Notes on generation of DCHAIN's data libraries

Hunter N. Ratliff

April 2019 - August 2021

Contents

Introduction and Disclaimer

This document outlines the procedures employed in updating the nuclear data libraries used by the DCHAIN-PHITS code [1]. This document is an excerpt of my internal notes documentation I had composed while working on DCHAIN's development/modernization on the PHITS team. I am no longer employed on the PHITS team, but I have been given approval from Tastuhiko Sato leading the PHITS team to publish this document and its scripts.

While these were not originally intended to be published, these notes and scripts appear to be of interest to the PHITS community; therefore, I am sharing them now (April 2026). This document and the accompanying Python scripts are presented as-is. This work was conducted on a Windows 10 PC using contemporary Python versions of the time (namely, versions 3.7 to 3.9). While some of these scripts have been used since for a few one-off library updates, some have remained untouched since late 2019. These procedures and scripts are no longer actively maintained (at least not by this author) and are presented here more for the purposes of preservation of knowledge and utility to those engaging in similar efforts in the future.

Do note that while the data generation/formatting procedures remain true that some of the additional information here is outdated. This document was largely composed toward the beginning of much of my work on DCHAIN, before I had implemented a variety of additional features, such as being able to specify a data library file name in the DCHAIN input. So, keep in mind that anything presented in the PHITS manual, DCHAIN manual, or any lecture materials that conflict with this document are newer than the contents of this document and should supersede it.

1 Access to / formatting of nuclear data values

This section covers how to prepare nuclear data for DCHAIN [1]. Generally speaking, DCHAIN has its own custom format used for all nuclear data values. Most of these data values were originally obtained from data libraries in the ENDF format. The data files located in the `/data/` folder which need to be addressed in this work are listed below.

1. `fendla2_175-r4` is the neutron reaction cross section library for interactions with all isotopes. It's formatting is non-trivial and is discussed in Section 1.1.

2. `spd-dcylib` (and it's human-readable cousin, `spd-dcytxt`) contains all of the decay data for each nuclide (branching ratios, decay modes, decay energies, etc.). Its formatting, while specified in Reference 2, is non-trivial and is discussed in Section 1.2.

3. `lib_ec` contains the competing positron and electron capture decay data. The formatting is much more simple. This may or may not need to be something whose data is varied. Regardless, it will need to be generated to match the decay data library file; this is discussed in Section 1.3.

4. `dose_rate_coeff.dat` contains very simply formatted activity-to-effective-dose conversion factors in units of $(\mu Sv \cdot m^2)/(MBq \cdot hr)$ for every isotope discussed in more detail in Section 1.4.

While some data is hard-coded into DCHAIN, the above files are read into DCHAIN when executed. This has the important implication that, as long as the filenames and paths are the same, the values within each file can be varied without needing to recompile DCHAIN. This is the approach taken in this work.

However, if one of the core ideas of this work is to modify many values many times, a system for keeping all of these variations organized is clearly necessary. Now is an opportune time to discuss that organization. There are two high-level file locations to keep in mind here:

- The DCHAIN/PHITS installation data folder, `C:\phits\dchain-sp\data`
- The separate directory for modification and processing of data, `C:\work\JAEApostdoc\Induced_Activity_Project`

The PHITS installation directory should be kept as pristine as possible. The only time files will be moved here is when they are ready to be employed in a PHITS/DCHAIN simulation. Otherwise, all operations will take place in the "project" directory whose general structure is shown below.

```
Induced_Activity_Project/
 |- DCHAIN_data_formatting/
     |- DCHAIN_data_files
     |- ENDF-to-DCHAIN_decay_library_conversion/
         |- DCHAIN-formatted_libraries/
             |- jendlddf2015-pure
         |- ENDF-formatted_libraries/
             |- jendl-ddf-2015/
     |- ENDF-to-DCHAIN_xs_library_conversion/
         |- DCHAIN-formatted_libraries/
             |- jendl-ad2017_0K_fixed/
             |- jendlad2017-pure
         |- ENDF-formatted_libraries/
             |- jendl-ad2017_0K_fixed/
     |- ENDF-to-ACE?
```

The final data files to be used in DCHAIN are stored in the `/DCHAIN_data_files/` directory.

