

October - 1992

**A NUMERICAL PROBLEM WITH  
NJOY89/BROADR**

**E.B. Webster  
October 1992**

	NAME	SIGNATURE	POSITION	DATE
Lead Author	Mr. E.B Webster	<i>E.B. Webster</i>	PMG1	7/12/92
Checked	Mr. C.J. Dean	<i>C.J. Dean</i>	PMG2	7/12/92
Approved	Dr. M.J. Halsall	<i>M.J. Halsall</i>	Section Leader	8/12/92

**Reactor Physics, Shielding and Criticality Department  
Safety Engineering Systems Division  
AEA Reactor Services  
Winfrith Technology Centre**

The information which this report contains is accurate to the best knowledge and belief of the UKAEA, but neither the UKAEA nor any person acting on behalf of the UKAEA make any warranty or representation expressed or implied with respect to the accuracy, completeness or usefulness of this information, nor assume any liabilities with respect to the use of, or with respect to any damages which may result from the use of any information, apparatus, method or process disclosed in this report.

# **A NUMERICAL PROBLEM WITH NJOY89/BROADR**

**E.B. Webster**

## **SUMMARY**

This note describes a numerical problem with the NJOY89 module BROADR, which was discovered while performing some simple exercises with NJOY, including the Doppler broadening of a simple  $1/v$  cross-section to high temperature.

The quality assurance exercise having brought the problem to light, it was necessary to investigate it. The investigation is described, and some possible circumventions are given, together with their effects upon computing time.

**Reactor Physics, Shielding and Criticality Department  
Safety Engineering Systems Division  
AEA Reactor Services  
Winfrith Technology Centre**

# A NUMERICAL PROBLEM WITH NJOY89/BROADR

## 1. INTRODUCTION

This note describes a numerical problem with the BROADR module of NJOY89, when Doppler broadening cross-sections to high temperatures, and proposes possible solutions.

The Nuclear Data and Fast Reactor Methods Team of Reactor Physics, Shielding and Criticality Department, AEA Reactor Services, use NJOY89 a great deal to process nuclear data (JEF2.2) for thermal reactor physics (WIMS), fast reactor physics (the European cell code ECCO), for Monte Carlo criticality applications (MONK6) and for shielding applications (MCBEND). We employ a double precision version of NJOY89, running on SUN Microsystems SPARCstation workstations. The method of achieving double precision on a 32-bit machine is to use the -R8 option of the SUN Microsystems Fortran 1.4 compiler, which automatically doubles the storage for both real and integer variables, and converts all real arithmetic to double precision. This method of doubling NJOY89 has the advantage of not requiring changes to the source code. (It is quite easy to double NJOY89 using TOOLPACK tools to convert the source code to double precision, but this makes the installation and generation of updates rather difficult.)

As part of a Quality Assurance exercise, some simple NJOY89 exercises were performed, including the Doppler broadening of a  $1/v$  like cross-section,  $B_{10}(n,\alpha)$  to high temperatures. Some results of this sort of exercise are given in the NJOY User's Manual. In one of our cases, where the temperature was set at 15keV (174MK), the results of plotting the group cross-sections were not as expected.

This note describes how the problem was investigated, and proposes some solutions.

## 2. INVESTIGATION OF PROBLEM

Examination of the point cross-sections produced by RECONR showed no anomalies, but the point cross-sections from GROUPT had a number of curious features (a hump, spikes and zero values), as is shown in Figure 1.

Attention was turned to the BSGMA subroutine of BROADR, which is called to generate the broadened cross-section for any given energy value.

It was decided to concentrate upon the zero cross-section values (which may well be negative ones - BSGMA sets any negative values to zero) since they are easy to find. The BSGMA routine was 'instrumented', so that when it had calculated a zero cross-section for a given energy value, a flag was set to switch on tracing, and the whole BSGMA calculation was repeated with debug prints.

It became clear that the negative cross-sections were being generated in the section of code starting "loop over other intervals " (BROADR 01053), shown below. The numerical difficulty had to be caused by the routine HUNKY, which is used to calculate the values of the variables  $s_1$  and  $s_2$ , used in adding terms to the cross-section,  $sbt(i)$ .

```

c      loop over other intervals                                BROADR  01053
do 220 l=klow,khml                                           BROADR  01054
x=e(l+1)                                                       BROADR  01055
aa=x-y                                                         BROADR  01056
xx=e(l)**2                                                     BROADR  01057
if (xx.eq.x*x) go to 220                                       BROADR  01058
call hunky                                                     BROADR  01059
denom=1./(x*x-xx)                                             BROADR  01060
do 218 i=1,nreac                                             BROADR  01061
slope=(s(i,l+1)-s(i,l))*denom                                   BROADR  01062
218 sbt(i)=sbt(i)-s(i,l)*s1-slope*s2                           BROADR  01063
      if(l11.eq.1) write(6,*) 'SBT, DELTA ', sbt(1),          DEBUG
      &                -s(i,l)*s1-slope*s2                  DEBUG
      if (aa.gt.atop) go to 230                                BROADR  01064
220 continue                                                  BROADR  01065

```

HUNKY normally works as follows. It has previously calculated a set of values  $f(1:5)$  for a value of a parameter  $a$  which is now held in  $alast$ . It initialises  $h(1:5)$  to the old value of  $f(1:5)$ , and calls HUNKY to calculate new values of  $f(1:5)$  for the current value of  $a$ .

