacker-Williams method assumes highly relativistic speeds for the projectile. However, this formalism is often applied to experiments that are only mildly relativistic ($\gamma \cong 2–3$). A full quantum treatment is necessary in such cases; some of the Weiszacker-Williams assumptions are not valid. By taking these features into account, the cross section of the heavy ion reactions can be reduced by as much as 25%. In the written computer code, the quantum effects are accounted for by using the quantum equivalent spectra from Benesh et al.[8,9] and can be found in the computer code in Appendix A.

## PROGRAM CODE

The following is a breakdown of the computer code written to calculate heavy-ion cross sections. This program has the capability to compute the cross section of a heavy ion reaction involving any two nuclei for which data is available. The target and projectile parameters required as inputs are: mass number, atomic number, nuclear charge radius, nuclear cross-sectional area, giant dipole resonance (GDR) energy, giant quadropole resonance (GQR) energy, nuclear GDR and GQR peak widths, and the nuclear radius. These parameters are read into the

program as a single structure, so that the program needs to make only one inquiry to the specified nuclear data file to extract all necessary data.

The program then reads the two requested nucleus files (one projectile, one target). Both the semiclassical and quantum-mechanical cross sections of the heavy ion reaction between the two nuclei is calculated. The quantum-mechanical cross section is the desired end result; the semiclassical cross section serves as a check.

The main body of the program consists of five major functions, which in turn call several other subroutines to be executed. Sigma_WW takes the target and projectile parameters and calculates the Weiszacker-Williams cross section. To do this, the program executes numerical integration over a specified range. Sigma_E2 takes these same parameters and calculates the E2, or quadropole, contribution to the cross section. NE1sigma_E2 calculates the cross term of the integration. This does not correspond to any experimental measured values, but is used to calculate the semiclassical heavy ion cross section SigmaHI. The dipole contribution to the cross section (SigmaE1) is also calculated. The calculation of the quantum-mechanical cross section follows a similar pattern. QMSigmaE1 and QMSigmaE2 calculate the dipole and quadropole cross sections respectively, and are combined to produce the quantum-mechanical heavy ion cross section QMSigmaHI.

Various subroutines not listed perform the bulk of the computational work in the program. These subroutines, as well as the main program, can be found the code in Appendix A.

## RESULTS

Using data for a $^{12}C$ projectile nucleus and a $^{197}Au$ target nucleus, the calculated cross sections are listed below. They are also compared to corresponding semiclassical and quantum predictions in papers by Norbury and Benesh et al.[9,10]

| **Cross section calculation for ($^{12}C$, $^{197}Au$)** | **Predicted cross section (millibarns)** |
|---|---|
| **Semiclassical cross section (Norbury)** | 43-45 |
| **Semiclassical cross section (computed result)** | 44 |
| **Quantum cross section (Benesh et al.)** | 40 |
| **Quantum cross section (computed result)** | 39 |

The published predictions and the computed result from the written code correspond quite well to one another. Thus, the program matches published values as expected. It can now be used to compute cross sections for other nuclear reactions in a reliable manner. The code will be especially useful for reactions that have only been analyzed semiclassically. The semiclassical cross sections (from a previous analysis and from the code) should match, which will then strengthen the validity of the quantum correction.

This is of particular importance if we return to the Solar Neutrino Problem discussion in the introduction. Upon measuring the cross section for $^{8}B$ breakup, the result can be used to calculate a value for the "astrophysical S-factor." When the cross section is written in a slightly different manner, this S-factor appears as one of the parameters. Its significance is that it relates directly to the predicted age of the universe! The S-factor has been calculated in various ways by many different researchers. As a result, there is a large discrepancy among predicted values of this S-factor. The heavy-ion induced $^{8}B$ breakup reaction performed recently by Motobayashi et al.[1] provides another way to derive the S-factor. Motobayashi's quote for the S-factor is significantly lower than those quoted in other experiments. However, the analysis used on the

8B breakup reaction was semiclassical, not fully quantum mechanical. The code in this paper brings us one step closer to analyzing the experimental data of Motobayashi's group and determining whether or not their quoted value is justified.

## SUMMARY

A computer code has been produced which determines the reaction cross section of heavy ion reactions between any two nuclei. This code can be applied to a wide variety of heavy ion reactions. This includes both present experiments and past experiments which have only been analyzed semiclassically. The analysis of both present and past experiments is vital in finding a resolution to the solar neutrino debate, and may also generate new ideas about heavy ion collisions.

**WORKS CITED**

[1] Collins, P.D.B., A.D. Martin, and E.J. Squires. Particle Physics and Cosmology, Wiley, New York, 1989.

[2] Simmons, L.M. (1984), "Science Underground: The Search for Rare Events," Los Alamos Science, Number 11, Summer/Fall: pp. 160-171.

[3] T. Motobayashi et al. (1994), "Coulomb Dissociation of $^{8}$B and the $^{7}$Be(p,$\gamma$)$^{8}$B Reaction at Low Energies," Physical Review Letters **73**, 2680.

