TECHNICAL REPORT
IGE–305

A PYNJOY TUTORIAL

A. Hébert

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
October 3, 2016

Contents

List of Figures

Chapter 1

Python script to generate DRAGLIB and ACELIB

This chapter will describe the procedure for generation of cross section libraries in DRAGLIB format and ACE format. DRAGLIB format libraries will be used for lattice analysis using the code DRAGON and ACE formatted libraries will be used for the analysis using Monte Carlo code MCNP5. Both the sets of libraries have been generated starting from the same evaluated datafiles, that were recommended by the IAEA WIMS Library Update Project (WLUP). A total of 200 nuclides have been used for analysis in this thesis. The libraries have been generated by using a Python script. The script is seen in Figure 1. The input for selected nuclides is presented in Figure 2. Another interesting addition to the script is to compare the generated ACELIB, with the PENDF tape generated by an alternate procedure as recommended by IAEA. This involves regenaration of the PENDF file from the ACELIB using a code package called ACELST. Modules from PREPRO code system developed by Red Cullen are subsequently used on the evaluated data files and PENDF tape is generated. The modules used are LINEAR, RECENT, SIGMA1. Subsequently one can do a comparison of the two PENDF files using COMPLOT, which is another important module of PREPRO code system.

This chapter has three main sections. First section will give a general description of instance variables and methods needed to generate the various libraries. The second section will describe the steps needed to generate DRAGLIB. The third section will describe the generation of ACELIB and the final section will describe verification of generated ACE libraries by procedure recommended by IAEA.

1 PyNjoy Script

The generation of libraries has been made simple by the use of Python script. A complete processing of an evaluated datafile is done by invoking modules of NJOY through an object oriented Python script. A unique class named PyNjoy contains the various intsance variables and methods required to use NJOY in the simplest possible way. Any modification of this model is possible in order to accommodate various processing requirements. The PyNjoy script is located on `Version4_wc/Njoy99/python/PyNjoy.py`.

2 PyNjoy instance variables

Python parameters used in the input are described in this section.

`self.evaluationName` - Path of directory where you want to store the pendf, gendf, draglib and acelib files. The path can be prefixed by `/tmp/` to force the files to be created locally on the `/tmp` directory.
`self.execDir` - Path of directory where the executable of Njoy is found.
`self.legendre` - Order of Legendre polynomials for neutrons (= 1 for LINEAR anisotropy in LAB – default)
`self.legendregg` - Order of Legendre polynomials for gamma particles (default: = 6)
`self.nstr` - Option for a particular neutron group structure (= 22 for the XMAS 172–group structure)
`self.gstr` - Option for a particular gamma group structure, for producing neutron–gamma coupled sets (equal to zero by default)
`self.iwt` - Type of flux weighting in GROUPR (= 1 for user-defined spectra; = 3 for $1/E$ weighting; = 4 recommended/default)
`self.wght` - User-defined weighting spectra to be used if `self.iwt` = 1. Given as `"""`–delimited string.
`self.autolib` - Three-component tuple containing the energy limits of the autolibs (must correspond to energy-group limits) and the elementary lethargy width of the autolibs
`self.temperatures` - Value of temperatures at which the cross sections are generated
`self.hmat` - Material name that is included in the DRAGLIB – User dependent
`self.mat` - `mat` number of nuclide under consideration
`self.hmatgg` - Photo-atomic element name that is included in the DRAGLIB – User dependent
`self.matgg` - Photo-atomic `mat` number of element under consideration
`self.za` - ZA number, mainly required for generation of $S(\alpha, \beta)$ cross section in ACER
`self.scatName` - Name of $S(\alpha, \beta)$ cross section identifier for inclusion in xsdir
`self.suff` - The suffix to be attached to nuclide in ACELIB – User dependent
`self.evaluationFile` - Path of the evaluated datafile.
`self.scatteringLaw` - Path of file having thermal scattering data (default = `None`)
`self.scatteringMat` - `mat` number in scattering data file
`self.fission` - Choice for including delayed neutron fission data in GROUPR module
`self.Espectra` - $G_{\text{chi}} + 1$ values of energy limits (eV) defining the energy-dependent fission spectra. If this instance variable is not set, a unique fission spectrum is used. Recommended limits for fast-reactor applications are `( 1.100028e-4, 1.831564e5, 4.978707e5, 1.353353e6, 1.964033e7 )`.
`self.ss` - Two-component tuple containing energy limits in eV for the self-shielding domain
`self.potential` - Value of the potential cross section used in the flux calculator of GROUPR
`self.dilutions` Tuple containing the dilution values that need to be considered for calculation of resonance integrals and probability tables
`self.dirName` - Directory name to store data for independent verification of ACELIB.
`self.tempace` - Temperature at which ACELIB needs to be generated.
`self.eFiss` - Fission energy in MeV. Used in cases where this value is not available in the evaluation.
`self.branchingNG` - Radiative capture isomeric branching ratio (default = `None`). If you use this value, don't forget to reset it to `None` after the isotope is completed.
`self.branchingN2N` - N2N isomeric branching ratio (default = `None`). If you use this value, don't forget to reset it to `None` after the isotope is completed.
`self.purr` - Set to 1 to use PURR module. By default, use UNRESR.
`self.oldlib` - Name of an existing DRAGLIB file that is modified by `self.draglib()`. The existing DRAGLIB, in ASCII format, must be placed in the same directory where the input Python file is found. This DRAGLIB is copied in the execution directory `self.execDir` and updated by `self.draglib()`.

