

TECHNICAL REPORT
IGE-361

A PYNJOY-2012 TUTORIAL

A. HÉBERT AND R. CHAMBON

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal
March 31, 2024

Contents

Contents	ii
List of Figures	iii
1 Python script to generate DRAGLIB and ACELIB	1
1.1 PyNjoy Script	1
1.1.1 PyNjoy instance variables	1
1.2 PyNjoy methods	3
1.3 DRAGLIB Library Generation	4
1.3.1 The modules and data recovery used in the process	4
1.3.2 Sequential calculations	5
Main object instantiation	5
Heavy water	6
Zirconium	6
Xe135	6
U235	7
Process burnup	7
1.3.3 Parallel calculations	8
Main object instantiation	8
Thermal scattering	9
Nominal isotopes	10
Fissionable isotopes	10
Fission products	11
Isotopic DRAGLIB computation	12
DRAGLIB concatenation	13
Process burnup	13
1.3.4 CPU cost	13
1.4 ACELIB Library Generation	13
1.4.1 Heavy water	14
1.4.2 Zirconium	14
1.4.3 Xe135	15
1.4.4 U235	15
2 Format conversion of Draglibs	17

List of Figures

1	Flow chart for generation of ACELIB and DRAGLIB formatted libraries	4
2	Lumping of isotopes A and C	5

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. 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 regeneration 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 COMPLIT, 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.

1.1 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 instance variables and methods required to use NJOY in *sequential mode*. Any modification of this model is possible in order to accommodate various processing requirements. The original `PyNjoy` script is located on `Version4_wc/Njoy99/python/PyNjoy.py`. A newer version of the script has been recently made to introduce parallel computing. This version is available on webpage <http://www.polymtl.ca/merlin/pynjoy2012.htm> and is located on file `Njoy2012_EPM/python/PyNjoy.py`. It is backward compatible with previous input files.

1.1.1 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. Note that the path is relative to the current directory.

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.legendrebe - Order of Legendre polynomials for electrons (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 (equal to zero by default)

self.estr - Option for a particular electron group structure (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.hmatbe` - Electro-atomic element name that is included in the DRAGLIB – User dependent.
`self.matgg` - Photo-atomic `mat` number of element under consideration.
`self.matbe` - Electro-atomic `mat` number of element under consideration.
`self.zaid` - ZA number, mainly required for generation of $S(\alpha, \beta)$ cross sections in ACER.
`self.scatName` - Name of $S(\alpha, \beta)$ cross section identifier for inclusion in `xmdir`.
`self.suff` - The suffix to be attached to nuclide in ACELIB – User dependent.
`self.evaluationFile` - Path of the ENDF evaluated datafile.
`self.relaxationFile` - Path of the ENDF atomic relaxation datafile.
`self.stoppingPowerFile` - Path of the ENDF stopping power datafile.
`self.scatteringLaw` - Path of file having thermal scattering data (default = `None`).
`self.scatteringMat` - `mat` number in scattering data file.
`self.thermal_cutoff` - thermal cutoff energy (in eV). This value is used in modules THERMR, RESKR, DRAGR and ACER. Default value is 4.0 eV.
`self.reskEhi` - upper energy (in eV) in the resolved energy domain where the *resonant elastic scattering kernel* (RESK) effect is taken into account. The lower energy is set to `self.thermal_cutoff`. By default, an asymptotic kernel is used.
`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 the same format (either ASCII or BINARY), 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()`.
`self.uniform` - Set to 0 to use ASCII format for the DRAGLIB, 1 for SEQ_BINARY. Default value is 0. WARNING, the SEQ_BINARY is created with fortran 90 which includes delimitation markers. It is not compatible with other language. Thus it has to be converted into ASCII for use in DRAGON. However, the gain of time calculation makes the process worth it.
`self.groupboundaries` - Arbitrary multigroup structure for GROUPR module. Default value is `None`.
`self.concat` - Set to 0 to use only one DRAGLIB that is updated for each new isotopes, 1 to compute an independent DRAGLIB for each isotope (meaning that the complete DRAGLIB will be concatenated after. Default value is 0.