[4] Athanassopoulos, C. et al. (1995), "Candidate Events in a Search For Anti-Muon-Neutrino $\rightarrow$ Anti-Electron-Neutrino Oscillations," submitted to Physical Review Letters.

[5] Jackson, John D. Classical Electrodynamics. Second edition. Wiley, New York, 1975.

[6] Bertulani, C.A., and G. Baur (1988), "Electromagnetic Processes in Relativistic Heavy Ion Collisions," Physics Reports **163**, 299.

[7] Norbury, John W. (1990), "Electric Quadropole Excitations in the Interactions of $^{89}$Y with Relativistic Nuclei," Physical Review **C41**, 372.

[8] Benesh, C.J., and J.L. Friar (1993), "Electromagnetic Dissociation of Nuclei in Heavy Ion Collisions," Physical Review **C48**, 1285.

[9] Benesh, C.J., A.C. Hayes, and J.L. Friar (1996), "A Quantum-Mechanical Equivalent-Photon Spectrum for Heavy-Ion Physics," submitted for publication.

[10] Norbury, John W. (1990), "Electric Quadropole Excitations in Relativistic Nucleus-Nucleus Collisions," Physical Review **C42**, 711.

[11] Press, William H. et al. Numerical Recipes in C: the Art of Scientific Computing. Cambridge University Press, New York, 1992.

## APPENDIX A: COMPUTER CODE

Program list:

working.c

QMN_E1_w.c

QMN_E2_w.c

N_E1_w.c

N_E2_w.c

sigma_WW.c

sigma_E2.c

K0.c[11]

K1.c[11]