The  $h(1:5)$  are then calculated using

$$h(1:5) = h(1:5) - f(1:5)$$

However, if the difference between the old and new  $f(1:5)$  values is small

$$h(k) \leq 1.0e-5 * \text{abs}(f(k)),$$

then  $h(k)$  is recalculated by a different method, using the routine HNABB.

A test was performed by setting the value of the tolerance to a large value (20.0) for calls to HUNKY in the above section of code only. The anomalies in the broadened  $B10(N,\alpha)$  point cross-section disappeared.

The setting of the HUNKY tolerance to 20.0 was removed, and some extra code added to HUNKY so that if the zero cross-section detected flag (l11) were set, the variables  $s1$  and  $s2$  were calculated in HUNKY both by using the standard  $h$  values and by using ones calculated using the better, but more expensive HNABB routines.

The 'exact' and 'approximate'  $s1$  and  $s2$  values ( $s1/s2$  and  $s01/s02$ ) were used in BSIKMA to print out the contributions to the cross-section, thus:

```

do 218 i=1,nreac                                             BROADR  01061
slope=(s(i,l+1)-s(i,l))*denom                                   BROADR  01062
218 sbt(i)=sbt(i)-s(i,l)*s1-slope*s2                           BROADR  01063
      if(l11.eq.1) write(6,*) 'SBT, DELTA 1,2 ', sbt(1),
      &                -s(i,l)*s1-slope*s2 , -s(i,l)*s01-slope*s02
      if (aa.gt.atop) go to 230                                BROADR  01064

```

Study of the "s" variables and the corresponding values of h made it clear that differences in the seventh and eighth place of the h(1) term could cause large differences in the value and sign of the contribution to the cross-section.

This suggested that the approximation used for calculating f(1) was inadequate.

The code used to calculate f(1) is

```

data a0,a1,a2,a3,a4,a5/.3275911,.254829592,-.284496736,
&          1.421413741,-1.453152027,1.061405429/
asq=a*a
expo=exp(-asq)
r=1./(1.+a0*a)
f(1)=0.5*expo*r*(a1+r*(a2+r*(a3+r*(a4+a5*r))))

```

and is taken from Ref. 1.

f(1) is  $\frac{\text{erfc}(a)}{2}$ , half the complementary error function.

A new approximation was found in Ref. 2 for *erfc(a)*, which returns the complementary error function to better than 1.2e-7 everywhere. The code for f(1), using this new approximation is:

```

t = 1.0/ (1.0+0.5*a)
f(1) = 0.5*t*exp(-a*a-1.26551223+t* (1.00002368+t* (0.37409196+
&      t* (0.09678418+t* (-0.18628806+t* (0.27886807+t* (-1.13520398+
&      t* (1.48851587+t* (-0.82215223+t*0.17087277))))))))))

```

The function funky was recoded using the new approximation, and gave much better results for the B10(n,α) reaction. They were, however, not perfect, with a few small spikes still remaining, as may be seen in Figure 2.

The third approximation for *erfc(a)* was that in the NAG FORTRAN subroutine library, Ref. 3, in the version which is said to give 16 significant figures. Use of this in the FUNKY routine resulted in the complete elimination of the spikes, as is shown in Figure 3.

### 3. EFFECTS UPON EXECUTION TIME OF DIFFERENT *erfc(a)* APPROXIMATIONS

There are two questions to be asked concerning the effects upon execution time of the three approximations used to calculate *erfc(a)*. The first is: what is the effect upon the B10(n,α) case at 15keV? The number of points tabulated by RECONR was 394, and the number of points generated by BROADR and the corresponding execution times for BROADR, using the three approximations for *erfc(a)* were as follows:

Standard NJOY89 approximation, Ref. 1	NUMERICAL RECIPES approximation, Ref. 2	NAG Library approximation, (16 sig. figs) Ref. 3
#points generated = 547	#points generated = 442	#points generated = 398
BROADR run time = 167.18s	BROADR run time = 101.58s	BROADR run time = 51.17s

It is clear that use of the best approximation to *erfc(a)*, the NAG16 library function, has resulted in fewer calls to BSIGMA, and therefore a much reduced execution time.

The second question is: what are the relative execution times for the three approximations to  $f(1)$ , in other circumstances where the Ref. 1 approximation is perfectly adequate? A simple program was written to time 20,000 evaluations of each of the approximations, plus the NAG one giving 8 significant figures, and to compare the errors of the standard NJOY89 approximation, the "Numerical Recipes" approximation and the "NAG8" one with the "NAG16" approximation. The program is shown in Figure 4.

The times were:

```
TIMES XNAG16 0.8900 XNUM 0.6500 XFUNKY 0.5600 XNAG8 0.7100
```

indicating that the "Numerical Methods" approximation is 16% slower than the original NJOY89 approximation, while the NAG8 one is 27% slower and the NAG16 is 59% slower, per invocation. Some of the errors for the NJOY89, "Numerical Methods" and NAG8 approximations, relative to the NAG16 approximation, are shown in Figure 5.