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** = 1.0×10^{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.

The following instance variables are related to the production of an ACE file using module **acer**:

self.acesuff - Id suffix for **zaid**. Default value is “.00”.

self.acefname - ACE file name. Default value is “acecandu”.

self.acedirName - ACE directory name. Default value is **None**.

1.2 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 GROUPT. If **eaf=1**, a eaf-compatible simplified processing is performed.

self.gamma() - To generate photo-atomic (gamma) group GENDF file using modules MODER, RECONR and GAMINR.

self.electron() - To generate electro-atomic (electron) group GENDF file using modules MODER, RECONR and ELECTR.

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.draglibcat() - To concatenate two DRAGLIB files using module DRAGR. The coherence of the energy meshes is verified. Both DRAGLIB must be the same format (either ASCII or SEQ_BINARY)

self.draglibconv() - To convert a DRAGLIB file using module DRAGR. if **self.uniform** is equal to 1, a SEQ_BINARY) DRAGLIB is transformed into a ASCII file, for a value of 0, it is the oposite.

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.

1.3 DRAGLIB Library Generation

1.3.1 The modules and data recovery used in the process

The modules that will be used in the generation of DRAGLIB are MODER, RECONR, BROADR, PURR (if dilutions present), THERMR, GROUPR and DRAGR. Figure 1 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.

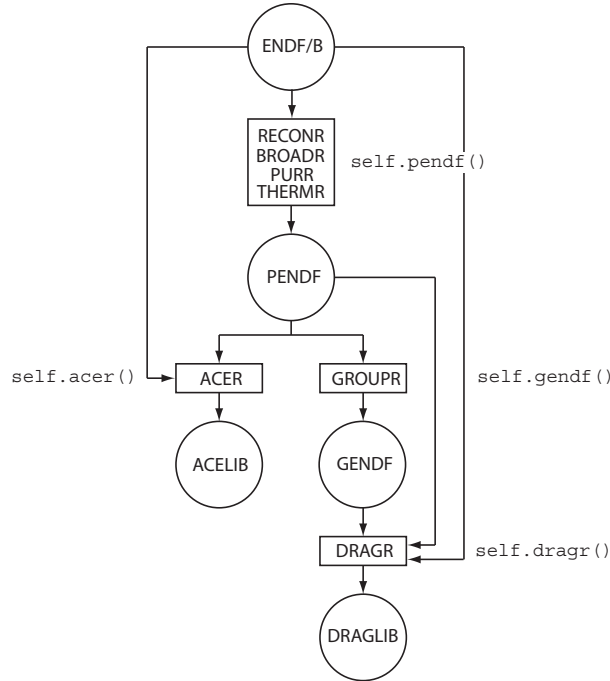


Figure 1: Flow chart for generation of ACELIB and DRAGLIB formatted libraries

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 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`. 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 and depicted in Fig. 2. 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.

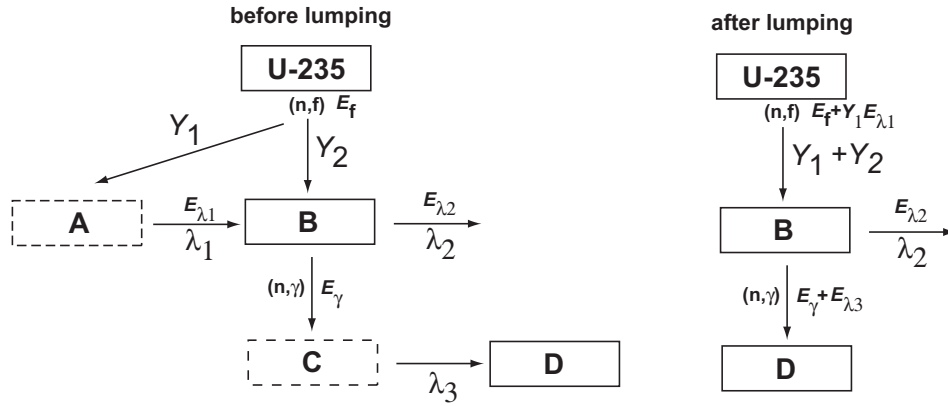


Figure 2: Lumping of isotopes A and C

Information on energies for various reaction types like (n, γ) , (n, f) , $(n, 2n)$, $(n, 3n)$, $(n, 4n)$, (n, α) , (n, p) , $(n, 2\alpha)$, (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.