B_min.c

```
/*-------------------------------------------------------------------
WORKING.C

This program will determine the integral of the following:
Weiszacker-Williams cross setion SigmaWW (=NE1(sigmaWW) )
deltaSigma = SigmaHI - SigmaWW (=NE2*sigmaE2 - NE1*sigmaE2 )

From these, the following will be determined:
Heavy Ion cross section SigmaHI (= SigmaWW - (NE2*sigmaE2-NE1*sigmaE2) )
E2 cross section SigmaE2 (=NE2*sigmaE2)
E1 cross section SigmaE1 (=SigmaHI - SigmaE2)

NOTES:  The float variables which hold these values will be as listed above.
The functions which calculate them will have an underline incorporated into
the name (like Sigma_E2).

Sigma_WW -- functions to calculate total cross section (sigma(w)*N(w))
Sigma_E2
Delta_Sigma

sigma_ww -- functions to calculate photon cross section (sigma(w))
sigma_E2
sigma_E1

N_E1_w   -- functions to calculate photon spectrum (N(w))
N_E2_w
-------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include <istream.h>
#include <iomanip.h>
#include <fstream.h>    //Needed to allow files to be read
#define HBARC 197.0
#define ALPHA 0.00730

struct nucleus
{
  int Anucl;         //Mass #
  int Znucl;         //Atomic #
  float chgradius;   //nuclear charge radius
  float sigma0;      //nucleus cross section    (fm^2)
  float w0;          //Giant Dipole resonance energy (MeV)
  float wE20;        //Giant Quadropole res. energy (MeV)
  float GAMMA;       //nucleus GDR peak width  (MeV)
  float GAMMAE2;     //nucleus GQR peak width (MeV)
  float nuclradius; //nuclear radius
};

float Sigma_WW(nucleus projectile, nucleus target, float gama);
float Sigma_E2(nucleus projectile, nucleus target, float gama);
float NE1sigma_E2(nucleus projectile, nucleus target, float gama);
float QMSigmaE1(nucleus projectile, nucleus target, float gama);
float QMSigmaE2(nucleus projectile, nucleus target, float gama);
float sigma_WW(float w, float sigma0, float w0, float GAMMA);
float sigma_E2(float w, nucleus target);
float N_E1_w(float w, float gama, float Zproj, float Aproj, float Atarg);
float N_E2_w(float w, float gama, float Zproj, float Aproj, float Atarg);
float QMN_E1_w(float w, float gama, float Zproj, float Aproj, float Atarg);
float QMN_E2_w(float w, float gama, float Zproj, float Aproj, float Atarg);
float B_min(float Aproj, float Atarg);

main()
{
```

```
  .

  float SigmaWW, SigmaHI, SigmaE1, SigmaE2, NE1sigmaE2, QM_SigE1, QM_SigE2, QMSig
  float gama;
  nucleus projectile, target;
  char projfile[20], targfile[20];
  FILE *fp;

//-----------------------------------------------------
//Read projectile nucleus parameters from file
//Also set value for gama (projectile kinetic energy)
//-----------------------------------------------------

  printf("Name of projectile file: ");    //Request projectile & target files
  gets(projfile);
  fp = fopen(projfile, "r");            //Open projectile file for reading
  if(fp == 0)
        { fprintf(stderr, "Can't open projectile file ");
          exit(1);}
  fscanf(fp," %d", &projectile.Anucl);  //Read projectile data
  fscanf(fp," %d", &projectile.Znucl);
  fscanf(fp," %f", &projectile.chgradius);
  fscanf(fp," %f", &projectile.sigma0);
  fscanf(fp," %f", &projectile.w0);
  fscanf(fp," %f", &projectile.wE20);
  fscanf(fp," %f", &projectile.GAMMA);
  fscanf(fp," %f", &projectile.GAMMAE2);
  fscanf(fp," %f", &projectile.nuclradius);
  fclose(fp);

  gama= 3.1;    //Relativistic gamma (Kinetic Energy - experimental parameter)

//-----------------------------------------
//Read target nucleus parameters from file
//-----------------------------------------

  printf("Name of target file: ");
  gets(targfile);
  fp = fopen(targfile, "r");
  if(fp == 0)
        { fprintf(stderr, "Can't open target file ");
          exit(1);}
  fscanf(fp," %d", &target.Anucl);
  fscanf(fp," %d", &target.Znucl);
  fscanf(fp," %f", &target.chgradius);
  fscanf(fp," %f", &target.sigma0);
  fscanf(fp," %f", &target.w0);
  fscanf(fp," %f", &target.wE20);
  fscanf(fp," %f", &target.GAMMA);
  fscanf(fp," %f", &target.GAMMAE2);
  fscanf(fp," %f", &target.nuclradius);
  fclose(fp);

//-----------------------------------------
//Call functions to calculate cross sections
//-----------------------------------------

        SigmaWW =Sigma_WW(projectile, target, gama);

        SigmaE2 =Sigma_E2(projectile, target, gama);
        NE1sigmaE2=NE1sigma_E2(projectile, target, gama);
        SigmaHI = SigmaWW + (SigmaE2 - NE1sigmaE2);
        SigmaE1 = SigmaHI - SigmaE2;
        QM_SigE1= QMSigmaE1(projectile, target, gama);
        QM_SigE2= QMSigmaE2(projectile, target, gama);
        QMSigmaHI = QM_SigE1 + QM_SigE2;
```

```
        printf("\n");
        printf("SigmaWW = %f\n", SigmaWW); //Wieszacker-Willians x-section
        printf("SigmaHI = %f\n", SigmaHI); //Heavy Ion x-section
        printf("SigmaE1 = %f\n", SigmaE1); //Dipole contrib. to HI x-section