#### 4. DISCUSSION

A numerical problem has been identified in the NJOY89 module BROADR, when Doppler broadening cross-sections to very high temperatures. The cause of the problem has been identified as the (relatively) poor accuracy of the approximation used for the complementary error function at around  $x=1.42$ , a value which is used in high temperature broadening.

One work-around is to replace the original approximation by one which costs little more to execute (per invocation), taken from Ref. 2. Use of this approximation reduced the BROADR execution time by 39%, for the high temperature case studied. However, a better one, which takes 59% longer to execute (per invocation), is available from Ref. 3. For the high temperature B10( $n,\alpha$ ) case, this approximation reduced the standard NJOY89 execution time by 69%.

The possibility was examined that by using the NAG16 approximation for  $erfc(a)$ , the accuracy would be such that a satisfactory accuracy for the  $h(1)$ , say 7 significant figures, might be achieved even when the number of calls to HNABB was reduced by tightening the tolerance to read:

$$h(k) \leq 1.0e-8 * \text{abs}(f(k))$$

The effect of using the NAG16 approximation and tightening the above tolerance to  $1.0e-8$  in the 15keV B10 case was to reduce the run time from 51.17 to 33.36 seconds. HNABB was, even in this case, called 167,521 times. The cross-section was not affected by using the  $1.0e-8$  tolerance.

To examine whether use of the NAG16 approximation, together with the  $1.0e-8$  tolerance would have any effect on cases where broadening was performed to a temperature used in reactor operation, a test was performed using JEF2.2 PU240, broadened to 1500K. Use of the NAG16 approximation together with the standard  $1.0e-5$  tolerance caused an increase in run time of 40.2% over that for the standard NJOY89.62W. However, when the NAG16 approximation was used with a tolerance of  $1.0e-8$ , the run time became 12% less than the standard NAG89.62W, with a corresponding decrease in the number of calls to HNABB from 1,776,072 to 756,093.

Two versions of NJOY89 were then built in exactly the same way. The differences were in the source of BROADR. In one, the standard BROADR was used. In the other, the tolerance  $1.0e-5$  was altered to  $1.0e-8$ , and the approximation for  $erfc(a)$  was replaced by a call to the NAG library routine S15ADF.

To permit easy exploration of differences, the same U235 case was run using both versions of the code. The sequence of modules was : MODER, RECONR, BROADR, UNRESR, THERMR and GROUPE. The case was one used in production running, to produce a 1968 group library for Fast Reactor cell

calculations. Eight sigma-p values and seven temperatures were specified. The results were that the standard case took 8,469 seconds (on a SUN Microsystems SPARCstation 2 processor) for BROADR, while the modified code took 9,050 seconds, or 6.8% longer. The standard BROADR generated 73,294 points on the energy grid, while the modified code generated 73,272. A colleague supplied a program for comparing two GENDF files. Use of this demonstrated that there were no differences in the group cross-sections, to within a tolerance of 0.001.

The process was repeated for the longer U238 case. The BROADR time rose from 22,157 seconds to 22,827, an increase of 3.0%. The number of energy grid points rose from 180,798 to 180,804.

The conclusion from this exercise is that altering NJOY to use the NAG subroutine library 16 sig. fig. approximation to  $erfc(a)$ , together with a change in a tolerance in BROADR from 1.0e-5 to 1.0e-8, has little effect upon running time and none upon the cross-sections generated. These alterations remove problems with broadening cross-sections at very high temperatures.

## REFERENCES

- 1 M. Abramowitz and I.A. Stegun  
Handbook of Mathematical Functions  
Dover Publications, New York, 1965.
- 2 W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling  
Numerical Recipes - The Art of Scientific Computing.  
Cambridge University Press, 1986.
- 3 The NAG Fortran Library, Mark 15.  
The Numerical Algorithms Group Ltd., 1991.

