ENDF-373

INTERNATIONAL ATOMIC ENERGY AGENCY

NUCLEAR DATA SERVICES

DOCUMENTATION SERIES OF THE IAEA NUCLEAR DATA SECTION

IAEA-NDS-217
April 2014

PROGRAM ENDF2C: CONVERT ENDF DATA TO STANDARD FORTRAN, C AND C++ FORMAT (VERSION 2014-1)

Dermott E. Cullen

National Nuclear Data Center, Brookhaven National Laboratory (alumnus)
Nuclear Data Section, IAEA, Vienna (alumnus)
University of California (retired)

1466 Hudson Way, Livermore, CA 94550, Tel. +1 925 443 1911
E-mail: redcullen1@comcast.net Website: http://home.comcast.net/~redcullen1

Abstract: The ENDF2C code is designed to ensure that ENDF evaluated data is in a standard, high-precision format compatible for direct use by FORTRAN, C and C++ codes. Here it is IMPORTANT to understand that ENDF2C cannot perform miracles by making data more precise, it can only preserve the accuracy of the data that it reads, but it can always ensure it is in the standard, approved format. Although this code and report only address data in the ENDF format, it is extremely important to understand that the problems addressed here apply to current nuclear data in ANY FORMAT: ENDF, ACER, GND, etc.
ENDF2C is designed to convert ENDF data to standard FORTRAN, C and C++ format. This code is designed for:
1) ENDF data in any ENDF format = ENDF-1 through ENDF-6.
2) On any type of computer = 32- or 64-bit system/compiler

This code tries to keep things as simple as possible
1) There are NO INPUT PARAMETERS.
2) It reads an ENDF formatted file named ENDF2C.IN
3) It writes an ENDF formatted file named ENDF2C.OUT
4) It writes a report file named ENDF2C.LST

The code is designed to be easily used on any computer - not only today's computers, but also anything that comes along in the future. So you can be assured that once you start using ENDF2C to produce ENDF data in standard format your compatibility problems are over - not just today, but well into the future.

The report is available online on https://www-nds.iaea.org/publications/nds/iaea-nds-0217/

Nuclear Data Section
International Atomic Energy Agency
P.O. Box 100
A-1400 Vienna
Austria

E-mail: NDS.Contact-Point@iaea.org
Fax: (43-1)26007
Telephone: (43-1)2600-21710
Web: http://www-nds.iaea.org

Disclaimer

TABLE OF CONTENTS

Introduction 1
Author's Message 1
Method 1
Running Time 2
Documentation 2
What Computer Can You USE? 2
Reading and Interpreting Data 2
Acknowledgments 3
Accuracy and Precision 4
Data Format 4
ENDF Character and Binary Formats 5
Mea Culpa 5
Caveat Emptor 6
Today’s Solution 7
Egocentric 7
7-digit versus 9-digit Precision 8
Cross Sections: 9 digits versus 7 digits 15
Closing Remarks and Recommendations 17
Conclusion 19
References 20

Introduction

The ENDF2C code is designed to ensure that evaluated data in ENDF format [1] are compatible for direct use in FORTRAN, C and C++ codes. It also ensures that the data include the highest possible precision (9 or 10 digits); here it is IMPORTANT to understand that ENDF2C cannot perform miracles by making data more precise, it can only preserve the accuracy of the data that it reads.

ENDF2C is designed to convert ENDF data to standard FORTRAN, C and C++ format. This code is designed for:
1) ENDF data in any ENDF format = ENDF-1 through ENDF-6.
2) On any type of computer = 32- or 64-bit system/compiler

This code tries to keep things as simple as possible:
1) There are NO INPUT PARAMETERS.
2) It reads an ENDF formatted file named ENDF2C.IN
3) It writes an ENDF formatted file named ENDF2C.OUT
4) It writes a report file named ENDF2C.LST

The code is designed to be easily used on any computer - not only today's computers, but also anything that comes along in the future. So you can be assured that once you start using ENDF2C to produce ENDF data in standard format your compatibility problems are over - not just today, but well into the future.

Author's Message

I consider ensuring that ENDF data are in a standard, officially approved format for FORTRAN, C and C++ is SO IMPORTANT that this code does only one thing - and only one thing - and it does it in the simplest possible manner - efficiency is NOT a consideration - ONLY accuracy and general utility of the ENDF data is considered.

Method

Other codes that attempt to do the same thing - including codes written by me decades ago - are very complicated, and therefore ERROR PRONE because they try to deal with each and every variant in which data can be coded in the ENDF format. Needless to say this means that every time the ENDF formats and procedures change these codes MUST also be changed.

In contrast, ENDF2C uses my almost 50 years of experience dealing with the ENDF format to realize that except for the comments at the beginning for each evaluation (MF/MT=1/451), every line of ENDF data is IDENTICAL - in every version of the ENDF format, from the original ENDF to today's ENDF-6. So to translate ENDF data into an official format I do not have to consider differences in each section (MF/MT) of ENDF data.

Every line of ENDF is divided into 6 fields, each 11 columns wide. Each of the 6 fields is either, blank, integer or floating point. Floating point fields ALL include a decimal point (.). So that ALL this code does is convert every floating point field to standard format.

In order to ensure that this PRESERVES the accuracy of the data, this is done by reading and writing each ENDF line as characters. Blank and integer fields are copied exactly as read. ALL floating point numbers that are read are converted internally from character to floating point - they are then converted back into characters in a standard, officially-approved format, for output.

As a last step to ensure the accuracy of results the characters to be output are again converted from characters to floating point, and the numerical value that is output is compared to the numerical value originally read. If there is ANY DIFFERENCE the characters strings read and written are listed in the output; the characters strings read and written as well as the difference is listed in the output report (ENDF2C.LST) and on the screen.

Running Time

It takes only seconds to translate an ENDF formatted evaluation, so running time need not be a consideration. Concentrate on keeping it simple and reliable - that should be your focus.

Documentation

ALL of my codes that process ENDF data and change it in ANY WAY document what they have done by adding comment lines at the end of the comment section (MF/MT=1/451) of each evaluation. This allows data users to determine the pedigree of the data they are using, by reading these comments. This code documents what it has done by adding the following 2 comment lines.

```
***************** Program ENDF2C (Version 2014-1) ***************
      Convert ENDF Data to Standard FORTRAN, C and C++ Format
```

WARNING - This documentation is IMPORTANT to data users and it should not be deleted. For example, I highly recommend that when you receive ENDF data from any source take the time to look near the end of the comment section (MF/MT=1/451) to see if this documentation is present. If not, PROTECT YOURSELF from problems using the data by FIRST running it through this code.

What Computer Can You USE?

This code has been kept as simple as possible, so that it can be used on ANY computer, and ANY computer system, e.g., 32-bit or 64-bit. The running time to convert an ENDF evaluation to Standard format is trivial, so that optimization or any other attempt to make it run faster is needless; just keep it simple, compile and execute as simply as possible, What is important here is accuracy of the results and general utility on any computer/system.

Reading and Interpreting Data

All data is read by this code in character form and internally translated into integer or floating point form as needed. This means:

1) It's difficult to get the code to crash by improperly defining input.

2) Your input can be in quite general form and will still be properly interpreted by the code, e.g., 14, 1.4+1, 14E+00, 1.4D+01, are all 14 as far as this code is concerned.

3) The code can distinguish between **BLANK** and **ZERO** input - **WARNING** - when this

documentation says **BLANK**, it means **BLANK** - in this case nothing else except a BLANK will be interpreted correctly as input.

Acknowledgments

I thank **Morgan White** (LANL) for initially pointing out that "E less" formatted ENDF data is not a FORTRAN standard, and it cannot be directly read by C and C ++ codes. I thank **Andrej Trkov** (NDS, IAEA, Vienna) for providing the STANEF [2] code used in his study. I thank **Bob MacFarlane** (LANL) for providing NJOY [4] results, **Doro Wiarda** (ORNL) for providing AMPX [5] results, and **Goran Arbanas** (ORNL) for providing SAMRML [6] results. I thank **Arjan Koning** (NRG, Petten) for testing the code and supplying feedback that significantly improved the code. I thank **Maurice Greene** (ORNL) for reviewing/correcting the report; converting gibberish into English.