The following instance variables are related to the production of a single Wimslib isotope using module `wimsr`:
`self.wmat` - Identification of material for the WIMS library. This identification is a floating-point number.

`self.sgref` - Reference sigma zero. All isotopes are evaluated at dilution `self.sgref` in case where the resonance tables are not used. This variable has a strong effect on the WIMSLIB values. **Important:** `self.sgref` must be selected in tuple `self.dilutions`. The default value is `self.sgref` $= 1.0 \times 10^{10}$ barn, corresponding to infinite dilution.
`self.jp1` - Number of components in the user-defined current weighting spectra (to be used for the transport correction).
`self.goldstein` - Value if a Goldstein-Cohen parameter. It is assumed that all Goldstein-Cohen parameters are constant in the resonant-energy groups.
`self.p1flx` - User-defined current weighting spectra (to be used for the transport correction) if `self.jp1` > 0. Given as `"""`–delimited string.
`self.iverw` - $= 4$: Produce a WIMS-D4 formatted library; $= 5$: Produce a WIMS-E formatted library.
`self.yields` - User-defined burnup data for an isotope (cards 5 and 6 of `wimsr` module dataset). Given as `"""`–delimited string.
`self.ifprod` - Fission product flag. $= 0$: Not a fission product (default); $= 1$: Fission product, no resonance tables; $= 2$: Fission product, with resonance tables.

3 PyNjoy methods

`self.pendf([eaf])` - To generate point ENDF file (to be used as starting point for all other data type generations including DRAGLIB, ACE, WIMSD etc) using the modules MODER, RECONR, BROADR, PURR (if dilutions present), THERMR. If `eaf=1`, a eaf-compatible simplified processing is performed.
`self.gendf([eaf])` - To generate group ENDF file using modules MODER and GROUPR. If `eaf=1`, a eaf-compatible simplified processing is performed.
`self.gamma()` - To generate photo-atomic (gamma) group ENDF file using modules MODER, RECONR and GAMINR.
`self.draglib([fp])` - To generate a DRAGLIB file using modules MODER, DRAGR and add/update the new isotopic data in the DRAGLIB file. If `fp=1`, the scattering information are stored as diagonal matrices in the DRAGLIB.
`self.matxs()` - To generate an ascii MATXS file using modules MODER and MATXSR.
`self.makeFp()` - call `self.pendf()`, `self.gendf()` and `self.draglib(fp=1)` for a single fission product.
`self.burnup()` - Process burnup data for the complete library. An important file that is needed while generating the burnup data is named as "chain(self)". If `self=candu` then the file is named `chaincandu`. This file contains the information of energy from all isotopes generated using single DRAGR runs. This file is now generated automatically.
`self.acer()` - To generate ACELIB using modules MODER, RECONR, BROADR, PURR (if dilutions present), THERMR and ACER.
`self.wims()` - To generate an ascii Wimslib file using modules MODER and WIMSR.
`self.comp()` - To run the verification test using modules ACELST, LINEAR, RECENT, SIGMA1, FIXUP and COMPLOT.

4 DRAGLIB Library Generation

The modules that will be used in the generation of DRAGLIB are MODER, RECONR, BROADR, PURR (if dilutions present), THERMR, GROUPR and DRAGR. Figure 3 gives the flow chart for generation of DRAGLIB formatted library. It is important to identify the nuclides that are needed to be included as part of library and corresponding evaluated datafiles from respective data centres need to be compiled in a particular directory. This will help in cross verification and assessment at any stage of library generation. In this section specific examples of elements will be provided to understand the nuances of DRAGLIB library generation. The examples will be such that all the reactor type materials will be chosen. They are scattering material (heavy water), structural material (Zirconium), fission product (Xe135), Actinide (U-235). A special example for burnup dependent data will also be provided. For isotopes that have resonances and whose presence in fuel can alter the flux in the energy region between 2.76792 eV and 1.66156e4 eV, cross sections are generated at specific dilution values. The choice of dilution values is shown in Table-1 and shown in Figure 4. The value for potential scattering cross section is obtained using the Fortran code `getmf2.f` which is provided by IAEA. Using this code, and the evaluated datafile for the particular element, one can obtain the value for `self.potential`. In general we cannot provide more than ten temperature values and ten dilution values. If one has to generate cross sections for more than ten dilution values, it has to be split, as shown in example for U-235. After each “`self.dragr()`” run, one will obtain a file “`out_draglib_elementname`”. Energy information is recovered from this ascii file by method `self.burnup()` and is collected in a file named `'chain' + self.evaluationName` (stored on directory `self.evaluationName`). Sometimes, it is likely that fission energy is not provided in the tape. In that case one obtains “`????????`” in the “`out_draglib_elementname`” file. In that case, one has to obtain the energy value from some other source (typically, from another evaluation) and provide it using the `self.eFiss` instance variable, even if one doesnot use the tape for generation of multigroup data.

File 8 of ENDF evaluation contains half-lives, decay modes, decay energies, and radiation spectra for most isotopes. Information concerning the decay of the reaction products is also given in this file. In addition, fission product yield data (MT=454 and 459) for fissionable materials and spontaneous radioactive decay data (MT=457) for the nucleus are included.