        printf("SigmaE2 = %f\n", SigmaE2); //Quadropole cnt. to HI x-section
        printf("NE1sigmaE2 = %f\n", NE1sigmaE2); //part of delta in mid-calc.
        printf("QMSigmaE1 = %f\n", QM_SigE1); //QMDipole contrib. to HI x-section
        printf("QMSigmaE2 = %f\n", QM_SigE2); //QMQuadropole cnt. to HI x-section
        printf("QMSigmaHI = %f\n", QMSigmaHI); //QMHI x-section

        return 0;
}
//----------------END OF MAIN()-----------------------------------------


//------------BEGINNING OF FUNCTIONS-----------------------------------

float Sigma_WW(nucleus projectile, nucleus target, float gama)
{
        float wmin,wmax,steps,wcurr,wnext,delta,sum,range;
        float Aproj, Zproj, chgradius;
        float Atarg, Ztarg, sigma0, w0, wE20, GAMMA;
        int n;
        float sigmaWW_a, sigmaWW_b, NE1_a, NE1_b;
        float intervalarea;

       //Integration parameters:
          wmin=   8.0/HBARC;       //Minimum energy limit
          wmax=   50./HBARC;       //Maximum energy limit
          steps=  84.;             //# of integration steps
          range=  (wmax*HBARC)-(wmin*HBARC);
          delta=  ((wmax-wmin)/steps);
          sum =   0.;
       //Experimental Parameters
          gama=   gama;                          //Relativistic gamma
       //Particle parameters:
          Aproj=  projectile.Anucl;              //Projectile mass #
          Zproj=  projectile.Znucl;              //Projectile charge
          chgradius= projectile.chgradius;       //charge radius
          Atarg=  target.Anucl;                  //Target mass #
          sigma0= target.sigma0;                 //Photon cross-section
          w0=     (target.w0)/HBARC;             //Target GDR energy (1/fm)
          wE20=   (target.wE20)/HBARC;           //Target GQR energy (1/fm)
          GAMMA=  (target.GAMMA)/HBARC;          //Target peak spread (1/fm)

       //-----------------------------------------------//
       // Trapezoidal integration routine for SigmaWW //
       //-----------------------------------------------//
//TEST
//printf("sigmaWW\t NE1\t intervalarea\t sum\n");
        for(n=0;n<steps;n++)
        {
                wcurr  = wmin+(n*delta);
                wnext  = wmin+((n+1)*delta);
                sigmaWW_a= sigma_WW(wcurr, sigma0, w0, GAMMA);
                sigmaWW_b= sigma_WW(wnext, sigma0, w0, GAMMA);
                NE1_a    = N_E1_w(wcurr, gama, Zproj, Aproj, Atarg);
                NE1_b    = N_E1_w(wnext, gama, Zproj, Aproj, Atarg);

              intervalarea = (((sigmaWW_a*NE1_a) + (sigmaWW_b*NE1_b))/2)*delta;
              sum = sum + intervalarea;
//TEST
//printf("%f\t%f\t%f\t%f\n", sigmaWW_a, NE1_a, intervalarea,sum);

        }
        return sum;
```

```
}
//-------------------------------------------------------------------------
float Sigma_E2(nucleus projectile, nucleus target, float gama)
{
        float wmin,wmax,steps,wcurr,wnext,delta,sum,range;
        float Aproj, Zproj, chgradius;
        float Atarg, Ztarg, sigma0, w0, wE20, GAMMA;
        int n;
        float intervalarea, sigmaE2_a, sigmaE2_b, NE2_a, NE2_b;

       //Integration parameters:
          wmin=   8.0/HBARC;       //Minimum energy limit
          wmax=   50./HBARC;       //Maximum energy limit
          steps=  42.;             //# of integration steps
          range=  (wmax*HBARC)-(wmin*HBARC);
          delta=  ((wmax-wmin)/steps);
          sum =   0.;
       //Experimental Parameters
          gama=   gama;                        //Relativistic gamma
       //Particle parameters:
          Aproj=  projectile.Anucl;            //Projectile mass #
          Zproj=  projectile.Znucl;            //Projectile charge
          chgradius= projectile.chgradius;     //charge radius
          Atarg=  target.Anucl;                //Target mass #
          sigma0= target.sigma0;               //Photon cross-section
          w0=     (target.w0)/HBARC;           //Target GDR energy (1/fm)
          wE20=   (target.wE20)/HBARC;         //Target GQR energy (1/fm)
          GAMMA=  (target.GAMMA)/HBARC;        //Target peak spread (1/fm)

       //---------------------------------------------//
       // Trapezoidal integration routine for SigmaE2 //
       //---------------------------------------------//
//TEST
printf("sigmaE2\t NE2\t intervalarea\t sum\n");
          for(n=0;n<steps;n++)
            {
              wcurr  = wmin+(n*delta);
              wnext  = wmin+((n+1)*delta);
              sigmaE2_a=  sigma_E2(wcurr, target);
              sigmaE2_b=  sigma_E2(wnext, target);
              NE2_a=    N_E2_w(wcurr, gama, Zproj, Aproj, Atarg);
              NE2_b=    N_E2_w(wnext, gama, Zproj, Aproj, Atarg);

              intervalarea = (((sigmaE2_a*NE2_a) + (sigmaE2_b*NE2_b))/2)*delta;
              sum = sum + intervalarea;
//TEST
printf("%f\t%f\t%f\t%f\n", sigmaE2_a, NE2_a, intervalarea,sum);

            }
          return sum;
}