JEF2.2 B10(N,alpha) at 5keV

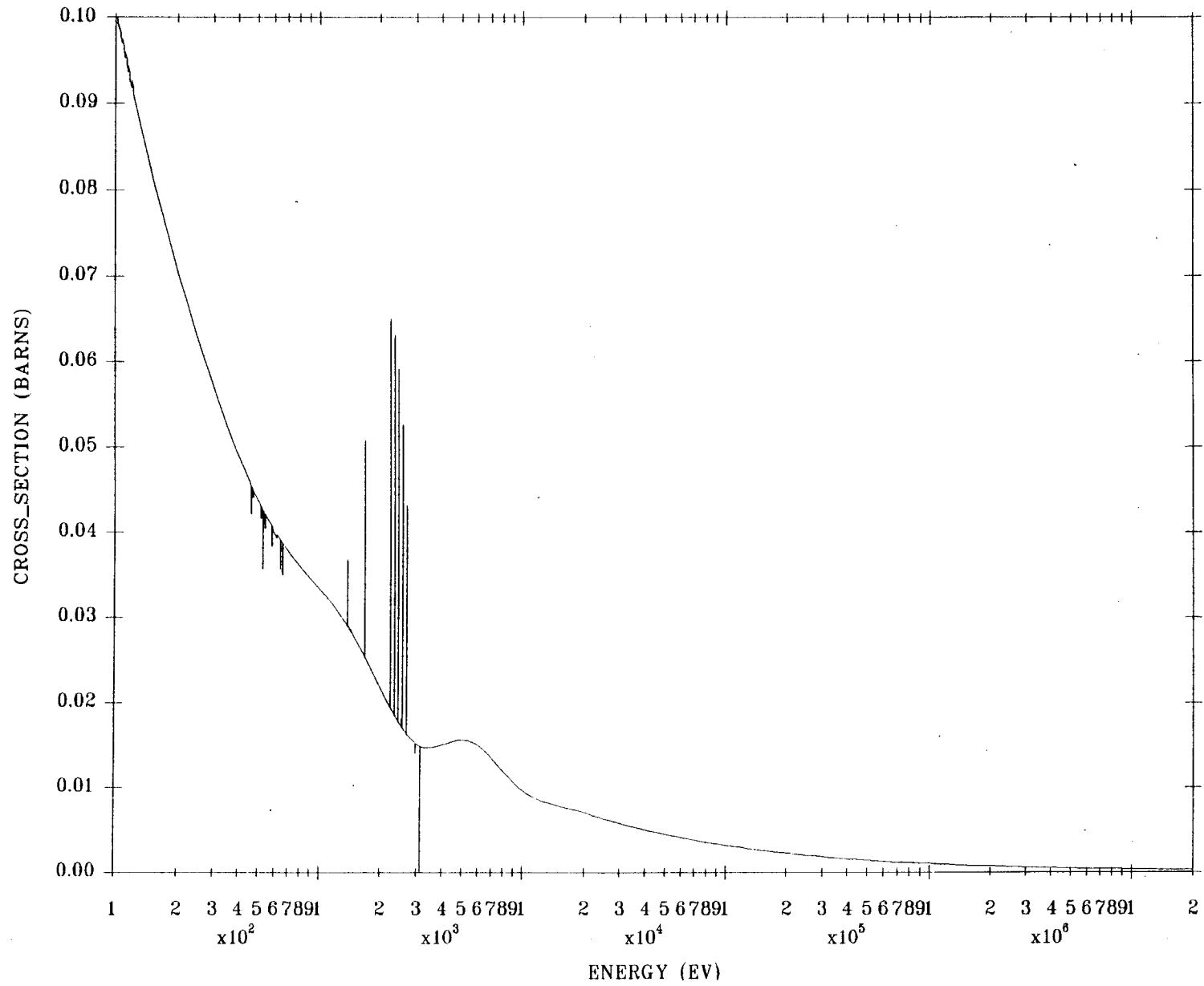


Figure 1. B10(n, $\alpha$ ) cross-section, using standard Winfrith NJOY89.62W

JEF2.2 B10(N,alpha) at 15keV

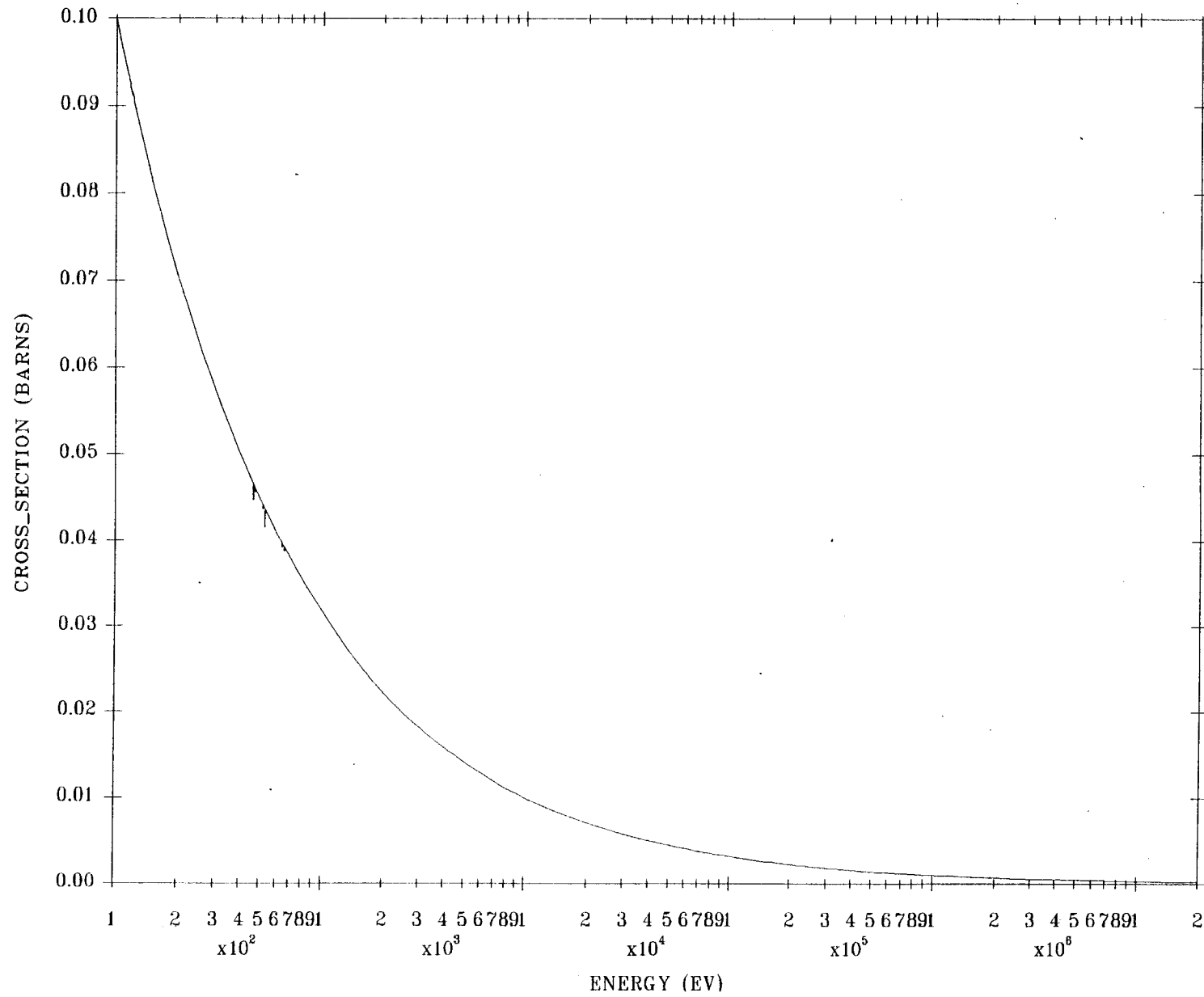