Accuracy and Precision

Before discussing accuracy and precision I should first define what I mean by these terms. By **ACCURACY** I mean how uncertain is the nuclear data is that we use, and by PRECISION I mean how many digits we use to represent our nuclear data. To understand the accuracy of our nuclear data, let me briefly discuss how we obtain and use it. I like to divide the process we use in order to get final answers from our application codes into four steps:

Measurement – We still cannot define nuclear data based strictly on theory, so we measure important nuclear data; that is, data that we can afford to measure.

Evaluation – Here I include nuclear models – we cannot afford to measure everything we need, so the available measured data, nuclear model predictions, and yes, even best-guesses are combined and used to completely define what we need – this is more of an art than a science.

Processing – There is a lot of work to do between having nuclear data in the ENDF format and the form we need it in for use in our application codes. Historically this step has been ignored or taken for granted. But it is very important if we are to preserve the accuracy of the evaluated data that is actually used in our applications.

Applications – Finally we come to "the proof of the pudding" where we use the nuclear data in our neutron transport codes [8, 9], to predict results for our systems of interest.

The results of our applications compared to measured results can be used in an attempt to "close the loop" by allowing us to change our evaluated data so that our calculated results better agree with measured results. But care must be used here to ensure that the changes in the evaluated data are really improved physics, and not just a "fit" to accommodate the uncertainties in our nuclear data, processing and application codes. In order to accomplish this we must make every effort to control the accuracy of the nuclear data that we produce and use.

My general rule of thumb is that we do not know any cross section in any material at any temperature and neutron incident energy to better than roughly 1%; we may know some integral parameters more accurately, but not energy dependent cross sections. Each of the above steps adds its own uncertainty to the nuclear data, and these combine to define the overall uncertainty, or accuracy, of the data we end up using in our applications.

So PLEASE understand that in this paper when I state that our processing code now attempt to achieve a target uncertainty of 0.1%, this does not mean that we claim to know the cross sections to within 0.1%;. **This 0.1% is the ADDITIONAL UNCERTAINTY introduced in processing the evaluated data.** In order to define the total uncertainty this MUST be added to the inherent uncertainty in the evaluated data; again, I assume at least 1%. If the uncertainties in these four steps are uncorrelated they will combine quadratically: $[E1^2+E2^2+E3^2+E4^2]^{1/2}$, and our objective is to ensure that the uncertainty introduced by processing is small compared to the uncertainty in the evaluation; our target 0.1% uncertainty due to processing meets this criteria.

Data Format

In this report I will be discussing data in the ENDF format [1]. But be aware that the conclusions I present here apply to nuclear data in any format. My conclusion is that today's nuclear data requires at least 9 digits of precision. It is IMPERATIVE that this precision be maintained all the way up until it is actually used in our applications. Currently some of our application-oriented formats, such as ACER for MCNP [9], and the extended ENDL format I use with TART [8], already include the required precision. So, the focus here is on ensuring that our processing codes produce results with enough precision to preserve the accuracy of the data we actually use in our applications.

ENDF Character and Binary Formats

In the original documentation for ENDF/B, ENDF-102, two separate formats were described in detail: one using 80 characters per record, corresponding to the 80-column width of a computer card, and a more compact binary format. The intent was that the COMPUTER and LANGUAGE **INDEPENDENT CHARACTER** FORMAT be used to exchange nuclear data between users. It was assumed that each user would convert the character data to the COMPUTER and LANGUAGE **DEPENDENT BINARY** FORMAT to used in-house

Somewhere along the way the details for the binary format have been lost from the ENDF/B documentation, ENDF-102. Indeed today when I search the entire ENDF102 report the word “binary” is only mentioned four times, in reference to a binary format, but there is no longer any detailed description of the ENDF/B binary format.

Personally I think this is a BIG loss, because if one was using the ENDF/B binary format the question of precision and standard formats do not arise, since the data is stored to the full precision of each computer without any truncation. In contrast when using the ENDF/B character format, precision and standard formats become extremely important. Indeed, if no one was using the ENDF character format there would be no reason for this document.

Mea Culpa

First I will include here a brief history lesson on ENDF data formats. In particular I will take credit, or blame, for the non-standard format in which ENDF data has been distributed for over 40 years.

When I began work in 1967 at the newly created National Nuclear Data Center (NNDC), Brookhaven National Lab, the ENDF system was in its infancy. Prior to ENDF each Laboratory or facility in the United States had its own computer system for handling nuclear data for use with its application codes. Generally each of these systems was designed only for in-house use, and in most cases they were incompatible. This incompatibility prevented exchange of data between users as well as comparison of the results of application codes for data testing.

ENDF was an attempt to standardize the computer formats used for nuclear data throughout the United States to allow the exchange of nuclear data between producers and users of nuclear data; I should mention that much of ENDF was designed based on the earlier British UKNDF system.

At the time virtually all data for use on computers originated on 80-column computer cards, so that ENDF was naturally based on an 80-column format. Although we haven’t used computer cards in decades, the ENDF format turned out to be so successful and was so strongly supported by generally available computer codes, it is still used today.

From the beginning we could see that if it succeeded ENDF would be a big step forward for nuclear data. But from the very beginning we ran into a precision problem. ENDF was, and still is, based on data fields 11 columns wide, either fixed or floating point. This meant that in STANDARD floating point FORTRAN output in E11.4 format included a precision of on some computers only four (0.1234E+05), or on other computers five (1.2345E+04) digits. This might seem like more than enough accuracy for cross sections that we might know to an accuracy of 1%, but we quickly ran into what sounds like a trivial problem to uniquely define the target nucleus, in ENDF terminology ZA (1000*Z+ A, combining the atomic number and weight). For example, U238 is identified as 92238. Unfortunately when this was written in STANDARD E11.4 format on some computers this was 9.2238E+04, but on other computers it became 0.9224E+05 = 92240, rather than the intended 92238 – which was DISASTEROUS!!!! Trying to use 1pE11.4 format to force an extra digit of accuracy did not solve the problem, because on some computers it caused negative numbers to overflow of the 11 column format ENDF format, and produced *********** as output.

For ENDF, I avoided this problem by using what we today call "E less" floating point numbers. Specifically, I began writing ALL floating point number without "E" preceding the exponent, e.g., " 1.23456+04". Before adopting this format for use in ENDF I first tested to be sure that this worked on every computer/system that I could try at the time; here by "work" I meant that any ENDF data user could accurately read this data using nothing more than a standard FORTRAN E11.4 format. Only after assuring that this could generally be used did I adopt it for use in ENDF. Recently I was amused to look at the ENDF/B-II data (circa 1970), which I noted already included "E less" formatted data; this provided six (6) digits of accuracy, and at the time solved our problem of more precisely defining ENDF data. Later we realized that almost all ENDF data only requires a single digit exponent, so we extended our precision to produce what we today refer to as "E less", 7-digit format, e.g., " 1.234567+4".

I will conclude this section by taking credit for my inventing the "E less" format that has served us well over the years, and decades. But I will also take the blame (Mea Culpa) for the problems that it can cause today, as I describe below.

Caveat Emptor

Until fairly recently this "E less", 7-digit format has met our needs. I used it in my codes from roughly 1970 to 2000, or about thirty years. However, I always felt a little uneasy about this "E less" format. Over the years I could never find any book or publication on FORTRAN that mentioned this "E less" format. So I finally realized that it has always been non-standard, and we were just lucky that FORTRAN compilers were not strict enough to cause us problems. My worry was that we have no way of knowing whether or not our luck will continue into the future, and can we afford to take the chance that it won't? I think not.