//-------------------------------------------------------------------------
float NE1sigma_E2(nucleus projectile, nucleus target, float gama)
{
        float wmin,wmax,steps,wcurr,wnext,delta,sum,range;
        float Aproj, Zproj, chgradius;
        float Atarg, Ztarg, sigma0, w0, wE20, GAMMA;
        int n;
        float intervalarea, sigmaE2_a, sigmaE2_b, NE1_a, NE1_b;

       //Integration parameters:
          wmin=   8.0/HBARC;       //Minimum energy limit
          wmax=   50./HBARC;       //Maximum energy limit
          steps=  42.;             //# of integration steps
          range=  (wmax*HBARC)-(wmin*HBARC);
```

```
        delta=  ((wmax-wmin)/steps);
        sum =   0.;
      //Experimental Parameters
        gama=   gama;                       //Relativistic gamma
      //Particle parameters:
        Aproj=  projectile.Anucl;           //Projectile mass #
        Zproj=  projectile.Znucl;           //Projectile charge
        chgradius= projectile.chgradius;    //charge radius
        Atarg=  target.Anucl;               //Target mass #
        sigma0= target.sigma0;              //Photon cross-section
        w0=     (target.w0)/HBARC;          //Target GDR energy (1/fm)
        wE20=   (target.wE20)/HBARC;        //Target GQR energy (1/fm)
        GAMMA=  (target.GAMMA)/HBARC;       //Target peak spread (1/fm)

      //-------------------------------------------------//
      // Trapezoidal integration routine for NE1sigma_E2 //
      //-------------------------------------------------//

//TEST
printf("sigmaE2\t\t NE1\t\t intervalarea\t sum\n");

        for(n=0;n<steps;n++)
        {
                wcurr  = wmin+(n*delta);
                wnext  = wmin+((n+1)*delta);
                sigmaE2_a= sigma_E2(wcurr, target);
                sigmaE2_b= sigma_E2(wnext, target);
                NE1_a    = N_E1_w(wcurr, gama, Zproj, Aproj, Atarg);
                NE1_b    = N_E1_w(wnext, gama, Zproj, Aproj, Atarg);

                intervalarea = (((sigmaE2_a*NE1_a) + (sigmaE2_b*NE1_b))/2)*delta;
                sum = sum + intervalarea;
//TEST
printf("%f\t%f\t%f\t%f\n", sigmaE2_a, NE1_a, intervalarea,sum);

        }
        return sum;
}
//-----------------------------------------------------------------------
float QMSigmaE1(nucleus projectile, nucleus target, float gama)
{
        float wmin,wmax,steps,wcurr,wnext,delta,sum,range;
        float Aproj, Zproj, chgradius;
        float Atarg, Ztarg, sigma0, w0, wE20, GAMMA;
        int n;
        float intervalarea, sigmaWW_a, sigmaWW_b, QMNE1_a, QMNE1_b;

      //Integration parameters:
        wmin=   8.0/HBARC;      //Minimum energy limit
        wmax=   50./HBARC;      //Maximum energy limit
        steps=  42.;            //# of integration steps
        range=  (wmax*HBARC)-(wmin*HBARC);
        delta=  ((wmax-wmin)/steps);
        sum =   0.;
      //Experimental Parameters
        gama=   gama;                       //Relativistic gamma
      //Particle parameters:
        Aproj=  projectile.Anucl;           //Projectile mass #
        Zproj=  projectile.Znucl;           //Projectile charge
        chgradius= projectile.chgradius;    //charge radius
        Atarg=  target.Anucl;               //Target mass #
        sigma0= target.sigma0;              //Photon cross-section
        w0=     (target.w0)/HBARC;          //Target GDR energy (1/fm)
        wE20=   (target.wE20)/HBARC;        //Target GQR energy (1/fm)
        GAMMA=  (target.GAMMA)/HBARC;       //Target peak spread (1/fm)
```

```
      //-------------------------------------------------//
      // Trapezoidal integration routine for QMSigmaE1 //
      //-------------------------------------------------//
//TEST
printf("sigmaE1\t QMNE1\t intervalarea\t sum\n");
         for(n=0;n<steps;n++)
           {
             wcurr  = wmin+(n*delta);
             wnext  = wmin+((n+1)*delta);
             sigmaWW_a=  sigma_WW(wcurr, sigma0, w0, GAMMA);
             sigmaWW_b=  sigma_WW(wnext, sigma0, w0, GAMMA);
             QMNE1_a=    QMN_E1_w(wcurr, gama, Zproj, Aproj, Atarg);
             QMNE1_b=    QMN_E1_w(wnext, gama, Zproj, Aproj, Atarg);

             intervalarea = (((sigmaWW_a*QMNE1_a) + (sigmaWW_b*QMNE1_b))/2)*delt
             sum = sum + intervalarea;
//TEST
printf("%f\t%f\t%f\t%f\n", sigmaWW_a, QMNE1_a, intervalarea,sum);

           }
         return sum;
}

//-------------------------------------------------------------------------
float QMSigmaE2(nucleus projectile, nucleus target, float gama)

{
        float wmin,wmax,steps,wcurr,wnext,delta,sum,range;
        float Aproj, Zproj, chgradius;
        float Atarg, Ztarg, sigma0, w0, wE20, GAMMA;
        int n;
        float intervalarea, sigmaE2_a, sigmaE2_b, QMNE2_a, QMNE2_b;

      //Integration parameters:
         wmin=   8.0/HBARC;       //Minimum energy limit
         wmax=   50./HBARC;       //Maximum energy limit
         steps=  42.;             //# of integration steps