File 8 information is processed by module DRAGR. A large number of fission products are included in the evaluated file for each element capable of undergoing fission. For example, in the fission product yield data file included in ENDF/B-VI rel. 8, one can notice that there are information of 1232 fission products for 0.0253 eV fission of ^{233}U, 1247 fission products for 0.0253 eV fission of ^{235}U etc. But the evaluations are not available for all the nuclides, as most of them have very short half-lives and in the reactor context, can be considered insignificant. They are subsequently lumped by a procedure that is built in DRAGR. If there are nuclides with long half lives, but are not available as evaluated files, a warning is provided before lumping the corresponding element. The DRAGR user has the complete control over the lumping process. DRAGR currently has no capability to produce pseudo fission product, i.e., custom library isotopes made from the combination of many minor ENDF fission products. All the isotopes missing in the `'chain' + self.evaluationName` file are tested against a lumping criterion and are lumped. The criterion for lumping a depleting isotope is a half life less than thirty days and a fission yield less than 0.01%. If this criterion is not met, this isotope is lumped and a warning message is issued.

Information on energies for various reaction types like (n, γ), (n, f), (n, 2n), (n, 3n), (n, 4n), (n, α), (n, p), (n, 2α), (n, np), (n, d), (n, t) are recovered from earlier DRAGR single-isotope calculations and used for inclusion in relevant depletion data in DRAGLIB format. The fission energy (n, f) is obtained from MF1 MT458 and the energy from delayed betas and gammas are subtracted from it. Information regarding energies for other reactions are derived by DRAGR from MF3. The corresponding MT numbers for the above mentioned reactions (other than (n, f)) are 102, 16, 17, 37, 107, 103, 108, 28, 104, 105 respectively. The complete information required to do the depletion calculations is provided in ten specific records of the DRAGLIB file.

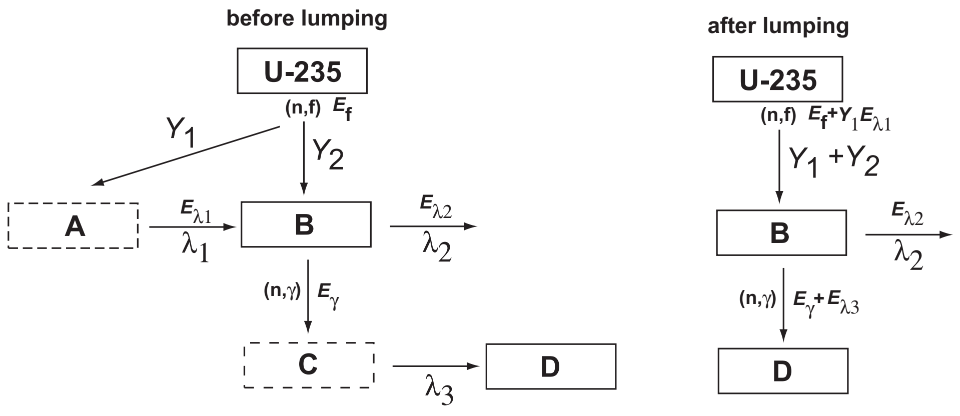

Figure 1.1: Lumping of isotopes A and C

4.1 Main object instanciation

The Python dataset starts with the instanciation of an object (named `candu` in the following examples) and with the definition of a few instance variables that are common to many isotopes.

```
#!/usr/local/bin/python
from PyNjoy import *
from os import uname
candu = PyNjoy()
candu.evaluationName = "/tmp/endfb7r0"
candu.execDir = "../" + uname()[0]
candu.nstr = 22
candu.iwt = 4
candu.autolib = (2.76792, 677.2873, 0.00125)
candu.legendre = 1
candu.Espectra = None
```

4.2 Heavy water

Heavy water is used in CANDU reactors as moderator and coolant. So it is important to generate consistent data for efficient analysis of CANDU lattices. The instance variables and methods that will be used are

```
candu.hmat = "H2_D2O"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 128
candu.evaluationFile = "/home/develop/Tripoli4/JEF2/eval/jef2.neutron.H2.bcd"
candu.scatteringLaw = "/home/develop/Tripoli4/JEF2/eval/jef2.neutron.D_D2O.bcd.therm"
candu.scatteringMat = 11
candu.fission = None
candu.dilutions = None
candu.pendf()
candu.gendf()
candu.draglib()
```

4.3 Zirconium

Zirconium is used in CANDU reactors as cladding material and also for pressure tube and calandria tube. Zirconium has some resonances and as a result of this it is important to generate the cross sections of zirconium for certain dilution values. The instance variables and methods that will be used are

```
candu.hmat = "Zr0"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 4000
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/jendl306.asc"
candu.fission = None
candu.ss = (2.76792, 1.66156e4)
candu.potential = 6.5144
candu.dilutions = ( 1.e10, 10000.0, 3549.18335, 1259.67004, 447.079956, 158.676849, \
56.3173141, 19.9880447, 7.09412289, 2.51783395 )
candu.pendf()
candu.gendf()
candu.draglib()
```

4.4 Xe135

Xenon is a very important fission product in nuclear reactors. It is very important to estimate the number density of this nuclide as a function of burnup. This will help in estimating reactivity variations due to change in concentration of the nuclide. By using makeFp option, we avoid generating scattering matrices in (NG X NG) format, where NG is number of groups. We instead generate only the scattering matrices along the diagonal.

```
candu.hmat = "Xe135"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.scatteringLaw = None
candu.legendre = 0
candu.fission = None
candu.dilutions = None
candu.mat = 5458
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/jendl310.asc"
candu.makeFp()
```