1.3.2 Sequential calculations

Up to early 2019, DRAGLIB could be only computed in a sequential way, one isotope at the time, adding data to a formatted ASCII file. Examples of input file are provided with the distribution in the 'python' subfolder for different evaluations and energy structures. In this section, the main aspects of these files are illustrated.

Main object instantiation

The Python dataset starts with the instantiation of an object (named `candu` in the following examples) and with the definition of a few instance variables that are common to many isotopes. In this example, the PyNjoy script is supposed to be in the same folder as the script used to generate the DRAGLIB.

```
#!/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
```

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_D20"
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_D20.bcd.therm"
candu.scatteringMat = 11
candu.fission = None
candu.dilutions = None
candu.pendf()
candu.gendf()
candu.draglib()
```

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()
```

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()
```

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. Note that in an earlier version, the number of dilutions was limited to 10 for one computation. Thus, two calls were needed, as shown in the example below. However, this restriction does not exist anymore. Thus, in the example below, the user could replace the two dilution arrays by a longer one, and make only one call to the different modules.

```
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()
```

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()
```

1.3.3 Parallel calculations

Early 2019, parallel computing was introduced in the DRAGLIB generation as an initiative of J. Ortensi and reprogrammed by R. Chambon to make it user friendly and backward compatible with sequential input files. Now, calculations for all isotopes can be performed separately, and a final concatenation is performed for one isotope at the time, adding data to a formatted ASCII file or an unformatted SEQ_BINARY file. Examples of input file are provided with the distribution in the 'python' subfolder for different evaluations and energy structures, their name ends with '_pl'. In this section, the main aspects of these files are illustrated. The general idea is to set the calculations options and data for all isotopes, which define a 'job' for each isotope, and to store them all in an array. Then, all jobs are independent, and thus can be executed on several CPU if available.

A unique class named `PyNjoy_mp` contains the various instance variables and methods required to use NJOY in *multiprocessing mode*. The dispatching of the work is managed by a Python class named 'multiprocessing' and its subclass 'Pool' (see ¹). The structure of the input file is as follows:

- 1 Main instantiation
 - 1a Declare all common options
 - 1b Create a reference job
- 2 Create a job for each isotope as a modified copy of the reference job. Similar ones are grouped together:
 - 2a Isotopes with thermal evaluations: no dilution, scattering law
 - 2b Nominal isotopes: no dilution
 - 2c Fissionable isotopes: dilution, fission energy, ...
 - 2d Fission product: scattering matrix reduced to its diagonal. First without dilution (most of them), then with dilution
- 3 Run all jobs
- 4 Concatenate all individual DRAGLIB
- 5 Add depletion data

Main object instantiation

This section corresponds to step 1. The Python dataset starts the declaration of the common option such as computation folder, NJOY execution file path, evaluation files path,... and with the instantiation of an object (named `job_ref` in the following examples) with the definition of a few instance variables that are common to many isotopes. In this example, the `PyNjoy_mp` script is supposed to be in the same folder as the script used to generate the DRAGLIB.

```
#!/usr/bin/env python
from PyNjoy_mp import *
from os import uname
import os, sys, time
import copy
from collections import OrderedDict
# parallel distribution of the library generation
# Author: Richard Chambon - Ecole Polytechnique
# based on a modified version by Javier Ortensi - Idaho National laboratory
```