By 2000 I realized that although we could use this "E less" format with FORTRAN, we had a MAJOR problem when it came to C and C++; here data in this format is not only DEFINITELY NOT COMPATIBLE it is also DANGEROUS. As I had done many years ago when I tested my "E less" format for use with FORTRAN, by 2000 I did similar tests for C and C ++. My results proved to me that I had to abandon my "E-less" format. What I discovered by testing a variety of C and C++ compilers or several computers was that, depending on the computer I tried:

1) 1.234567+4 was interpreted as 1.234567 (exponent ignored)
2) 1.234567+4 was interpreted as 5.234567 (exponent added)
3) 1.234567+4 was interpreted as 1.234567 followed by another field equal to 4
4) I couldn't find any computer where this was interpreted corrected
5) Worst of ALL, none of the computers gave any ERROR message = **SCARY!!!**

To me these results proved I had to abandon my invention, "E less" output, and devise a new and better solution to ensure that values are always read consistently on all computers and in all languages. "E less" output has served us well over the years for use in ENDF, but by today as with all things it has outlived its usefulness. Indeed my tests show that to continue to use it is DANGEROUS; so I will conclude this section by saying that as far as "E less" data is concerned **Caveat EMPTOR = Let the User BEWARE!!!**

Today's Solution

When I realized the problem with the "E less" format, it wasn't very hard to come up with a solution; a solution that not only avoids the "E less" format problem, but also extends the precision from seven (7) digits to nine (9) or even ten (10) digits when needed = a win-win situation = compatible AND more precise.

My solution is a true example of "Columbus and the egg". You may ask: "Why haven't we always used this? But I should mention that it took me over thirty years of using ENDF/B to think of this "Columbus and the egg" solution.

Most data in ENDF units; eV, barns, are in the range .1 to 9,999,999,999. Any number in this range can be output to the ENDF format in F, rather than E, format, to,

" 12345.6789" = 9 digits, with a leading blank or sign, or,
"12345.67895" = 10 digits, if needed, and the number is not negative.

That's it = that's my entire solution. There are a few nitty-gritty details concerning what to do with number that are less than .1 or 10,000,000,000 or more, but suffice it to say that these number can all be written to at least seven digits of accuracy – so they are always at least as accurate as the "E less" formatted output.

In a nutshell, there is no disadvantage to using this extended precision, standard FORTRAN, C and C ++ format, and there are obvious advantages; i.e., if this isn't clear to you, see the preceding sections. If it still isn't clear read the following sections regarding the precision required today to accurately represent today's evaluated data; here I show that **it is physically impossible to accurately represent all of today's evaluated data using only 7-digit format – let me repeat that: it is PHYSICALLY IMPOSSIBLE!!!!**

Egocentric

One thing that really surprised me when I initially told ENDF evaluators and code developers about this problem and its solution was the **egocentric** response from some people. Specifically, to me it was obvious that we had to fix this problem at the source, by updating our data and codes, so that ENDF users throughout the world could easily and reliably use the data. So I was shocked to be told by some evaluators and code developers that rather than fix one evaluation or code, at the source, that they wanted everyone throughout the world to fix their codes to correctly handle this non-standard, dangerous "E less" format – if that isn't egocentric I don't know what is = I couldn't believe this strictly "me, me, me" egocentric view. Fortunately and eventually, I was able to convince most people to correct their codes and data at the source.

7-digit versus 9-digit Precision

So far I have only discussed the problem of compatibility, whereas there is an additional problem with regard to **the precision of ENDF data.** When ENDF started, evaluated neutron data was in a much more primitive state than it is today, and my extended 7-digit format was more than adequate to accurately define the data in the ENDF format. But today's evaluations, particularly the resolved resonance region, extend to much higher energies, and to accurately represent the energy dependent cross sections requires much more precision than is physically possible using the 7-digit format. For example, in ENDF/B-II (circa 1970) the resolved resonance region of U235 only extended up to 64 eV. In comparison, today we have new evaluations, such as Fe56, where the resolved energy range extends into the MeV range.

To more easily understand the problem, consider one example of a new evaluation. Let's start at the source of the data and take a look at the data distributed by the NNDC, Brookhaven. Before the ENDF data is distributed by NNDC it is first translated into a "standard" format using the STANEF code; STANEF output is in the "E-less" 7-digit format. This immediately produces a problem at the source, before the data is even distributed. The reason is easiest to understand by showing an example. Here are some resonance parameters from a new evaluation; I should mention that ALL evaluations produced by Oak Ridge's SAMMY [6] code use this same format, so this is but one of many examples:

```
1813678.125 7.000000-1 1.422769+4 8.036028-1 3.605933-1           2631 2151   90
1829159.750 7.000000-1 4.769847+1 1.046022+2 4.237417+1           2631 2151   91
1893056.250 7.000000-1 1.546389+4 6.114362+0 3.616063+0           2631 2151   92
1953226.375 7.000000-1 3.182862+4 1.287646+0 7.627806-1           2631 2151   93
1978236.750 7.000000-1 1.271467+3 1.456289+1 9.715957+0           2631 2151   94
2001532.125 7.000000-1 1.410720+4 5.307203-1 6.316836-2           2631 2151   95
```

For comparison here is the output I get when I run the STANEF code:

```
 1.813678+6 7.000000-1 1.422769+4 8.036028-1 3.605933-1 0.000000+02631 2151   90
 1.829160+6 7.000000-1 4.769847+1 1.046022+2 4.237417+1 0.000000+02631 2151   91
 1.893056+6 7.000000-1 1.546389+4 6.114362+0 3.616063+0 0.000000+02631 2151   92
 1.953226+6 7.000000-1 3.182862+4 1.287646+0 7.627806-1 0.000000+02631 2151   93
 1.978237+6 7.000000-1 1.271467+3 1.456289+1 9.715957+0 0.000000+02631 2151   94
 2.001532+6 7.000000-1 1.410720+4 5.307203-1 6.316836-2 0.000000+02631 2151   95
```

And here is the output I get when I run my ENDF2C code:

```
1813678.125 .700000000 14227.6900 .803602800 .360593300           2631 2151   90
 1829159.75 .700000000 47.6984700 104.602200 42.3741700           2631 2151   91
 1893056.25 .700000000 15463.8900 6.11436200 3.61606300           2631 2151   92
1953226.375 .700000000 31828.6200 1.28764600 .762780600           2631 2151   93
 1978236.75 .700000000 1271.46700 14.5628900 9.71595700           2631 2151   94
2001532.125 .700000000 14107.2000 .530720300 .063168360           2631 2151   95
```

A problem arises because ORNL's SAMMY code, when it produces resonance parameters, outputs its results for resonance energies using 10-digit accuracy. STANEF then truncates these to 7 digits; in the above tables I have highlighted one resonance energy that should be identical in all three cases. The remaining SAMMY parameters are in the same form as STANEF output, with everything to 7 digits using the "E-less" ENDF format (e.g., 1.234567+3), so there is no truncation, but it is the incompatible "E-less" format that can potentially cause major and unnecessary problems.

My ENDF2C code avoids the "E-less" ENDF format and produces "standard" FORTRAN, C and C++ formatted data. In this case ENDF2C maintains the 10-digit accuracy of the resonance energies and the 7-digit accuracy of the remaining parameters, while still producing results in legal, standard FORTRAN, C and C++ output format, without using the "E-less" ENDF format.

Note that ENDF2C will only use 10-digit output if it is absolutely necessary; this is done in an attempt to make the resulting output as human readable as possible, by having a "blank" column between data fields. We can see this effect in the above table of ENDF2C output, e.g., 1829159.750 is output with a leading "blank" because the "0" at the end is not ABSOLUTELY necessary to define the precision of this number. Here the leading "blank" for a line isn't that important, but if you can imagine a string of 6 successive numbers on a line with no "blanks" between them = 66 digits – this is almost impossible for a human to correctly interpret. The result is ENDF21C output that looks like this, with "blanks" between the fields:

```
1813678.125 700.000000 14227.6900 803.602800 360.593300 123.456789
```