4.5 U235

U-235 is one of the most prevalently used fissile material for energy production. In case of CANDU reactors, natural uranium is used as fuel, where weight (%) of U-235 is 0.711.

```
candu.hmat = "U235"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 9228
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/U-235"
candu.fission = 2
candu.ss = (2.76792, 1.22773e5)
candu.potential = 11.6070
candu.dilutions = ( 1.e10, 94.5317612, 56.3173141, 33.5510521, 19.9880447, \
11.9078817, 7.09412289, 4.22632504, 2.51783395, 1.5 )
candu.pendf()
candu.gendf()
candu.draglib()
candu.dilutions = ( 1.e10, 10000.0, 5957.50244, 3549.18335, 2114.42676, 1259.67004, \
750.448669, 447.079956, 266.347961, 158.676849 )
candu.pendf()
candu.gendf()
candu.draglib()
```

4.6 Process burnup

It is important to identify the tapes provided by evaluators which contains information on fission yields and decay chains. These files are provided as mentioned below, along with the chain(self) file mentioned already.

```
candu.fissionFile = "/home/karam/Njoy99/evaluations/database/TAPE.107"
candu.decayFile = "/home/karam/Njoy99/evaluations/database/TAPE.106"
candu.burnup()
```

5 ACELIB Library Generation

The modules that will be used in the generation of ACELIB are MODER, RECONR, BROADR, PURR(if dilutions present), THERMR and ACER. Figure 3 gives the flow chart for generation of ACE formatted library. In this section specific examples of elements will be provided to understand the nuances of ACELIB library generation. The examples will be along the same lines as that for DRAGLIB library generation, i.e scattering material (heavy water), structural material (Zirconium), fission product (Xe135), Actinide (U-235). The prsent script is such that the ACELIBs are appended in a single file named “acecandu” and is available in the same directory as the draglib file. The other important file that is generated is the “acexsdir”, which contains the information about the nuclides for which the cross sections are generated and the temperature at which the ACELIB is generated. A small code has been written-“append.f”, which will read the file acecandu and acexsdir and create the file “myxsdir”. This file has to be appended to existing xsdir file provided with MCNP5 data. It is important to provide suffix values “`self.suff`” for different temperatures. This will be automatically appended to the `ZA` value and written in main ACELIB and xsdir file. Care should be taken not to repeat the “.suff” value already used in xsdir file for other evaluations. For each temperature provide different “`self.dirName`” so that all the required data for comparison with PENDF tape generated using PREPRO code is made possible. The “`self.comp()`” does the task of verifying the ACELIBs generated, and is described in the next section. The example provided here helps in generating ACELIB alone. In case a DRAGLIB file also is to be generated, please refer to Figure 2.

5.1 Heavy water

```
candu.hmat = "H2_D2O"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 128
candu.za = 1002
candu.scatName = "hwtr"
candu.evaluationFile = "/home/develop/Tripoli4/JEF2/eval/jef2.neutron.H2.bcd"
candu.scatteringLaw = "/home/develop/Tripoli4/JEF2/eval/jef2.neutron.D_D2O.bcd.therm"
candu.scatteringMat = 11
candu.fission = None
candu.dilutions = None
candu.pendf()
candu.dirName = "D2O-1"
candu.tempace = ( 293.6,)
candu.suff=0.20
candu.acer()
candu.comp()
candu.dirName = "D2O-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.comp()
candu.dirName = "D2O-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()
candu.comp()
```

5.2 Zirconium

```
candu.hmat = "Zr0"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 4000
candu.za = 40000
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/jendl306.asc"
candu.fission = None
candu.dilutions = ( 1.e10, 10000.0, 3549.18335, 1259.67004, 447.079956, \
158.676849, 56.3173141, 19.9880447, 7.09412289, 2.51783395 )
candu.pendf()
candu.dirName = "Zr-1"
candu.tempace = ( 293.6,)
candu.suff=0.20
candu.acer()
candu.comp()
candu.dirName = "Zr-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.comp()
candu.dirName = "Zr-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()
candu.comp()
```

5.3 Xe135

```
candu.hmat = "Xe135"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.scatteringLaw = None
candu.legendre = 0
candu.fission = None
candu.dilutions = None
candu.mat = 5458
candu.za = 54135
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/jendl310.asc"
candu.makeFp()
candu.dirName = "Xe135-1"
candu.tempace = ( 293.6,)
candu.suff=0.20
candu.acer()
candu.comp()
candu.dirName = "Xe135-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.comp()
candu.dirName = "Xe135-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()
candu.comp()
```

5.4 U235

```
candu.hmat = "U235"
candu.temperatures = ( 293.6, 323.6, 573.6, )
candu.mat = 9228
candu.za = 92235
candu.scatteringLaw = None
candu.legendre = 0
candu.evaluationFile = "/home/karam/Njoy99/evaluations/database/U-235"
candu.fission = 2
candu.dilutions = ( 1.e10, 10000.0, 5957.50244, 3549.18335, 2114.42676, \
1259.67004, 750.448669, 447.079956, 266.347961, 158.676849 )
candu.dirName = "U235-1"
candu.tempace = ( 293.6,)
candu.suff=0.20
candu.acer()
candu.comp()
candu.dirName = "U235-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.comp()
candu.dirName = "U235-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()
candu.comp()
```