¹<https://docs.python.org/2/library/multiprocessing.html>

```
# of the workflow endfb7r1_shem361.py developed by EPM

#####
# file locations & options
#####
# folder name where computations are performed
evalName = "/tmp/shem361_endfb7r1_epm"
if not os.path.isdir(evalName): os.mkdir(evalName)
# relative path to the 'xnjox' code from current folder
execDir = "../" + uname()[0]
# evaluation folder paths
evalDir = "$HOME/evaluations/ENDFB7r1/neutrons/"
scatLawDir = "$HOME/evaluations/ENDFB7r1/thermal_scatt/"

#ncpu = 2
ncpu = mp.cpu_count()

job_ref = lib_base(evalName, execDir, evalDir, scatLawDir)
# General options
job_ref.nstr = 31 # SHEM-361 group structure
job_ref.eaf = 0
job_ref.temperatures = ( 293., 550., 900., 1200., 2000. )
job_ref.fission = None
job_ref.dilutions = None
# Specifics for output library format
job_ref.Espectra = None
job_ref.autolib = (22.53556, 1.11377e4, 0.0005)
job_ref.fp = 0
job_ref.concat = 1

njoy_jobs = VectPyNjoy()
njoy_jobs.setncpu()
```

Thermal scattering

This section corresponds to step 2a. An additional reference job ('job_ref_tsl') is defined to set options common to all isotopes with scattering law. Only the light and heavy water are presented in the example below.

```
#####
# Thermal scattering
#####
job_ref_tsl = copy.deepcopy(job_ref)
# General options
# Specifics for output library format

this_job = copy.deepcopy(job_ref_tsl)
this_job.legendre = 3
this_job.hmat = "H1_H2O"
this_job.mat = 125
this_job.evaluationFile = this_job.evaluationDir + "n-001_H_001.endf"
this_job.scatteringLaw = this_job.ScatteringLawDir + "tsl-HinH2O.endf"
this_job.scatteringMat = 1
```

```

this_job.temperatures = ( 293.6, 350.0, 400.0, 450.0, 500.0, 600.0 )
njoy_jobs.append(this_job)

this_job = copy.deepcopy(job_ref_tsl)
this_job.hmat = "H2_D20"
this_job.mat = 128
this_job.evaluationFile = this_job.evaluationDir + "n-001_H_002.endf"
this_job.scatteringLaw = this_job.ScatteringLawDir + "tsl-DinD20.endf"
this_job.scatteringMat = 11
this_job.temperatures = ( 293.6, 350.0, 400.0, 450.0, 500.0, 600.0 )
njoy_jobs.append(this_job)

print('Number of jobs from isotopes with thermal scattering: ' + str(len(njoy_jobs)))

```

Nominal isotopes

This section corresponds to step 2b. An additional reference job ('job_ref_nom') is defined to set options common to all nominal isotopes. Since only the ID and the evaluation file name change from an isotope to the other, a dictionary is used as shown below (only a part of it in this example) to define options. Then, the corresponding jobs are defined from it.

```

#####
# nominal isotopes
#####
c1 = len(njoy_jobs)
job_ref_nom = copy.deepcopy(job_ref)
# General options
# Specifics for output library format

iso_dict = OrderedDict()

iso_dict["H1"] = (125, "n-001_H_001.endf")
iso_dict["H2"] = (128, "n-001_H_002.endf")
iso_dict["H3"] = (131, "n-001_H_003.endf")
for iso_name, tup in iso_dict.items():
    this_job = copy.deepcopy(job_ref_nom)
    this_job.hmat = iso_name
    this_job.mat = tup[0]
    this_job.evaluationFile = this_job.evaluationDir + tup[1]
    njoy_jobs.append(this_job)

print('Number of jobs from nominal isotopes: ' + str(len(njoy_jobs) - c1))

```

Fissionable isotopes

This section corresponds to step 2c. An additional reference job ('job_ref_fiss') is defined to set options common to all fissionable isotopes. Only U235 and U238 are presented in the example below.

```

#####
# fissionable isotopes
#####
c1 = len(njoy_jobs)