         range=  (wmax*HBARC)-(wmin*HBARC);
         delta=  ((wmax-wmin)/steps);
         sum =   0.;
      //Experimental Parameters
         gama=   gama;                          //Relativistic gamma
      //Particle parameters:
         Aproj=  projectile.Anucl;              //Projectile mass #
         Zproj=  projectile.Znucl;              //Projectile charge
         chgradius= projectile.chgradius;       //charge radius
         Atarg=  target.Anucl;                  //Target mass #
         sigma0= target.sigma0;                 //Photon cross-section
         w0=     (target.w0)/HBARC;             //Target GDR energy (1/fm)
         wE20=   (target.wE20)/HBARC;           //Target GQR energy (1/fm)
         GAMMA=  (target.GAMMA)/HBARC;          //Target peak spread (1/fm)

      //-------------------------------------------------//
      // Trapezoidal integration routine for QMSigmaE2 //
      //-------------------------------------------------//
//TEST
printf("sigmaE2\t QMNE2\t intervalarea\t sum\n");
         for(n=0;n<steps;n++)
           {
             wcurr  = wmin+(n*delta);
             wnext  = wmin+((n+1)*delta);
             sigmaE2_a=  sigma_E2(wcurr, target);
             sigmaE2_b=  sigma_E2(wnext, target);
             QMNE2_a=    QMN_E2_w(wcurr, gama, Zproj, Aproj, Atarg);
             QMNE2_b=    QMN_E2_w(wnext, gama, Zproj, Aproj, Atarg);
```

```
            intervalarea = (((sigmaE2_a*QMNE2_a) + (sigmaE2_b*QMNE2_b))/2)*delt
            sum = sum + intervalarea;
//TEST
printf("%f\t%f\t%f\t%f\n", sigmaE2_a, QMNE2_a, intervalarea,sum);

          }
        return sum;
}
```

```
/*-------------------------------------------------------------------------
QMN_E1_W.C

Program to calculate quantum mechanical spectrum QMN_E1_w.
-------------------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>
#define PI 3.14159265
#define HBARC 197.0
#define ALPHA 0.00730

float B_min(float Aproj, float Atarg);

float QMN_E1_w(float w, float gama, float Zproj, float Aproj, float Atarg)
{

float LAMBDA, projrad, quantumNE1, quantumNE2, b_min, v;

        b_min = B_min(Aproj, Atarg);
        v = sqrt(1- (1/(gama*gama)));
        LAMBDA = sqrt( 1/(b_min*b_min) - (w*w)/(gama*gama*v*v) );
        projrad = 1.2*pow(Aproj, (1./3.));  // Projectile radius

quantumNE1 = (2*Zproj*Zproj*ALPHA/(PI*w*v*v)) *
             (
             log(LAMBDA*gama*v/w)*
             (1.+(1./3.)*( (w*w*projrad*projrad)/(gama*gama) ) )
             -
             (v*v/2)
             +
             (w*w/(2*gama*gama*LAMBDA*LAMBDA) )
             -
             (LAMBDA*LAMBDA*projrad*projrad/6 )
             );

return quantumNE1;
}
```

```
/*-----------------------------------------------------------------------
QMN_E2_W.C

Program to calculate quantum mechanical spectrum QMN_E2_w.
-----------------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>
#define PI 3.14159265
#define HBARC 197.0
#define ALPHA 0.00730

float B_min(float Aproj, float Atarg);

float QMN_E2_w(float w, float gama, float Zproj, float Aproj, float Atarg)
{

float LAMBDA, projrad, b_min, v, quantumNE1, quantumNE2;

        b_min = B_min(Aproj, Atarg);
        v = sqrt(1- (1/(gama*gama)));
        LAMBDA = sqrt( 1/(b_min*b_min) - (w*w)/(gama*gama*v*v) );
        projrad = 1.2*pow(Aproj, (1./3.));  // Projectile radius

quantumNE2 = ( (2*Zproj*Zproj*ALPHA)/(PI*w*v*v) )*
             (
             log(LAMBDA*gama*v/w)*
             ( (v*v) +(1./3.)*( (w*w*projrad*projrad)/(gama*gama) ) )
             -
             (v*v/2)
             +
             (w*w/(2*gama*gama*LAMBDA*LAMBDA) )
             +
             (2*LAMBDA*LAMBDA/(3*w*w) )*
               (1 - (w*w*projrad*projrad/6) + (3*w*w*projrad*projrad/(4*gama*gama
             -
             ( ( ( LAMBDA*LAMBDA + ((w*w)/(v*v)) )*
               ( LAMBDA*LAMBDA + ((w*w)/(v*v)) )-
               (w*w*w*w/(v*v*v*v) )
             )
             *projrad*projrad )/(9.*w*w)
             );

return quantumNE2;
}
```

```
/*-----------------------------------------------------
N_E1_W.C

Program to calculate the photon spectrum N(w) via the semi-
classical method of virtual quanta as presented in J.D.
Jackson's "Classical Electrodynamics"
(see page 723-4, 15.54, 15.55, 15.58)

N(w) is represented as "number_spec"
-----------------------------------------------------*/
#include <math.h>
#include <stdio.h>
#define PI 3.14159265
#define HBARC 197.0
#define ALPHA 0.00730

float K0(float);
float K1(float);
float B_min(float Aproj, float Atarg);

float N_E1_w(float w, float gama, float Zproj, float Aproj, float Atarg)
{
  float b_min,v,x;
  float dI_dw, number_spec;

/*-------------------------------------------------------------------
Variable Comments:

* "gama" represents relativistic gamma (dimensionless)
* As is standard, hbar and c are set equal to 1
* w is the photon energy, in units of (1/fm)