6 Verification of ACELIB

Cross section processing verification is described in ref(Trkov), where the steps involved is mentioned in detail. For the sake of completeness, we have presented the discussion as part of this chapter. The ACE ("A Compact ENDF") libraries produced by the NJOY nuclear data processing system, involves some data processing, which is susceptible to code errors. It is important to verify the ACELIB by comparing it with an independently processed data set. This may highlight possible errors due to LINEARization of data, resonance reconstruction from the resonance parameters included as part of the ENDF tape, and conversion of data representation for angular and energy distributions, where applicable. ENDF Pre-Processing codes included as part of PREPRO code system is independent to NJOY and can be considered for the data verification tasks

At the outset let us consider that the ACELIB has already been generated. The flow chart that needs to be subsequently followed for data verification is as under:

1. Run ACELST to generate the ACELIB summary of contents and to convert the data as much as possible into pointwise ENDF format. 2. Retrieve the basic evaluated nuclear data file from which the ACE library was generated. 3. Run LINEAR on the tested file to LINEARise any non-linear interpolation laws. 4. Run RECENT on the output of LINEAR to reconstruct cross sections from the resonance parameters. 5. Run SIGMA1 on the output of RECENT to Doppler-broaden the cross sections to the same temperature that is specified in the ACELIB. 6. Run FIXUP on the output of SIGMA1 to reconstruct the redundant cross sections. 7. Run COMPLOT to compare the output of ACELST and SIGMA1.

Each of the tasks above serves a specific purpose and provides useful information. The ACELST code is included in the SIGACE code package, which has been developed to generate high temperature ACELIBs from already provided low temperature files for use in MCNP. This code provides a compact listing of the contents of the ACELIB, which is useful when insufficient information is provided on how the file was generated. The COMPLOT comparison of the ACELST and SIGMA1 output can reveal any inconsistency in the redundant reactions. For example, the total cross section in the ACELST output from the ACELIB is given by the sum of the partials, but in SIGMA1 it is calculated directly from the total cross section. An inconsistency in the resonance range might indicate that the implied competitive widths in the resonance parameters are inconsistent with the pointwise data in file MF3 for the competing reaction (usually the inelastic cross section).

The codes are LINEAR, RECENT, SIGMA1, FIXUP, COMPLOT are provided free of cost at the website "http://www-nds.iaea.org/ndspub/endf/prepro/". The code ACELST.f is included in SIGACE code package, whcih can be downloaded from "http://www-nds.iaea.org/fendl21/downloads/". All the tolearnces are within 0.1% for data generation using PREPRO. It is important that this value is the same as that is used for PENDF file generation to make DRAGLIB or ACELIB.

Figures 6-8 gives the comparison of plots obtained using COMPLOT. It shows the magnitude of difference between generated ACELIB and PENDF generated using PREPRO. It was noted that the large difference in cross section is at the interface of resolved resonance and unresolved resonance region. This is not a major cause of concern though.