Figure 2. B10(n,α) cross-section, using Winfrith NJOY8.62W with "NUMERICAL RECIPES" erfc approximation

JEF2.2 B10(N,alpha) at 15keV

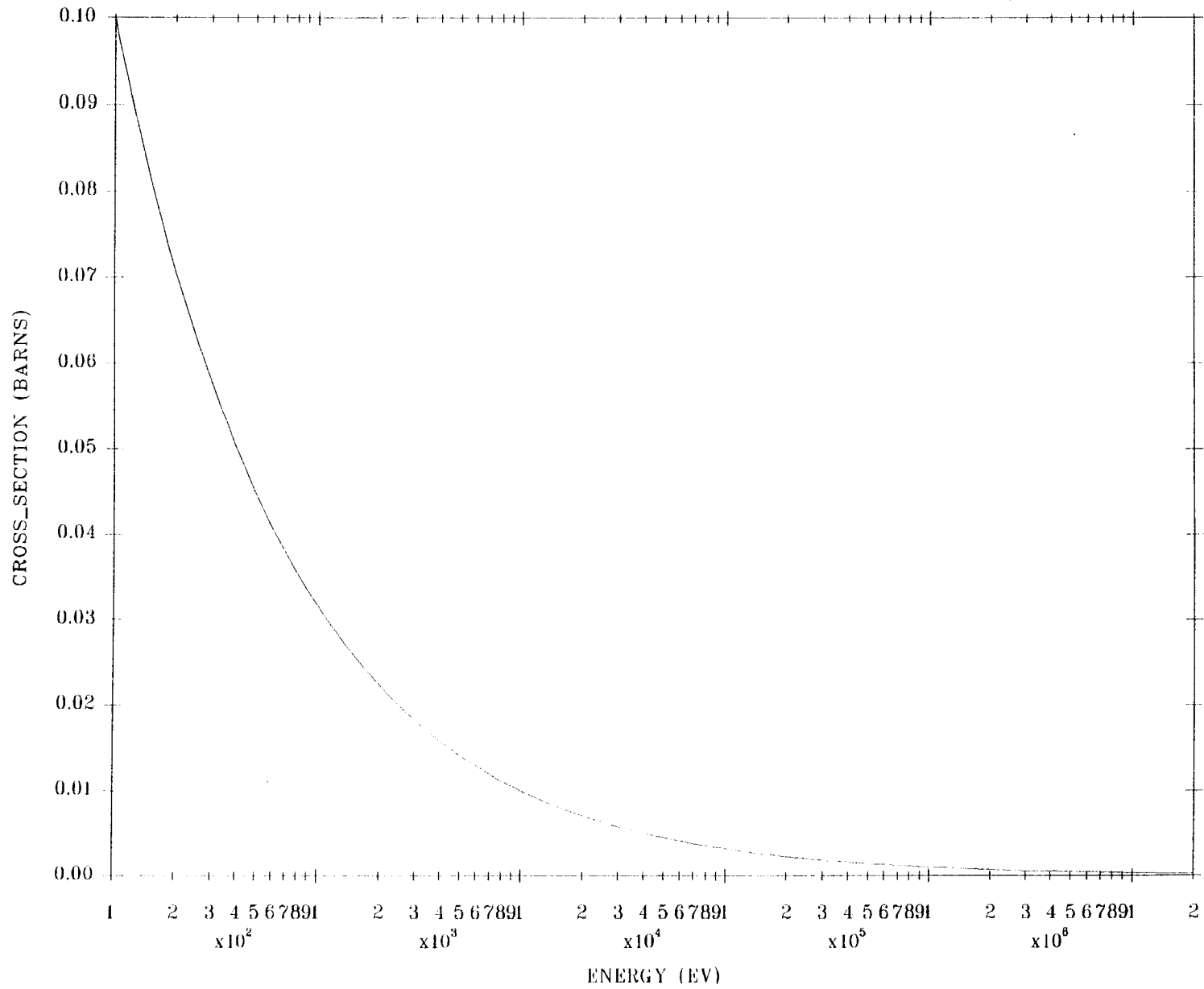


Figure 3. B10(n,α) cross-section, using Winfrith NJOY8.62W with "NAG16" erfc approximation