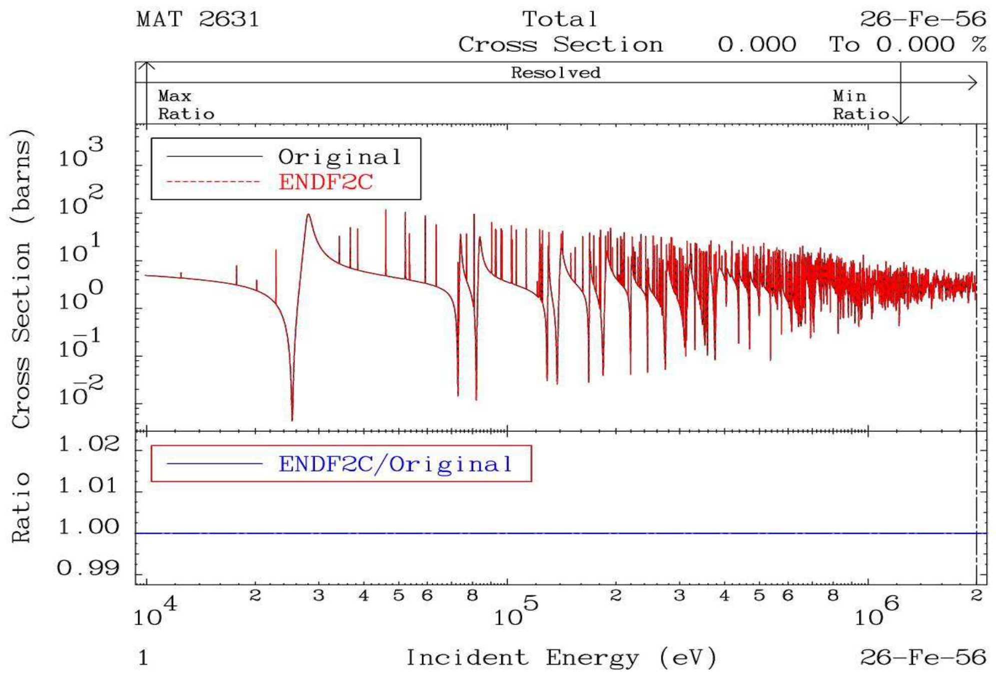
Rather than this, with no separation between fields,

```
1813678.125700.000000014227.69000803.6028000360.5933000123.4567890
```

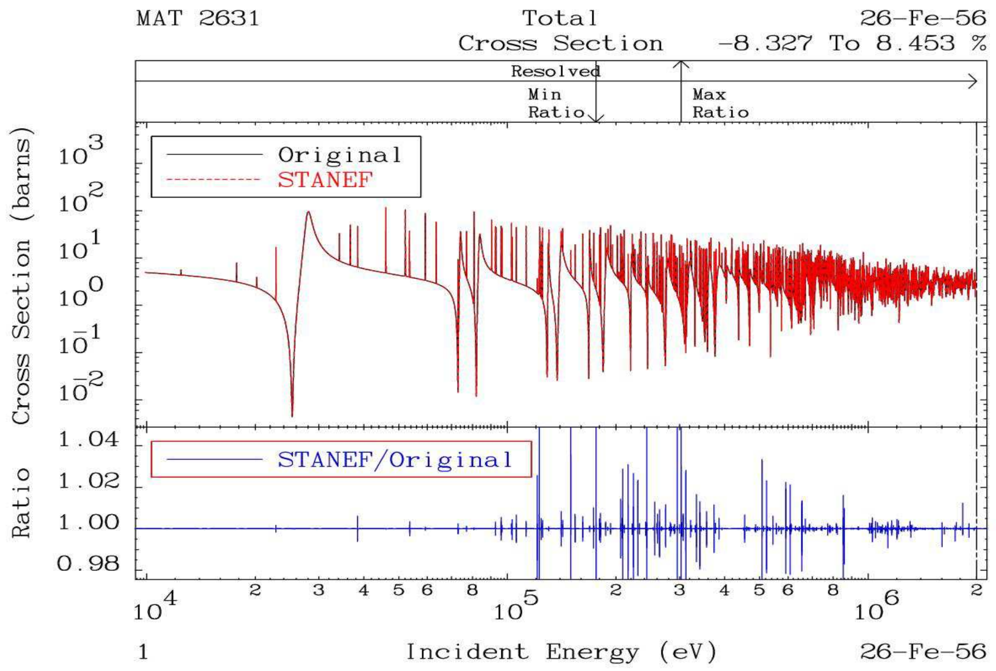
In both cases the six numbers are exactly numerically equivalent, but the 10 digit of 66 characters with no blanks between them is very difficult for humans to read, but computers have no problem reading either form as exactly numerically equivalent.

The following pages compare PREPRO [3] output using the ORIGINAL, ENDF2C , and STANEF results. The ORIGINAL and ENDF2C results agree. The STANEF results differ by up to more than 8%. Today these differences can be so easily avoided, while still maintaining the precision of the ORIGINAL evaluations as well as the standard format.

Comparison of ORIGINAL and ENDF2C results show EXACT agreement.



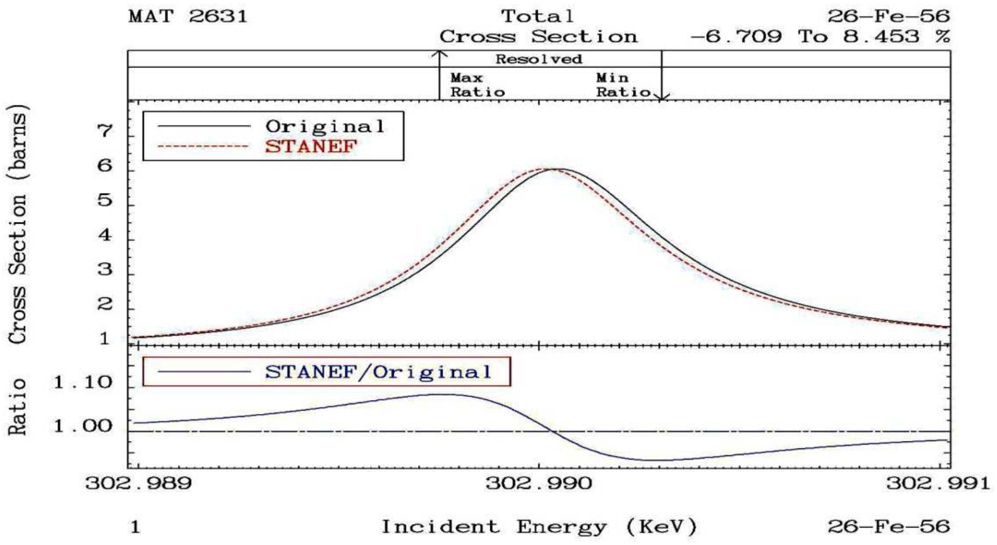
Comparison of ORIGINAL and STANEF results show over 8% differences.



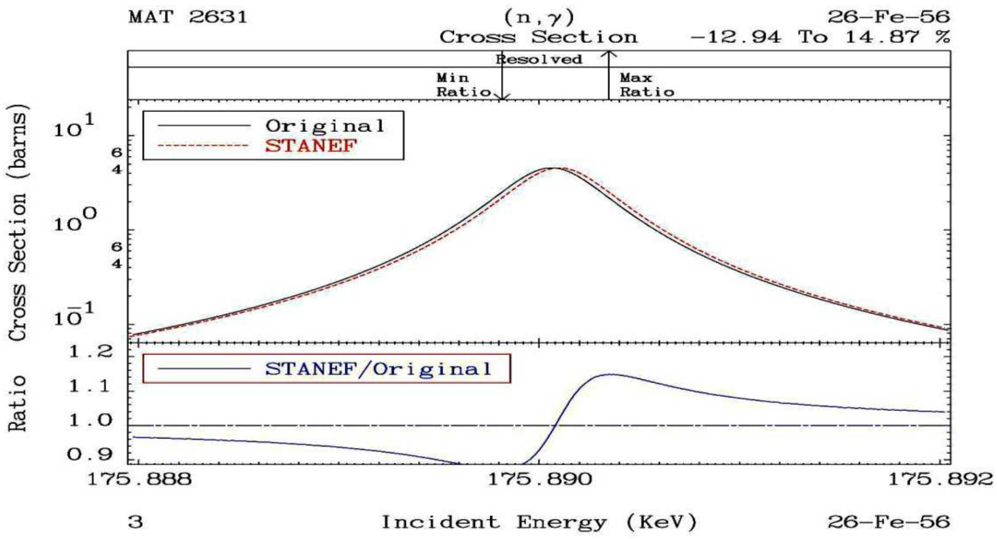
Are these differences important in our applications? Today we will never know, because the ORIGINAL evaluations are not distributed by NNDC; only the STANEF results are distributed. But that's not the point: the point is that today these differences are not necessary and can be so easily avoided.