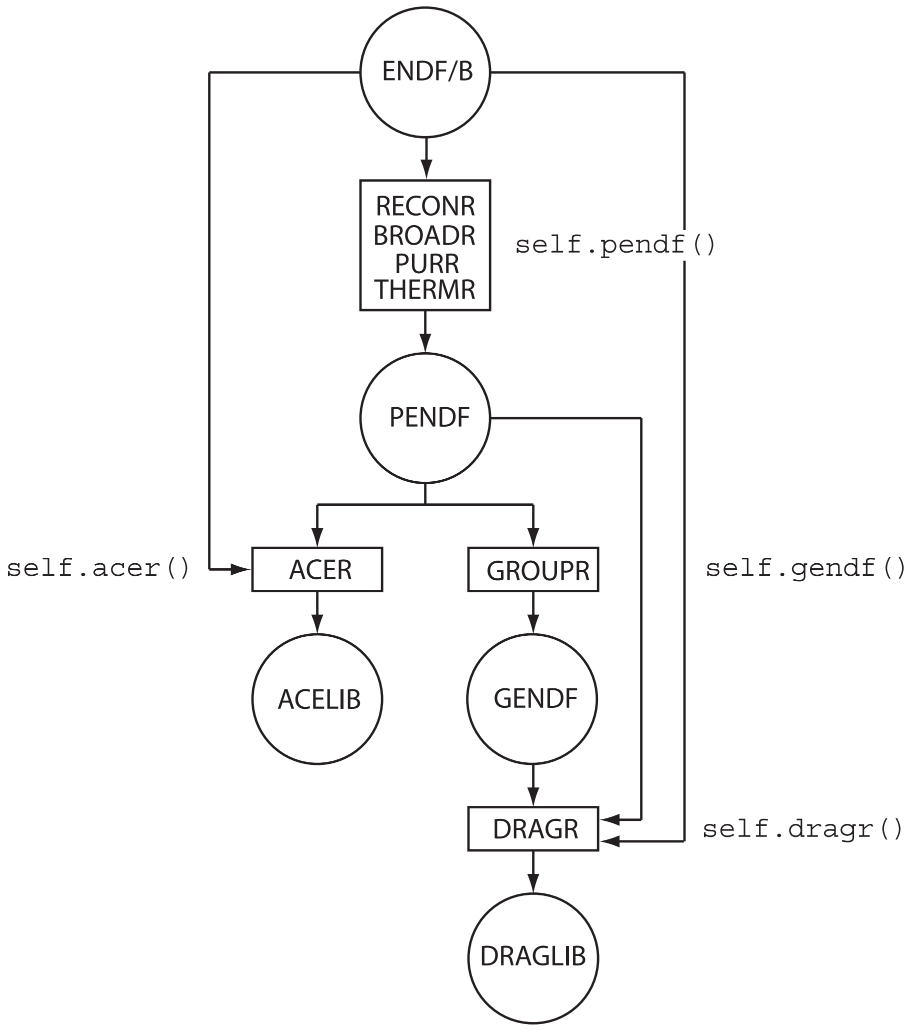

Figure 1.2: Flowchart for DRAGLIB/ACELIB Production

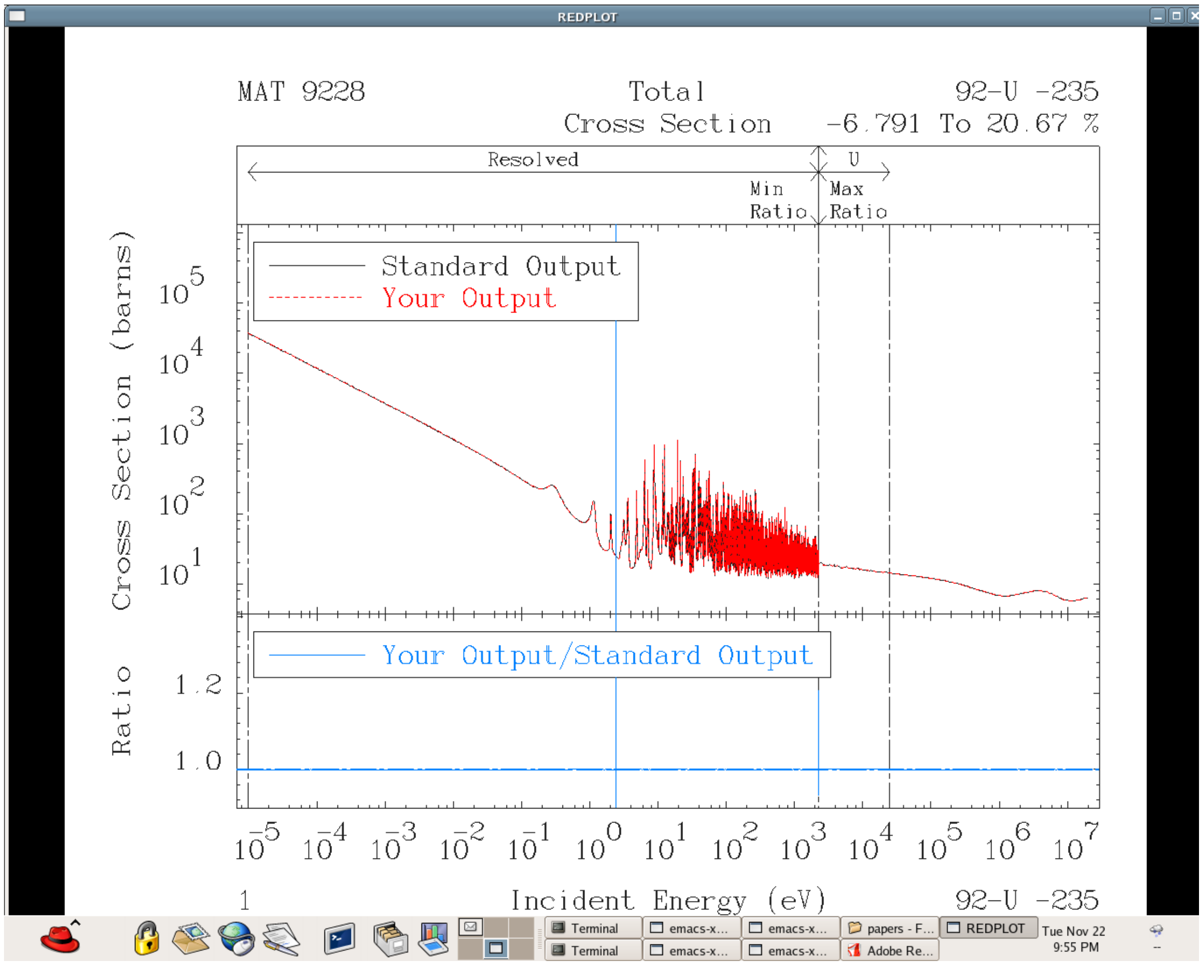

Figure 1.3: COMPLOT - Total cross section of U-235

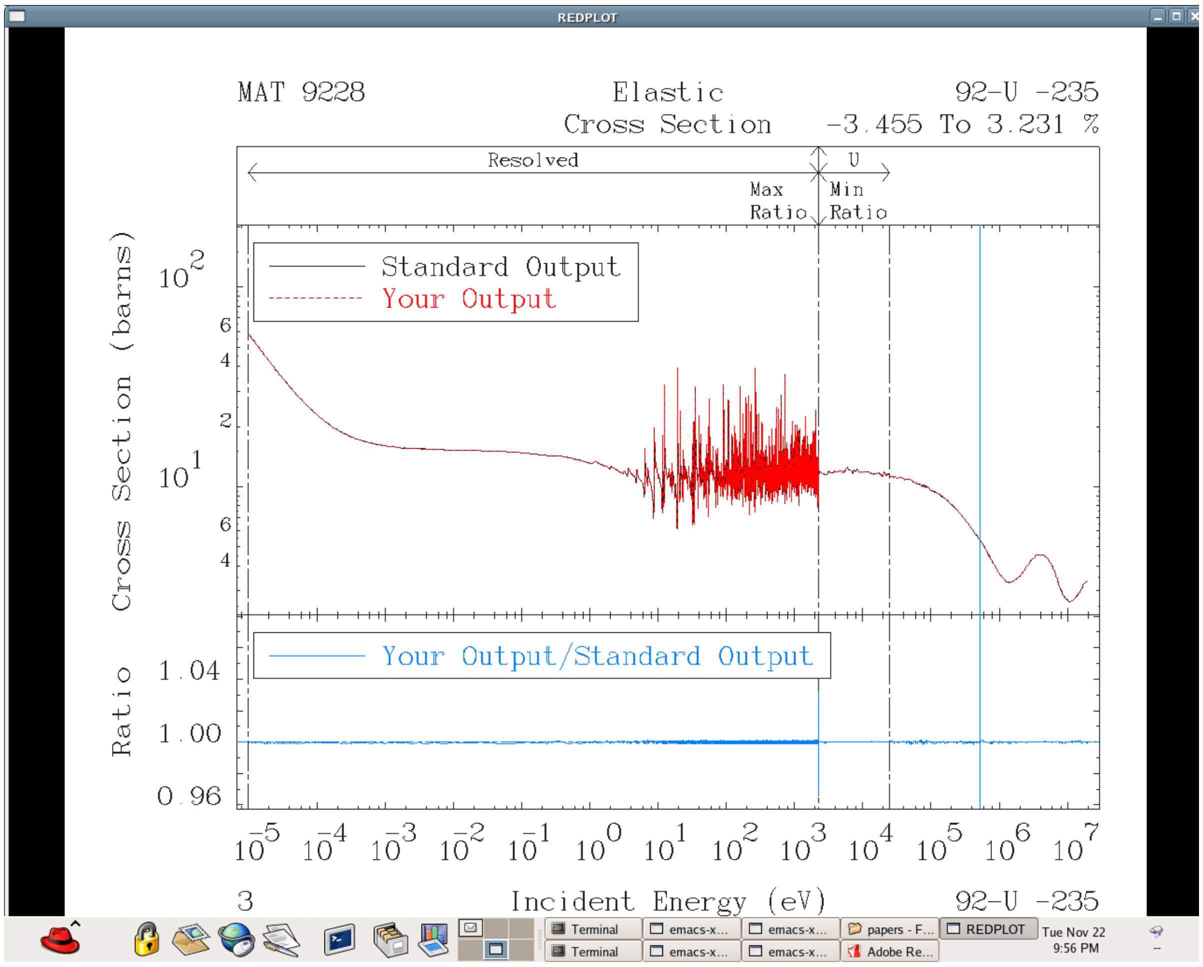

Figure 1.4: COMPLOT - Elastic cross section of U-235

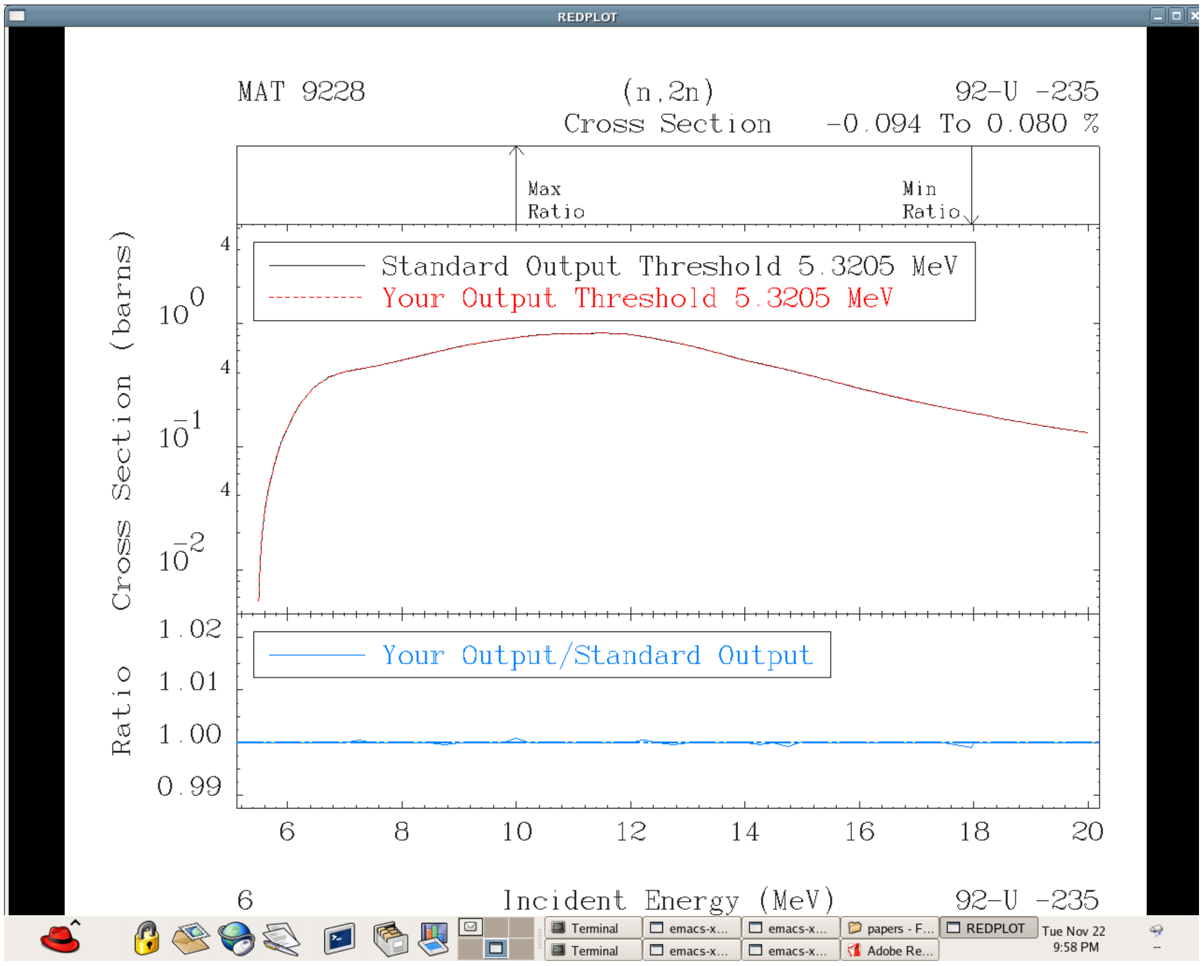

Figure 1.5: COMPLOT - N,2N cross section of U-235

Chapter 2

Creation of binary Dragligs

The PyNjoy.py script presented above produces an ascii-formatted (80-column) Draglib file in LCM/XSM export format. Dragon is expecting binary Draglibs, either in little– or big–endian direct–access XSM format. The required conversion can easily be performed using the Ganlib capabilities of Dragon.

Three files, present in `Version4_wc/Dragon/data/`, can be adapted to perform the conversion. These files are:

a2b_drglib.access: This Bourne–shell script is used to specify the location of the ascii-formatted Draglib produced by Njoy.

```
#!/bin/sh
#
# ACCESS FILE OF CONVERSION DATA SET a2b_drglib.x2m
#
if [ $# = 0 ]
   then
   echo "usage: a2b_drglib.access directory" 1>&2
   exit 1
fi
echo access a2b_drglib.access
ln -s "$1"/../Njoy99/python/Jeff3.1/draglibJeff3.1 EXPORT
ls -l
```