* Zproj*ALPHA is the charge of the projectile nucleus

* UNITS:
    b_min: fm
    gama:  -
    Zproj*ALPHA: ?
    v:     -
    x:     ?

-------------------------------------------------------------------*/

        b_min = B_min(Aproj, Atarg);
//        gama = 3.1;
//        Zproj = 6.; // charge of projectile (dimensionless)
        v = sqrt(1- (1/(gama*gama))); // v is velocity in dimensionless units (ov
        x = (w*b_min)/(gama*v);       // x is dimensionless

        dI_dw = ( (2.*Zproj*Zproj*ALPHA)/(PI*v*v) ) * ( (x*K0(x)*K1(x))-( (v*v*x*
*( (K1(x)*K1(x))-(K0(x)*K0(x)) )));
        number_spec = dI_dw/w;
//-------------------------------------------------------------
//  Print a table of values for the photon spectrum
//-------------------------------------------------------------

        printf("%f\t%f\t%f\n", w,x,number_spec);
        return number_spec;
}
```

```
/*-------------------------------------------------------------
N_E2_W.C

Program to calculate the E2 (quadropole) photon spectrum N_E2(w)
-------------------------------------------------------------*/
#include <math.h>
#include <stdio.h>
#define PI 3.14159265
#define HBARC 197.0
#define ALPHA 0.00730

float K0(float);
float K1(float);
float B_min(float Aproj, float Atarg);

float N_E2_w(float w, float gama, float Zproj, float Aproj, float Atarg)
{
  float b_min,v,x, k0, k1;
  float NE2;

/*-------------------------------------------------------------------
Variable Comments:
  Paramaters represent the same things as in N_w.c
-------------------------------------------------------------------*/

        b_min = B_min(Aproj, Atarg);
        v = sqrt(1.- (1./(gama*gama))); // v is velocity (dimensionless)
        x = (w*b_min)/(gama*v);        // x is dimensionless

        k0 = K0(x);
        k1 = K1(x);

/*-----------------------------------------
 Calculates the E2 (quadropole) number spectrum

// Line 1: coefficient
// Line 2: K1 squared term
// Line 3: K0K1 term
// Line 4: K1 squared - K0 squared term
-----------------------------------------*/
printf("Zproj = %f\n", Zproj);
printf("w = %f\n", w);
printf("v = %f\n", v);
printf("k0 = %f\n", k0);
printf("k1 = %f\n", k1);
printf("bmin = %f\n", b_min);



        NE2 = (2.*(Zproj*Zproj)*ALPHA/(PI*(v*v*v*v))) *
              (
              (2.*(K1(x)*K1(x))/(gama*gama) +
              ( x*K0(x)*K1(x)*(1.+(1./(gama*gama)))*(1.+(1./(gama*gama))) ) -
              ( x*x*v*v*v*v* (K1(x)*K1(x)-K0(x)*K0(x))/2.) )
              );
        NE2 = NE2/w;

//-------------------------------------------------------------
//  Print a table of values for the photon spectrum
//-------------------------------------------------------------
        printf("w\t\tx\t\tN_E2\t\tK0\t\tK1\n");
        printf("%f\t%f\t%f\t%f\t%f\n", w,x,NE2,k0,k1);
        return NE2;
}
```

```
/*-------------------------------------------------------------------------
SIGMA_WW.C

This program returns a value for the photon cross section (represented by
"sigma_w"; called SIGMAgamma(omega)) for w0, y, and sigma0 given IN
THIS PROGRAM'S CODE.  The function also requires that a value for omega (w)
be passed to it.
-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include <istream.h>
#include <iomanip.h>
#define HBARC 197.0

float sigma_WW(float w, float sigma0, float w0, float GAMMA)
{
float sigma_w;

        sigma_w = (sigma0 / (1. + ( ( (w*w)-(w0*w0) )*( (w*w)-(w0*w0) )/(w*GAMMA*
        return sigma_w;
}
```

```
/*-------------------------------------------------------------------------
SIGMA_E2.C

This program returns a value for the QUADROPOLE photon cross section
(represented by "sigma_w"; called SIGMAgamma(omega)) for w0, y, and
sigma0 given IN THIS PROGRAM'S CODE.  The function also requires that a
value for omega (w) be passed to it.
-------------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <iostream.h>
#include <istream.h>
#include <iomanip.h>
#define HBARC 197.0
#define PI 3.1414526536

struct nucleus
{
  int Anucl;         //Mass #
  int Znucl;         //Atomic #
  float chgradius;   //nuclear charge radius
  float sigma0;      //nucleus cross section
  float w0;          //nucleus resonance energy
  float wE20;        //Giant Quadropole res. energy (MeV)
  float GAMMA;       //nucleus GDR peak width  (MeV)
  float GAMMAE2;     //nucleus GQR peak width (MeV)
  float nuclradius; //nuclear radius
};

float sigma_E2(float w, nucleus target)
{
  float f, GAMMAE2, wE20, sigmaEWSR, sigmaE2;

//TEST
//printf("w\t\t wE20\t\t sigmaEWSR\t sigmaE2\n");

        f = 0.95;  // parameter for 197Au
        wE20=  target.wE20/HBARC;
        GAMMAE2=  target.GAMMAE2/HBARC;
//TEST
//printf("f = %f\n", f);
//printf("pi = %f\n", PI);
//printf("wE20 = %f\n", wE20);
//printf("target.GAMMAE2 = %f\n", target.GAMMAE2);
//printf("GAMMAE2 = %f\n", GAMMAE2);
//printf("target.Znucl = %d\n", target.Znucl);
//printf("target.Anucl = %d\n", target.Znucl);