**Figure 4. PROGRAM TO TIME EXECUTION OF ERFC(A) APPROXIMATIONS**

```

PROGRAM junk
C-----
C|      ..      |
C|      .. Local Scalars .. |
C-----
      DOUBLE PRECISION a0,a1,a2,a3,a4,a5,expo,r,t,t1,t2,t3
      REAL t4,y
      INTEGER i
C-----
C|      ..      |
C|      .. Local Arrays .. |
C-----
      DOUBLE PRECISION er1(20000),er2(20000),er3(20000),x(20000),
&      xfunky(20000),xnagl6(20000),xnag8(20000),xnum(20000)
      REAL times(2)
C-----
C|      ..      |
C|      .. External Functions .. |
C-----
      REAL dtime
      EXTERNAL dtime
C-----
C|      ..      |
C|      .. Intrinsic Functions .. |
C-----
      INTRINSIC abs,exp
C-----
C|      ..      |
C|      .. Data statements .. |
C-----
      DATA a0,a1,a2,a3,a4,a5/.3275911,.254829592,-.284496736,
&      1.421413741,-1.453152027,1.061405429/
C-----
C|      ..      |
C|      .. Executable Statements .. |
C|      set x values |
C-----
      DO 1 i = 1,20000
        x(i) = 0.001* (i-1)
1 CONTINUE
C-----
C|      nag 16E |
C-----
      t1 = dtime(times)
      DO 2 i = 1,20000
        t = 1.0D0 - 7.5D0/ (abs(x(i))+3.75D0)
        y = ((((((((((((((((((+3.328130055126039D-10*t-
&      5.718639670776992D-10)*t-4.066088879757269D-9)*t+
&      7.532536116142436D-9)*t+3.026547320064576D-8)*t-
&      7.043998994397452D-8)*t-1.322565715362025D-7)*t+
&      6.575825478226343D-7)*t+7.478317101785790D-7)*t-
&      6.182369348098529D-6)*t+3.584014089915968D-6)*t+
&      4.789838226695987D-5)*t-1.524627476123466D-4)*t-
&      2.553523453642242D-5)*t+1.802962431316418D-3)*t-
&      8.220621168415435D-3)*t + 2.414322397093253D-2
        y = (((((y*t-5.480232669380236D-2)*t+1.026043120322792D-1)*t-
&      1.635718955239687D-1)*t+2.260080669166197D-1)*t-
&      2.734219314954260D-1)*t + 1.455897212750385D-1
        xnagl6(i) = 0.5*exp(-x(i)*x(i))*y
2 CONTINUE
      t1 = dtime(times)

```

Figure 8. (contd.)

```

C-----
C|          |
C|      funky |
C-----
      DO 3 i = 1,20000
        expo = exp(-x(i)**2)
        r = 1./ (1.+a0*x(i))
        xfunky(i) = 0.5*expo*r* (a1+r* (a2+r* (a3+r* (a4+a5*r))))
3 CONTINUE
      t2 = dtime(times)
C-----
C|          |
C| numerical recipes |
C-----
      DO 4 i = 1,20000
        t = 1.0/ (1.0+0.5*x(i))
        xnum(i) = 0.5*t*exp(-x(i)*x(i))-1.26551223+
&      t* (1.00002368+t* (0.37409196+t* (0.09678418+
&      t* (-0.18628806+t* (0.27886807+t* (-1.13520398+
&      t* (1.48851587+t* (-0.32215223+t*0.17087277)))))))))
4 CONTINUE
      t3 = dtime(times)
C-----
C|          |
C|      nag 8E |
C-----
      DO 5 i = 1,20000
        t = 1.0D0 - 7.5D0/ (abs(x(i))+3.75D0)
        y = ((((((((((+3.1475326D-5)*t-1.3374589D-4)*t-
&      6.4127909D-5)*t+1.7866301D-3)*t-6.2316935D-3)*t+
&      2.4151396D-2)*t-3.4799165D-2)*t+1.0260225D-1)*t-
&      1.6357229D-1)*t+2.2600824D-1)*t-2.7342192D-1)*t +
&      1.4558972D-1
        xnag8(i) = 0.5*exp(-x(i)*x(i))*y
5 CONTINUE
      t4 = dtime(times)

      DO 6 i = 1,20000
        er1(i) = 1000000.0* (xnum(i)-xnag16(i))/xnag16(i)
        er2(i) = 1000000.0* (xfunky(i)-xnag16(i))/xnag16(i)
        er3(i) = 1000000.0* (xnag8(i)-xnag16(i))/xnag16(i)
6 CONTINUE
      WRITE (6,FMT='(4(a15,f6.4))') 'TIMES XNAG16 ',t1,' XNUM ',t3,
&      ' XFUNKY ',t2,' XNAG8 ',t4
      WRITE (6,FMT='(8a18)') 'X','XNAG16','XNUM','XFUNKY','XNAG8',
&      'PPM ERR XNUM','PCM ERR XFUNKY ','PCM ERR XNAG8'
      WRITE (6,FMT='(1p5d18.10,5x,f8.4,10x,f3.4,10x,f3.4)') (x(i),
&      xnag16(i),xnum(i),xfunky(i),xnag8(i),er1(i),er2(i),er3(i),i=1,
&      20000)
      END