In the above example, whether we start from the Original evaluation with 10-digit resonance energies, or the STANEF output where these have been truncated to 7-digits, when either representation is run through RECENT to reconstruct the energy dependent cross sections, the RECENT [3] results will be output to 9-digit precision. In this case, we expect the results to be small shifts in the resonance energies (7- vs. 10-digit precision), which are of little concern in any macroscopic or integral sense, as shown below using very small energy intervals where plots show large differences.

Total showing differences up to 8.4%

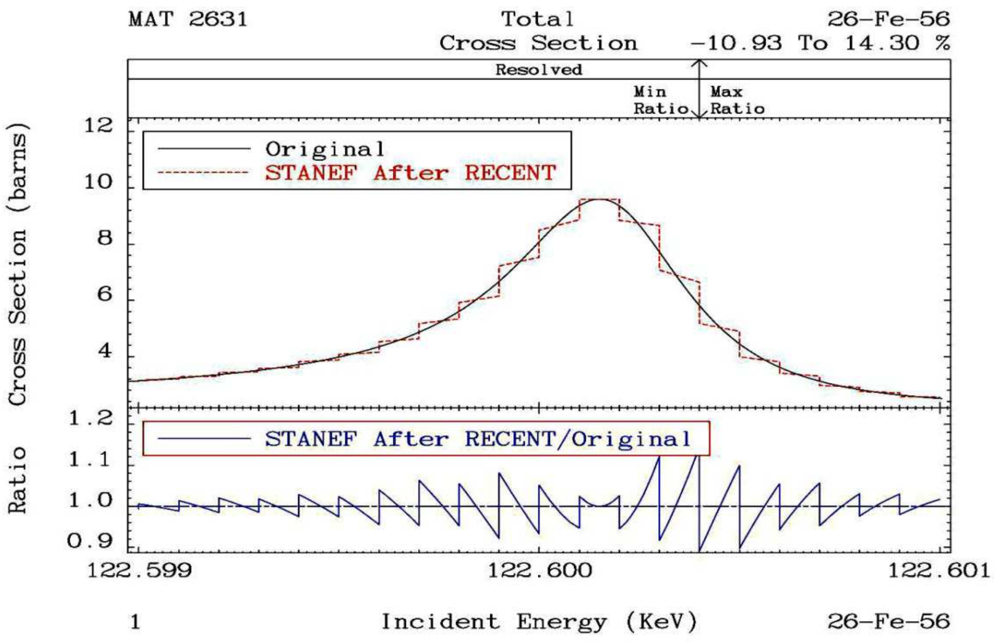


Capture showing differences up to 14.8%

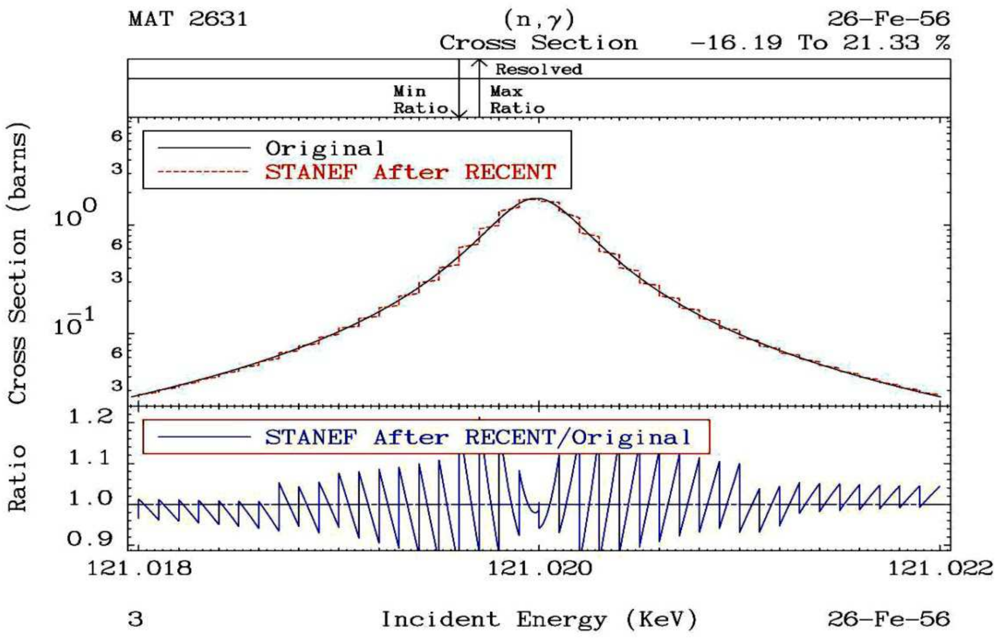


In this case we need not be overly concerned. But to see the real danger of using 7-digit output with today's modern evaluations let's see what happens when we run the 9-digit RECENT output through STANEF and truncate it to 7 digits.

Total showing differences up to 14.3%



Capture showing differences up to 21.3%



Now rather than the unimportant micro-shifts we saw above, **here we see that the 7-digit output is unable to accurately define the shape of these resonances – this is IMPORTANT to understand, so let me repeat: it is PHYSICALLY IMPOSSIBLE**. Rather than the smooth profile produced by the RECENT 9-digit output, the 7-digit STANEF output produces **Ziggurats = stepped pyramids**.

To understand the problem we need merely compare the 9-digit RECENT output near the peak of the resonance…

```
 122600.007 8.18440080 122600.018 8.34734256 122600.021 8.391426722631 3  1 6030
 122600.028 8.49343634 122600.055 8.86885730 122600.077 9.139853322631 3  1 6031
 122600.099 9.36221564 122600.120 9.51387393 122600.131 9.565109362631 3  1 6032
 122600.142 9.59483076 122600.153 9.60178112 122600.164 9.585085692631 3  1 6033
 122600.176 9.53939892 122600.178 9.52899291 122600.180 9.517793222631 3  1 6034
 122600.182 9.50580250 122600.184 9.49302393 122600.196 9.400050502631 3  1 6035
 122600.207 9.29106975 122600.218 9.16076388 122600.229 9.010880802631 3  1 6036
 122600.240 8.84348060 122600.251 8.66085453 122600.261 8.483667022631 3  1 6037
 122600.283 8.06587838 122600.305 7.62540465 122600.319 7.341331292631 3  1 6038
 122600.332 7.07896654 122600.354 6.64519685 122600.375 6.250366772631 3  1 6039
 122600.397 5.86278292 122600.419 5.50514627 122600.441 5.178659552631 3  1 6040
 122600.462 4.89577103 122600.484 4.62825344 122600.505 4.398676332631 3  1 6041
 122600.527 4.18301692 122600.549 3.99047205 122600.571 3.818722112631 3  1 6042
 122600.592 3.67214930 122600.614 3.53482178 122600.625 3.471802962631 3  1 6043
 122600.636 3.41225491 122600.658 3.30276311 122600.679 3.209061382631 3  1 6044
 122600.701 3.12093801 122600.722 3.04533219 122600.744 2.974035832631 3  1 6045
 122600.766 2.90992540 122600.788 2.85219006 122600.809 2.802371112631 3  1 6046
 122600.831 2.75512041 122600.853 2.71238188 122600.875 2.673669622631 3  1 6047
 122600.896 2.64007944 122600.918 2.60804847 122600.939 2.580185982631 3  1 6048
 122600.961 2.55355301 122600.983 2.52927697 122601.005 2.507124522631 3  1 6049
```