1.1 Conversion of ENDF cross section data to DCHAIN library format

To import nuclear data into DCHAIN, it is necessary to convert ENDF formatted files (like JENDL, ENDF, JEFF, TENDL, etc.) to the format used by DCHAIN. This section covers that process for cross section data. This section is based on an internal meeting which occurred on 2019/04/23 where this process was outlined and where I was provided with the following files:

Table 1: Files from meeting

File	Notes
`gendf-dchain.f`	Fortran script for converting GROUPIE output (in GENDF format) to a DCHAIN-ready format
`Os193.nt0.mod`	Updated ENDF-formatted file for ^{193}Os which is erroneous in the stock release of JENDL/AD-2017
`W187.nt0.mod`	Updated ENDF-formatted file for ^{187}W which is erroneous in the stock release of JENDL/AD-2017
`nb093/`	Folder of example files for ^{93}Nb

There are two common nuclear data formats: ENDF and ACE. ENDF is the format commonly distributed by nuclear data evaluators. ACE is the format used by MCNP and PHITS. DCHAIN uses neither of these formats and instead relies on its own custom format. Thus, a series of steps must be taken to convert ENDF files to a format compatible with DCHAIN. However, this also serves as a ripe opportunity to make any data modifications that may be desired since numerous tools exist for viewing and editing ENDF. NJOY is perhaps the most well known of these tools; unfortunately though, it apparently has some issues with MF9 in the ENDF files which will (likely) be important for this project. (I don't know if these issues exist in the newest NJOY, NJOY21, which is a complete overhaul of the old version, NJOY2016.) Fortunately, a suite of codes released by the IAEA, PREPRO 2018, is available to serve this purpose. (While I know NJOY can convert from ENDF to ACE, I do not know if PREPRO can as well; though, it is likely it can since the "Activate" code generates MF10 data, that used in the ACE format. Also, apparently results from "FIXUP", if outputted on a uniform energy grid, can be used as input for NJOY.)

The first library I will be working with is JENDL/AD-2017 which has been updated for activation cross sections important for decommissioning work. In total, the library contains 312 files. As noted in Table 1 though, the files for ^{193}Os and ^{187}W need to be replaced with the corrected ones provided. For this library, one can select either MF9 or MF10 and a temperature of 0K or 293.6K. The one which has T=0K libraries for MF9 was recommended to me. The JENDL page states that the MF10 versions primarily exist to be used in conjunction with NJOY to create ACE-formatted files. Also of note, this library contains all neutron-induced cross sections except for an additional file for proton-induced reactions with ^{56}Fe; this single proton file, whose extension is `.pt0` rather than `.nt0`, is simply ignored here. (Perhaps of note, the proton file can be processed through PREPRO but appears to not be outputted correctly from the Fortran script.)

The JEND library contains data files for each isotope. For this series of explanations, I will be using ^{93}Nb as an example isotope. So, the associated JENDL file will be called `nb093.nt0`. Concisely put, the steps to preparing ENDF files for DCHAIN are as follows:

1. Download and install/compile PREPRO 2018
2. Download the JENDL/AD-2017 library and replace the 2 erroneous files
    - (or other nuclear data library such as ENDF, TENDL, JEFF, etc.)
3. ~~Compile the Fortran script `gendf-dchain.f` → `gendf-dchain.exe`~~
    - ~~`ifort gendf-dchain.f -o gendf-dchain.exe`~~
4. For each ENDF file, convert to groupwise ENDF (GENDF) using the GROUPIE code of PREPRO and then format that for DCHAIN.
    (a) Select ENDF file to convert and move/copy it to a temp directory
    (b) Rename it from `nb093.nt0` to `ENDFB.IN`
    (c) With the `.INP` files for PREPRO provided by Konno-san present in the directory, run the gauntlet of codes in PREPRO as shown in the batch script below, yielding the GENDF file `GROUPIE.OUT`
    (d) Using my custom Python script gendf-2-dchain.py, convert GROUPIE's GENDF output to the DCHAIN format and move+rename the file to another directory.