```

1e6 x ERROR FROM NJOY89, "NUMERICAL RECIPES", NAG8 cf NAG16

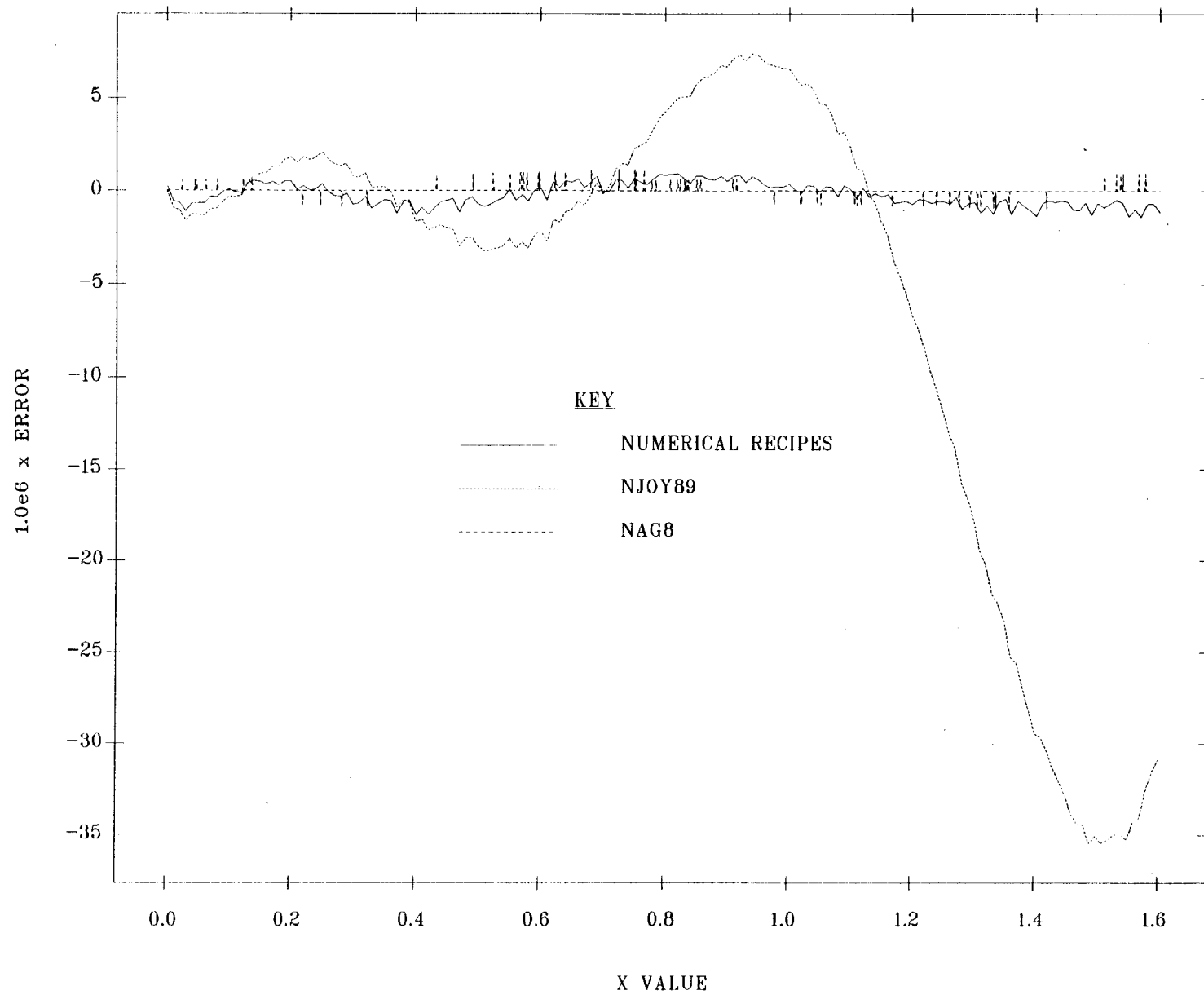


Figure 5. Errors (x 1.0e6) In using NJOY89, "Numerical RECIPES" and NAG8 approximations to erfc(x) cf NAG16