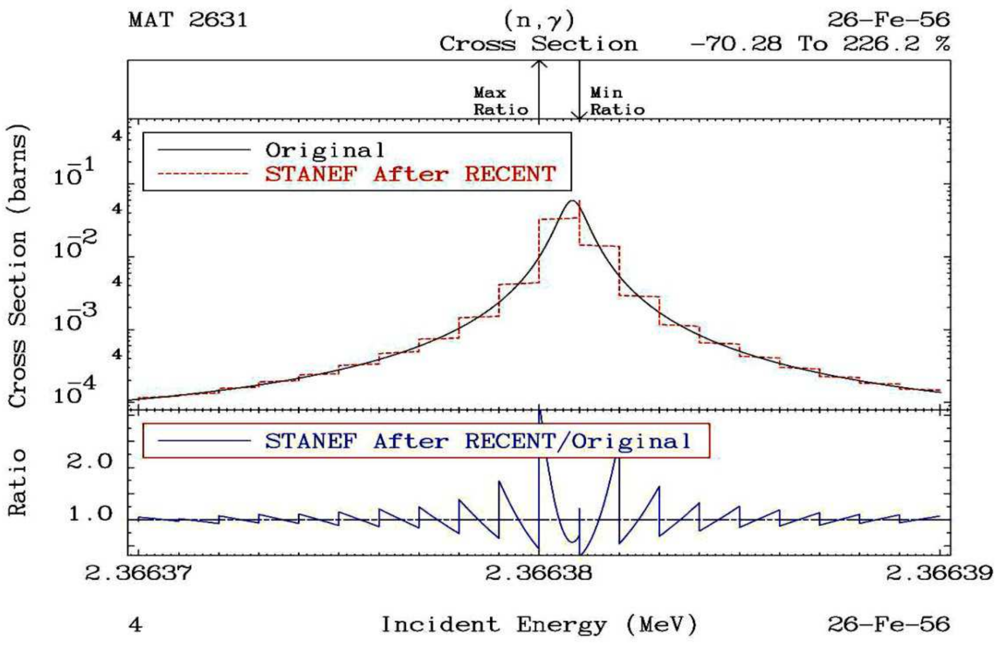
…to the STANEF 7-digit output over the same energy range:

```
 1.226000+5 8.184401+0 1.226000+5 8.347343+0 1.226000+5 8.391427+02631 3  1 4518
 1.226000+5 8.493436+0 1.226001+5 8.868857+0 1.226001+5 9.139853+02631 3  1 4519
 1.226001+5 9.362216+0 1.226001+5 9.513874+0 1.226001+5 9.565109+02631 3  1 4520
 1.226001+5 9.594831+0 1.226002+5 9.601781+0 1.226002+5 9.585086+02631 3  1 4521
 1.226002+5 9.539399+0 1.226002+5 9.528993+0 1.226002+5 9.517793+02631 3  1 4522
 1.226002+5 9.505802+0 1.226002+5 9.493024+0 1.226002+5 9.400051+02631 3  1 4523
 1.226002+5 9.291070+0 1.226002+5 9.160764+0 1.226002+5 9.010881+02631 3  1 4524
 1.226002+5 8.843481+0 1.226003+5 8.660855+0 1.226003+5 8.483667+02631 3  1 4525
 1.226003+5 8.065878+0 1.226003+5 7.625405+0 1.226003+5 7.341331+02631 3  1 4526
 1.226003+5 7.078967+0 1.226004+5 6.645197+0 1.226004+5 6.250367+02631 3  1 4527
 1.226004+5 5.862783+0 1.226004+5 5.505146+0 1.226004+5 5.178660+02631 3  1 4528
 1.226005+5 4.895771+0 1.226005+5 4.628253+0 1.226005+5 4.398676+02631 3  1 4529
 1.226005+5 4.183017+0 1.226005+5 3.990472+0 1.226006+5 3.818722+02631 3  1 4530
 1.226006+5 3.672149+0 1.226006+5 3.534822+0 1.226006+5 3.471803+02631 3  1 4531
 1.226006+5 3.412255+0 1.226007+5 3.302763+0 1.226007+5 3.209061+02631 3  1 4532
 1.226007+5 3.120938+0 1.226007+5 3.045332+0 1.226007+5 2.974036+02631 3  1 4533
 1.226008+5 2.909925+0 1.226008+5 2.852190+0 1.226008+5 2.802371+02631 3  1 4534
 1.226008+5 2.755120+0 1.226009+5 2.712382+0 1.226009+5 2.673670+02631 3  1 4535
 1.226009+5 2.640079+0 1.226009+5 2.608048+0 1.226009+5 2.580186+02631 3  1 4536
 1.226010+5 2.553553+0 1.226010+5 2.529277+0 1.226010+5 2.507125+02631 3  1 4537
```

Here the entire shape of the resonance is between 122.599 keV and 122.601 keV, and we can see that all of the above-tabulated points in both tables start with EXACTLY the same six-digit energy 122600. This means that with 9-digit RECENT output we only have three digits with which to define the entire shape of the resonance, which is adequate, but with 7-digit STANEF output we only have one digit!!!!, which is far from adequate. Compare what should be EXACTLY the same energy points that I have highlighted, and you will see smooth variation of the 9-digit RECENT energies, but **ALL OF THE HIGHLIGHTED 7-DIGIT STANEF ENERGIES ARE EXACTLY THE SAME VALUE, 1226600.2 eV**, which is what is causing the Ziggurats (stepped pyramids) that we see in the above figures = a constant X value (energy) and a range of Y values (cross sections), creating a vertical STEP in the above figure = nonsense, completely due to nothing but truncating to 7-digit energies.

This is more than an academic exercise. Until recently, except for PREPRO, all of our ENDF processing codes were using 7-digit output, and producing **EXACTLY THE SAME ZIGGURATS WE SEE IN THE ABOVE FIGURES.** Fortunately, recently as a results of code comparisons, today many of our codes use 9-digit output and produce energy-dependent results within our target allowable uncertainty (we all use 0.1%), the codes that today process properly include, in alphabetical order: AMPX, NJOY, PREPRO and SAMRML.

The bottom line here is to understand that due to the details included in modern evaluations **it is physically impossible for 7-digit output to accurately represent the energy dependent cross sections to anywhere near our target allowable uncertainty (0.1%)**. In this case we find differences in the total of over 14% and in capture over 21%; see the above plots = **140 to 210 times our target uncertainty of 0.1%**. Be aware that these are not isolated differences in a few resonances; we see these differences over the entire resolved resonance energy range, and this isn't even the worst case, e.g., the latest Fe56 evaluation includes resonances well up into the MeV range, an order of magnitude higher in energy than the resonances shown in the above figures. Below is but one example of a capture resonance in the MeV range **where truncating from 9 digits to 7 digits results in differences of up to 226% = over a factor of 2!!!!!**



Cross Sections: 9 digits versus 7 digits

Since our objective is to reproduce the cross sections to within 0.1%, one might assume we only need to know any cross sections to an accuracy of 3 or 4 digits. However, this does not take into account the fact that cross sections are often defined by sums or differences. For example, we often define our total as the sum of its parts. In the simplest case without fission and below the inelastic threshold we have:

Total = Elastic + Capture

You might then assume that obviously this means: Capture = Total - Elastic

But use caution. As just one example, the General Reich-Moore (LRF=7) ENDF treatment [1] does not directly define the capture cross section; it is defined by subtracting from other reactions. Similarly, in Monte Carlo calculations codes often select a reaction by subtracting cross sections, e.g., is it elastic? If not, subtract the elastic from the total and try the next reaction.

As a result, in a perfect world we could assume: Capture = Total - Elastic

But in the real world where cross sections can vary over many orders of magnitude and we must deal with the finite accuracy of computers, this seemingly simple relationship is difficult to actually achieve within our target accuracy of 0.1%.