```

```

job_ref_fiss = copy.deepcopy(job_ref)
# General options
# Specifics for output library format
job_ref_fiss.ss = (22.53556, 3.206464e5)

this_job = copy.deepcopy(job_ref_fiss)
this_job.hmat = "U235"
this_job.mat = 9228
this_job.evaluationFile = this_job.evaluationDir + "n-092_U_235.endf"
this_job.fission = 2 # fission with delayed neutrons
#this_job.ss = (22.53556, 3.206464e5)
this_job.potential = 11.6070
this_job.dilutions = ( 1.e10, 10000.0, 5957.50244, 3549.18335, 2114.42676, \
    1259.67004, 750.448669, 447.079956, 266.347961, 158.676849, 94.5317612, \
    56.3173141, 33.5510521, 19.9880447, 11.9078817, 7.09412289, 4.22632504, \
    2.51783395, 1.5 )
njoy_jobs.append(this_job)

this_job = copy.deepcopy(job_ref_fiss)
this_job.hmat = "U238"
this_job.mat = 9237
this_job.evaluationFile = this_job.evaluationDir + "n-092_U_238.endf"
this_job.fission = 2 # fission with delayed neutrons
#this_job.ss = (22.53556, 3.206464e5)
this_job.potential = 11.17103
this_job.dilutions = ( 1.e10, 10000.0, 5957.50244, 3549.18335, 2114.42676, \
    1259.67004, 750.448669, 447.079956, 266.347961, 158.676849, 94.5317612, \
    56.3173141, 33.5510521, 19.9880447, 11.9078817, 7.09412289, 4.22632504, \
    2.51783395, 1.5 )
njoy_jobs.append(this_job)

print('Number of jobs from fissionable isotopes: ' + str(len(njoy_jobs) - c1))

```

Fission products

This section corresponds to step 2d. An additional reference job ('job_ref_fiss') is defined to set options common to all fission product isotopes. Only few fission products without dilution and one with dilutions are presented in the example below. For fission isotopes without dilution, as for nominal isotopes, a dictionary is used to define the required data. Note that some isotopes without dilutions require additional data. In that case, they are defined independently as those with dilution; one is presented in the input below.

```

#####
# fission products
#####
c1 = len(njoy_jobs)
job_ref_fp = copy.deepcopy(job_ref)
# General options
job_ref_fp.legendre = 0
# Specifics for output library format
job_ref_fp.fp = 1

iso_dict = OrderedDict()

```

```

iso_dict["Ge72"] = (3231, "n-032_Ge_072.endf")
iso_dict["Ge73"] = (3234, "n-032_Ge_073.endf")
iso_dict["Ge74"] = (3237, "n-032_Ge_074.endf")

for iso_name, tup in iso_dict.items():
    this_job = copy.deepcopy(job_ref_fp)
    this_job.hmat = iso_name
    this_job.mat = tup[0]
    this_job.evaluationFile = this_job.evaluationDir + tup[1]
    njoy_jobs.append(this_job)

# fission product without dilution but additional options
this_job = copy.deepcopy(job_ref_fp)
this_job.hmat = "Cd114"
this_job.mat = 4849
this_job.evaluationFile = this_job.evaluationDir + "n-048_Cd_114.endf"
this_job.branchingNG = 0.079383
njoy_jobs.append(this_job)

# fp with dilution
job_ref_fp.ss = (22.53556, 1.858471e4)

this_job = copy.deepcopy(job_ref_fp)
this_job.hmat = "Zr90"
this_job.mat = 4025
this_job.evaluationFile = this_job.evaluationDir + "n-040_Zr_090.endf"
this_job.potential = 6.8813
this_job.dilutions = ( 1.e10, 10000.0, 3866.97, 1495.35, 578.2475, 223.6068, \
    86.4682, 33.4370, 12.9300, 5.0 )
this_job.autolib = (4.632489, 3.481068e3, 0.0005)
njoy_jobs.append(this_job)

print('Number of jobs from fission product isotopes: ' + str(len(njoy_jobs) - c1))

```

Isotopic DRAGLIB computation

This section corresponds to step 3. Before computations are performed, a test is performed to make sure that all required evaluation files are properly identified and present. In order to make the input files as much general as possible, the part where the standard NJOY output (pendf and gendf files) are computed is separated from the DRAGLIB computation. This way, if a user would like to produce library on an other format, only the last step need to be adapted.