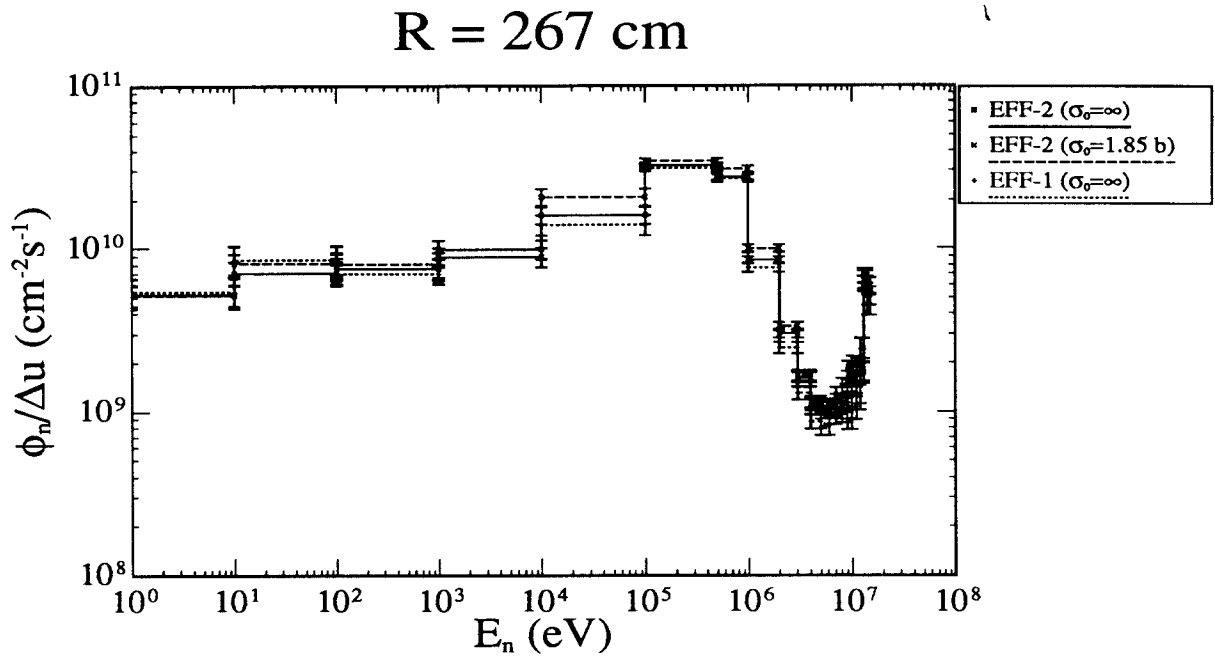


Figure 8: Neutron flux calculated by MCNP-4 in the NET541 geometry at  $R = 267 \text{ cm}$ . The results are given for EFF-2  $^{56}\text{Fe}$  data, self-shielded EFF-2  $^{56}\text{Fe}$  data, and EFF-1.3 Fe data.

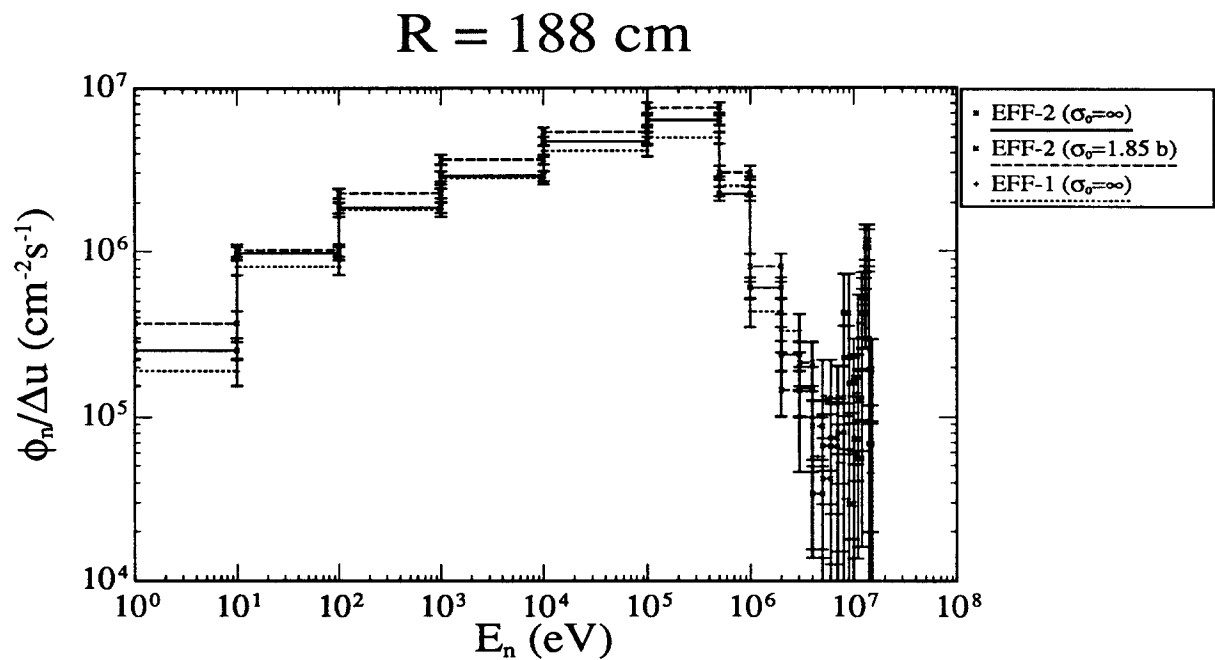


Figure 9: Neutron flux calculated by MCNP-4 in the NET541 geometry at  $R = 188 \text{ cm}$ . The results are given for EFF-2  $^{56}\text{Fe}$  data, self-shielded EFF-2  $^{56}\text{Fe}$  data, and EFF-1.3 Fe data.