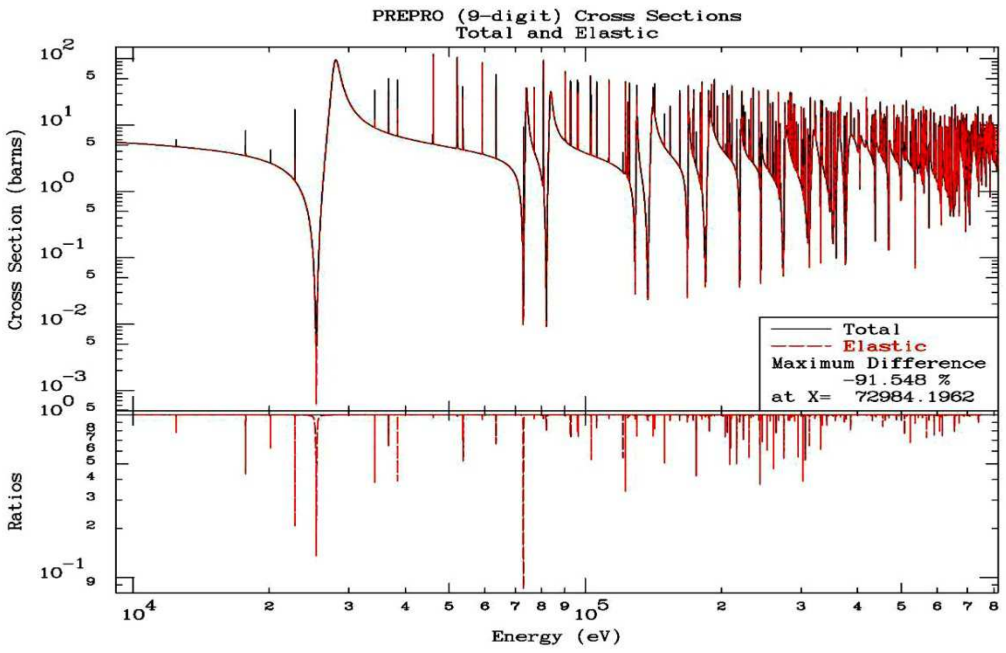
At 521082.982 eV

	9-digits	7-digits
Total	2.53264165	2.532642
Elastic	2.53259305	2.532593
Capture	0.0000485970	0.0000485970
Capture = Total - Capture	0.00004860	0.000049
Difference	-0.000000030	-0.0000004030
Per-Cent	-0.006%	-0.829%

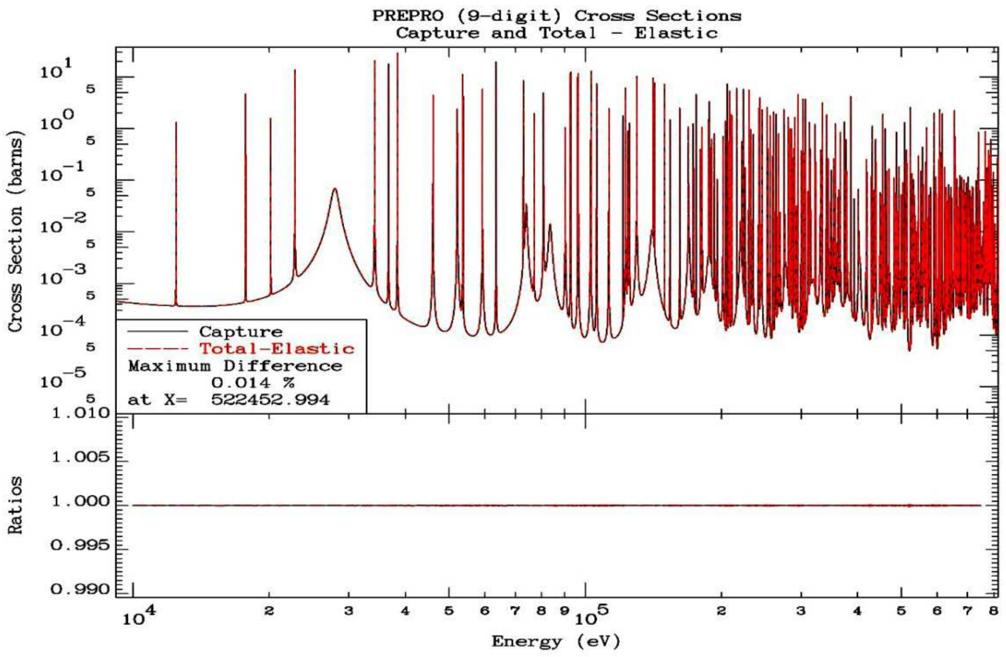
To illustrate the problem we need only consider one energy point in Fe-56 where the Total and Elastic agree to almost 5 digits (see the above table). This means if we try to define the Capture as the difference, with cross sections truncated to 9 digits we only have about 4 digits remaining to define the Capture. And with 7-digit accuracy we only have about 2 digits remaining (the digits shown as yellow in the table above are lost due to truncation). In this situation we can easily see (from the table above) that we can still achieve our 0.1% target allowable uncertainty with 9-digit cross sections, **but not with 7-digit cross sections**. In the table above, we can see that 9-digit cross section truncation results in 0.006% error (well within our 0.1% target), whereas 7-digits results in 0.829%, or roughly 100 times larger than the 9-digit results, which is exactly what we would expect when we lose 2-digits (9 versus 7). The following pages show results using cross sections truncated to 7 and 9-digit; these results include:

PREPRO 9-digit results below show first a comparison of the total and elastic, to illustrate that the two are equal over almost the entire energy range, except near some resonances where capture makes a significant contribution. Next, I compare the capture cross section tabulated in the evaluation to that defined as Total - Elastic. Here we see a maximum difference of about 0.01%, an order of magnitude below our target allowable uncertainty 0.1%; well within what we consider to be acceptable.

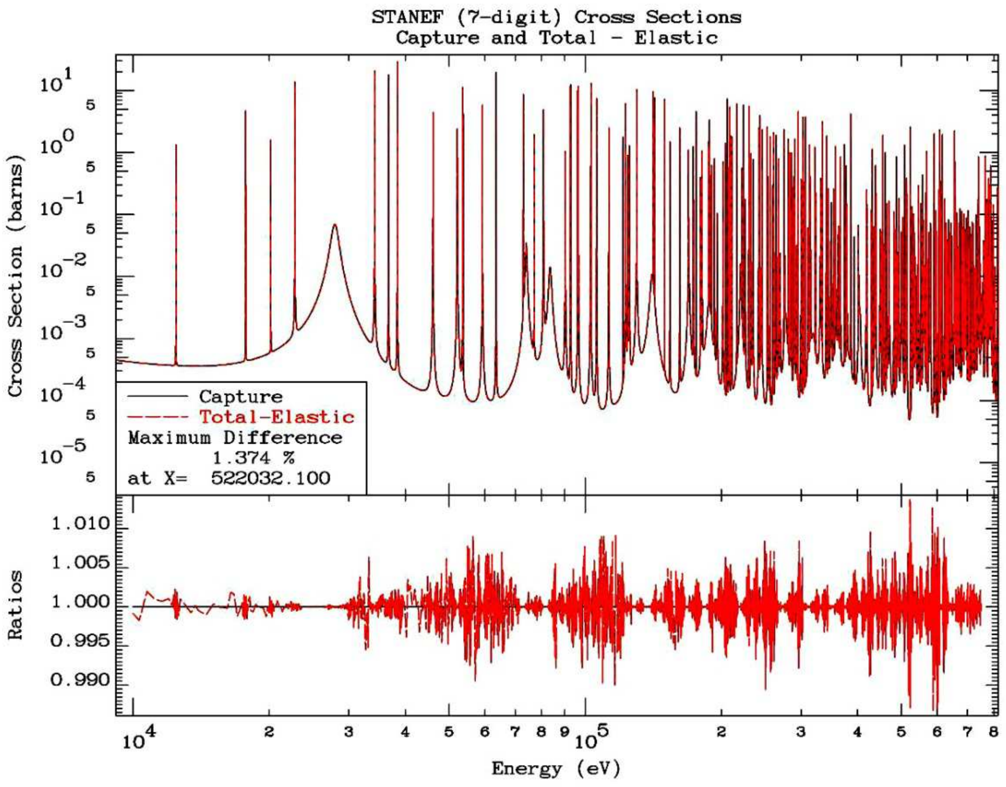
First, the Total and Elastic



Next, the Capture and Total – Elastic



Let us now look at the same results using STANEF 7-digit output comparing Capture to Total – Elastic:



Here we see substantial differences over the entire resolved energy range, in the worst case, up to 1.37%. This isn't even close to our target allowable uncertainty of 0.1%, and it is roughly 100 times as large as the difference we found using 9-digit output; based on truncating 2 digits (9 to 7 digits) this factor of 100 in precision is what we should expect.