The standard NJOY calculation is as follows:

```

#####
# check file locations
#####
for this_job in njoy_jobs:
    if not os.path.isfile(os.path.expandvars(this_job.evaluationFile)):
        raise PyNjoyError("evaluation file " + this_job.evaluationFile + " not found")
    if this_job.scatteringLaw != None:
        if not os.path.isfile(os.path.expandvars(this_job.scatteringLaw)):
            raise PyNjoyError("scatteringLaw file " + this_job.scatteringLaw + " not found")

```

```
njoy_jobs.pendf()
njoy_jobs.gendf()
```

Then comes the DRAGLIB specific instructions:

```
njoy_jobs.dendf()
njoy_jobs.dconcat()
```

DRAGLIB concatenation

This section corresponds to step 4. All individual DRAGLIB are concatenate one after the other. The corresponding input is presented in the previous section (last line).

Process burnup

For this step 5, 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.

```
this_job = copy.deepcopy(job_ref)
this_job.fissionFile = "$HOME/evaluations/ENDFB7r1/nfy/"
this_job.decayFile = "$HOME/evaluations/ENDFB7r1/decay/"
this_job.burnup()
```

1.3.4 CPU cost

To illustrate the usefulness of the multithread computing, a comparison is presented below for ENDF-B7.1 with the SHEM361 group structure and 321 nuclides. Table 1 presents the CPU time cost on a 24 processors computer (Intel(R) Xeon(R) CPU X5650 @ 2.67GHz).

Table 1: Complete DRAGLIB computation time

	Parallel 24 CPUs	Sequential 1 CPU
	2.34 h	33.2 h
details		
General options setting	0.1 s	119424s
All jobs definition	0.06 s	
All pendf and gendf	4261 s	
All draglib	4134 s	
All draglib concatenated	4 s	
Depletion	18.6 s	110.2 s

1.4 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 1 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 present 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 example provided here helps in generating ACELIB alone. In case a DRAGLIB file also is to be generated, please refer to Figure 2.

1.4.1 Heavy water

```
candu.hmat = "H2_D20"
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_D20.bcd.therm"
candu.scatteringMat = 11
candu.fission = None
candu.dilutions = None
candu.pendf()
candu.dirName = "D20-1"
candu.tempace = ( 293.6,)
candu.suff=0.20
candu.acer()
candu.dirName = "D20-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.dirName = "D20-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()
```

1.4.2 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.dirName = "Zr-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.dirName = "Zr-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()

```

1.4.3 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.dirName = "Xe135-2"
candu.tempace = ( 323.6,)
candu.suff=0.21
candu.acer()
candu.dirName = "Xe135-3"
candu.tempace = ( 573.6,)
candu.suff=0.22
candu.acer()

```

1.4.4 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.dirName = "U235-2"  
candu.tempace = ( 323.6,)   
candu.suff=0.21  
candu.acer()  
candu.dirName = "U235-3"  
candu.tempace = ( 573.6,)   
candu.suff=0.22  
candu.acer()
```

2 Format conversion of Draglibs

The early version of the PyNjoy.py script presented above could produce only an ascii-formatted (80-column) Draglib file in LCM/XSM export format. The current version can also produce binary. Dragon is expecting binary Draglibs, either in little- or big-endian direct-access XSM format. Since binary cannot be edited to look at values, both types may be useful. Using the Ganlib capabilities of Dragon, user can easily perform the required conversions.

The conversion from ascii-formatted to binary is presented below. The other way around can easily be programmed inspired by the example below.

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"
```