a2b_drglib.x2m: This CLE-2000 Dragon input data file invokes the equality module of Ganlib to perform the conversion.

```
*----
*  Draglib conversion, from ascii export to xsm binary
*----
XSM_FILE DRGLIB ;
SEQ_ASCII EXPORT ;
MODULE UTL: END: ;
*
DRGLIB := EXPORT :: EDIT 10 ;
UTL: DRGLIB :: DIR ;
END: ;
```

a2b_drglib.save: This Bourne–shell script is used to specify the location of the binary-formatted Draglib produced by the equality module of Ganlib. If the conversion is performed on a little-endian computer (Intel, OSF1), the Draglib is little-endian.

```
#!/bin/sh
#
# SAVE FILE OF CONVERSION DATA SET a2b_drglib.x2m
#
if [ $# = 0 ]
   then
   echo "usage: a2b_drglib.save directory" 1>&2
   exit 1
fi
echo access a2b_drglib.save
MACH=‘uname -s‘
Sysx="‘echo $MACH | cut -b -6‘"
if [ $Sysx = "CYGWIN" ]; then
   MACH=‘uname -o‘
elif [ $Sysx = "Darwin" ]; then
   MACH=‘uname -sm‘
elif [ $Sysx = "SunOS" ]; then
   MACH=‘uname -sm‘
fi
if [ "$MACH" = "Linux" -o "$MACH" = "OSF1" -o "$MACH" = "Cygwin" -o "$MACH" = "SunOS i86pc" \
       -o "$MACH" = "Darwin i386" -o "$MACH" = "Darwin x86_64" ]
 then
  echo 'use little endian libraries'
  pos=$1/../../libraries/l_endian
else
  echo 'use big endian libraries'
  pos=$1/../../libraries/b_endian
fi
mv DRGLIB $pos/draglibJeff3.1
ls -l