Closing Remarks and Recommendations

Fortunately, recently as a result of code comparisons today many of our codes have already been updated to use 9-digit output and produce energy-dependent results within our target allowable uncertainty (we all use 0.1%), these include in alphabetical order: AMPX, NJOY, PREPRO and SAMRML. In contrast STANEF is still producing "E less" 7-digit results, and to my knowledge there is no plan to update it, so PLEASE do not use STANEF; it has now been superseded by ENDF2C, which is more general and highly efficient in performing the one and only task that it is designed to perform = produce results in a standard, officially approved format, to as high a precision as possible.

However, I should stress that as of yet none of our processing codes produce output in the ENDF character format that is 100% FORTRAN, C, and C++ compatible, and at the same time as precise as possible. In pointing this out I include my own PREPRO codes – all of the ENDF data that PREPRO actually process are output in the correct format, but those parts of evaluations that are not processed are copied exactly as read, which can still be in the old "E less" 7-digit format = DANGEROUS!!

In writing this and presenting the earlier results in this report based on STANEF output, I am not trying to single out STANEF as the only code using "E less" 7-digit output. Until recently almost all of our processing codes were producing the results in the same form as STANEF; so the STANEF results shown above are merely indicative of results produced earlier by most of our processing codes. Not until our recent round of code comparisons did we realize the differences that 7- versus 9-digit precision and truly standard formats made in the accuracy and reliability of our ENDF formatted data. Processing code writers were shocked to “see” the Ziggurats (stepped pyramid) produced by their codes, and once they saw them they were able to quickly update their codes to remove them. STANEF must not be blamed for using this “E less” 7-digit format. If anyone is to blame it is me, since I can track this format, that I introduced to ENDF, all the way back to my RIGEL code [7] and ENDF/B-II data, both circa 1970. This format has served us well over the past decades, but today that time is passed, and we should accept the fact that for today’s highly precise evaluated data, and the increased use of computer languages other than FORTRAN (e.g., C and C++) we need the improved compatibility and precision provided by ENDF2C; so PLEASE use it.

Data Centers: PLEASE do not continue to convert ENDF formatted data to a “standard” form using “E less” 7-digit output. In fact there is nothing “standard” about his format; it has never been “standard” or approved by any governing body that defines standards for FORTRAN, C and C++. This “E less” 7-digit format has served us well since I introduced it circa 1970, but today it is not only no longer necessary, it is downright DANGEROUS, both from the viewpoint of standards and precision.

If you do want convert data to a truly standard form while maintaining the highest possible precision PLEASE use ENDF2C. Again, be aware the ENDF2C is not a miracle worker; it cannot increase the precision of the data it reads and then output. So once an evaluation is converted to so called “standard” format using “E less” 7-digit format, there is nothing ENDF2C can do to restore the evaluation to its original format, such as in the example presented earlier, where evaluated resonance energies were supplied by the evaluator to 10-digit precision, but truncated to 7-digit before the evaluation and actually distributed as “official” ENDF data. Is this loss of accuracy important? To me that’s the wrong question. The right question is: Was this truncation necessary? Today the answer is NO!!! Just use ENDF2C and avoid unnecessary truncation and ensure the data is in a truly standard format.

Data Evaluators and Processors: I STRONGLY SUGGEST that you update to use the 9-digit format; you should no longer use the “E less” 7-digit format. This will make your results both compatible and precise. This can be easily done by using subroutine OUT9, which is included in ENDF2C; this is the only routine I use in my PREPRO codes to format ALL floating point numbers for output. The routine is very simple and kept as compatible as possible for use on any computer/system. To use OUT9(X,FIELD), you call it with a 64 bit (REAL*8) number, X, and it returns 11 characters that you can output in the ENDF format (11A1). All of the rules to format the data for output are built into OUT9, so you can concentrate on physics and let OUT9 worry about outputting your results in a compatible format that is as precise as possible.

Data Users: My FINAL WARNING or at least STRONG SUGGESTION is that before you use ANY ENDF data in your applications, whether you received the data from a data center or as output from one of our processing codes ALWAYS RUN IT THROUGH the ENDF2C code BEFORE YOU USE THE DATA!!!! Again, be aware that ENDF2C is no miracle worker and it cannot restore any precision that may have been lost because STANEF or its equivalent was already used to process the data. But ENDF2C can guarantee that its output is in standard format that you can directly use on any computer, using any computer language. This is so simple and fast that you have nothing to lose by doing it – in contrast, potentially you have everything to lose if you do not ensure the data used in your applications are as compatible and accurate as possible = **CAVEAT EMPTOR!!!!!!**

Conclusion

The ENDF2C code is designed to ensure that evaluated data in ENDF format is compatible for direct use in FORTRAN, C and C++ codes. It also ensures that the data include the highest possible precision (9 or 10 digits); here it is IMPORTANT to understand that ENDF2C cannot perform miracles by making data more precise, it can only preserve the accuracy of the data that it reads. Once it goes through STANEF, it is too late to restore any precision that was lost.

ENDF2C is designed to convert ENDF data to standard FORTRAN, C and C++ Format. This code is designed for:
1) ENDF data in any ENDF format = ENDF-1 through ENDF-6.
2) On any type of computer = 32- or 64-bit system/compiler

This code tries to keep things as simple as possible.
1) There are NO INPUT PARAMETERS.
2) It reads an ENDF formatted file named ENDF2C.IN
3) It writes an ENDF formatted file named ENDF2C.OUT
4) It writes a report file named ENDF2C.LST

The code is designed to be easily used on any computer - not only today's computers, but also anything that comes along in the future. So you can be assured that once you start using ENDF2C to produce ENDF Data in Standard Format your compatibility problems are over - not just today, but well into the future.

References

[1] A. Trkov, M. Herman and D.A. Brown "ENDF-6 Formats Manual: Data Formats and Procedures for the Evaluated Nuclear Data Files, ENDF/B-VI and ENDF/B-VII", CSEWG Document **ENDF-102**, Report BNL-90365-2009 Rev. 2, December 2011, Brookhaven National Laboratory.

[2] A. Trkov, editor and maintainer, **STANEF**, unpublished; this code is maintained and distributed by the National Nuclear Data Section, Brookhaven National Laboratory.

[3]. D.E. Cullen, "**PREPRO**2012**:** 2012 ENDF/B Pre-processing Codes", IAEA-NDS-39, Rev. 15, October 2012, Nuclear Data Section, IAEA, Vienna

[4] R.E. McFarlane, et al, "The **NJOY** Nuclear Data Processing System, Version 2012", LA-UR-12-27079, Dec. 2012, Los Alamos National Laboratory.

[5] M. Greene et al, **AMPX**-77: A Modular Code System for Generating Coupled Multigroup Neutron Gamma Cross Section Libraries from ENDF/B-IV and/or ENDF/B –V", ORNL/CSD/TM-283, Oct. 1992, Oak Ridge National Laboratory.

and subsequent publications, such as, M.E .Dunn and N.M. Greene, "**AMPX**-2000: A Cross-Section Processing System for Generating Nuclear Data for Criticality Safety Applications", Trans. Am. Nucl. Soc., 86, 118 (2002), from Oak Ridge National Laboratory.

[6] N.M. Larson, "Updated User's Guide for **SAMMY**: Multilevel R-Matrix Fits to Neutron Data Using Bayes' Equations", ORNL/TM-9179/R8 (ENDF-264/R2), Oct. 2008, Oak Ridge National Laboratory.

[7] D.E. Cullen, "**RIGEL** - A Code to Retrieve ENDF/B Data", BNL-50300, ENDF-110, (TID-4500), 1970, Brookhaven National Laboratory.

[8] D.E. Cullen, "**TART**2012: An Overview of A Coupled Neutron-Photon, 3D, Combinatorial Geometry, Time Dependent Monte Carlo Transport Code", LLNL-TR-577352, June 2012, Lawrence Livermore National Laboratory.

[9] X-5 Monte Carlo Team, **MCNP:** A General Monte Carlo N-Particle Transport Code, Version 5, Volume I: Overview and Theory, X-5 Monte Carlo Team, Los Alamos National Laboratory report LA-UR-03-1987, April 24, 2003.