        sigmaEWSR = 2.*(f*0.22*79.*pow(197., 2./3.))/
                    (PI*target.GAMMAE2);  //UNITS: (ub*MeV^2)

//printf("sigmaEWSR (in ub*MeV^2) = %f\n", sigmaEWSR);

      //CONVERT sigmaEWSR TO (fm^4)
        sigmaEWSR = sigmaEWSR*(197.*197./10000.);

//printf("sigmaEWSR (in fm^4) = %f\n", sigmaEWSR);

        sigmaE2 = (sigmaEWSR*wE20*wE20)/
                  ( 1 +
                  ( ( (w*w-wE20*wE20)*(w*w-wE20*wE20) )/(w*w*GAMMAE2*GAMMAE2) )
                  );      //UNITS: (fm^2)
//TEST
//printf("%f\t %f\t %f\t %f\n", w, wE20, sigmaEWSR, sigmaE2);
```

```
        return sigmaE2;
}
```

```
/*-------------------------------------------------
KO.C

Function to calculate the Bessel function K1(x)
(consists of two functions from Numerical Recipies:
bessk0.c and bessi0.c)

NOTES:
* The iscssun g++ compiler does not allow the use
        of the "cin" and "cout" functions
-------------------------------------------------*/
#include <iomanip.h>
#include <math.h>

float bessk0(float);
float bessi0(float);

float K0(float var)
{
        float ans;
        ans = bessk0(var);
        return ans;
}

float bessk0(float x)
{
        float bessi0(float x);
        double y,ans;

        if (x <= 2.0) {
                y=x*x/4.0;
                ans=(-log(x/2.0)*bessi0(x))+(-0.57721566+y*(0.42278420
                        +y*(0.23069756+y*(0.3488590e-1+y*(0.262698e-2
                        +y*(0.10750e-3+y*0.74e-5))))));
        } else {
                y=2.0/x;
                ans=(exp(-x)/sqrt(x))*(1.25331414+y*(-0.7832358e-1
                        +y*(0.2189568e-1+y*(-0.1062446e-1+y*(0.587872e-2
                        +y*(-0.251540e-2+y*0.53208e-3))))));
        }
        return ans;
}

float bessi0(float x)
{
        float ax,ans;
        double y;

        if ((ax=fabs(x)) < 3.75) {
                y=x/3.75;
                y*=y;
                ans=1.0+y*(3.5156229+y*(3.0899424+y*(1.2067492
                        +y*(0.2659732+y*(0.360768e-1+y*0.45813e-2)))));
        } else {
                y=3.75/ax;
                ans=(exp(ax)/sqrt(ax))*(0.39894228+y*(0.1328592e-1
                        +y*(0.225319e-2+y*(-0.157565e-2+y*(0.916281e-2
                        +y*(-0.2057706e-1+y*(0.2635537e-1+y*(-0.1647633e-1
                        +y*0.392377e-2)))))));
        }
        return ans;
}
/* (C) Copr. 1986-92 Numerical Recipes Software #Q2Z3A`o. */
```

```
/*-------------------------------------------------
K1.C

Function to calculate the Bessel function K1(x)
(consists of two functions from Numerical Recipies:
besskl.c and bessi1.c)

NOTES:
* The iscssun compiler will not allow the use of
        the "cin" and "cout" functions.
-------------------------------------------------*/
#include <iostream.h>
#include <math.h>

float bessk1(float);
float bessi1(float);

float K1(float var)
{
        float ans;
        ans = bessk1(var);
        return ans;
}

float bessk1(float x)
{
        float bessi1(float x);
        double y,ans;

        if (x <= 2.0) {
                y=x*x/4.0;
                ans=(log(x/2.0)*bessi1(x))+(1.0/x)*(1.0+y*(0.15443144
                        +y*(-0.67278579+y*(-0.18156897+y*(-0.1919402e-1
                        +y*(-0.110404e-2+y*(-0.4686e-4)))))));
        } else {
                y=2.0/x;
                ans=(exp(-x)/sqrt(x))*(1.25331414+y*(0.23498619
                        +y*(-0.3655620e-1+y*(0.1504268e-1+y*(-0.780353e-2
                        +y*(0.325614e-2+y*(-0.68245e-3)))))));
        }
        return ans;
}

float bessi1(float x)
{
        float ax,ans;
        double y;

        if ((ax=fabs(x)) < 3.75) {
                y=x/3.75;
                y*=y;
                ans=ax*(0.5+y*(0.87890594+y*(0.51498869+y*(0.15084934
                        +y*(0.2658733e-1+y*(0.301532e-2+y*0.32411e-3))))));
        } else {
                y=3.75/ax;
                ans=0.2282967e-1+y*(-0.2895312e-1+y*(0.1787654e-1
                        -y*0.420059e-2));
                ans=0.39894228+y*(-0.3988024e-1+y*(-0.362018e-2
                        +y*(0.163801e-2+y*(-0.1031555e-1+y*ans))));
                ans *= (exp(ax)/sqrt(ax));
        }
        return x < 0.0 ? -ans : ans;
}
/* (C) Copr. 1986-92 Numerical Recipes Software #Q2Z3A`o. */
```

```
//-------------------------------------------------------------------
//B_MIN.C
//
//B_min() calculates the minimum impact parameter b_min as a function
//of the nuclear masses of the projectile and target (Aproj and Atarg
//respectively).  The calculation below gives a result b_min in units
//of (fm).
//-------------------------------------------------------------------

#include <stdio.h>
#include <math.h>

float B_min(float Aproj, float Atarg)
{
        float b_min, Aproj3, Atarg3;

        Aproj3 = pow(Aproj, 1./3.);
        Atarg3 = pow(Atarg, 1./3.);
        b_min = (1.34 * ((Aproj3 + Atarg3) - (0.75*((1/Aproj3)+(1/Atarg3)))));
        return b_min;
}
```