```

Once a Draglib has been converted in direct-access binary format, it is still possible to inspect its contents using the UTL: utility module. Here is an example where the list of available isotopes in the library and where the tabulated temperatures and non-infinite dilutions of U235 are printed.

```
*----
*  Draglib inspection (xsm direct-access binary format)
*----
XSM_FILE DLIB_J2 ;
MODULE UTL: END: ;
*
* list of isotopes:
UTL: DLIB_J2 :: DIR ;
*
* temperatures of U235:
UTL: DLIB_J2 :: STEP UP U235 DIR IMPR TEMPERATURE * ;
*
* non-infinite dilutions of U235 at 293 K:
UTL: DLIB_J2 :: STEP UP U235 STEP UP SUBTMP0001 DIR IMPR DILUTION * ;
END: ;
```

together with the access script file

```
#!/bin/sh
```

```
#
# Access Draglib in XSM DA binary format for look_drglib.x2m
#
if [ $# = 0 ]
   then
   echo "usage: look_drglib.access directory" 1>&2
   exit 1
fi
echo access look_drglib.access
MACH=‘uname -s‘
if [ $MACH = "Linux" -o $MACH = "OSF1" ]
 then
  echo ’use little endian libraries’
  pos=$1/../../libraries/l_endian
else
  echo ’use big endian libraries’
  pos=$1/../../libraries/b_endian
fi
if [ -f "$pos"/draglibJef2p2 ]
   then
    ln -s "$pos"/draglibJef2p2 DLIB_J2
fi
ls -l
echo "look_drglib access script terminated"
```