(e) ~~Rename `GROUPIE.OUT` to `fort.50`~~

(f) ~~Run the compiled Fortran script `gendf-dchain.exe` which outputs `fort.60`~~

(g) ~~Rename `fort.60` to `nb093_nt0.dchainlib` and move the file elsewhere.~~

5. The new DCHAIN-formatted groupwise library is now prepared.

6. Combine the individual library files into a single large file for DCHAIN to use.

Batch script for running the `ENDFB.IN` file through PREPRO to yield `GROUPIE.OUT`

```
endf2c
linear
recent
del LINEAR.OUT
sigma1
del RECENT.OUT
activate
del SIGMA1.OUT
fixup
del ACTIVATE.OUT
dictin
del FIXUP.OUT
groupie
```

I have written a Python script, gen_bat_scripts_for_ENDF-to-DCHAIN.py, which automatically generates a single batch script which completes step 4 in the earlier enumerated list for an entire folder of ENDF-formatted files. After generating the batch script and running it, the Python script in Appendix check_ENDF_DCHAIN_folder_diff.py can be used to check if all files from the original database were successfully converted.

Noting the text which has been stricken out, I did eventually decide to replace the Fortran script for converting GROUPIE's GENDF output to the DCHAIN format with my own Python script, gendf-2-dchain.py. The Fortran script would crash when handling any other data library than JENDL/AD-2017, meaning another solution was necessary if I wanted to add any more neutron reaction cross section libraries to DCHAIN than just that. While developing my own Python script, I found several of the reasons that the Fortran script broke, but rather than fixing the Fortran script I will continue to use my own Python one since it is much easier to read, debug, and develop.

Additionally, the Fortran script makes a core assumption critical for composing the DCHAIN library which is simply untrue for a majority of the data libraries, that every MF3 reaction MT had a corresponding MT entry in MF8 from which the reaction products could be identified. While my script will still prioritize taking information from MT entries in MF8 where available, by default my code will determine the product's identity (atomic number and mass) from the reaction taking place. The only downside is that one cannot determine whether the product is in its ground state or a metastable one, but this is still preferable to not being able to identify the product whatsoever. I have coded in this for all reactions with MT under 120.

DCHAIN uses one big file (as opposed to individual files for each isotope) at the beginning of the code. The ordering of isotopes and reactions in this file apparently does not matter. Two strategies exist for this project when modifying nuclear data:

1. Edit DCHAIN to rely on many small files for nuclides, meaning only individual files for specific isotopes must be updated when varying nuclear data.

2. Recompile the single large data file used by DCHAIN for every variation employing a modified data value.

Given DCHAIN's library size is comparatively small and assembly of the single large data file is quite rapid in comparison to the actual simulations being ran, adapting the code to instead use individual small data files was deemed to be an unwise use of time. The script for combining all of the individual DCHAIN library files into a single large library file, compile_DCHAIN_data_lib.py, can be found in Appendix A.4.

While it is impossible to avoid generating numerous large data library files with significant overlap with this approach, it would be good to avoid unnecessary duplication where possible.

After the new library file is generated, DCHAIN needs to know to use it. This can be done by renaming it to have the same name as the existing cross section library file and replacing the original one or by changing the filename used in the source code pointing to this library file. The former option poses a more challenging organizational issue while the later requires recompiling DCHAIN with every iteration / variation of data tested. Since the end goal will be to have this entire process handled by a batch script anyways, the perhaps best approach is the one which least impacts execution time. In this case, it means avoiding recompiling the code.

So, completed cross section data library files will be kept in a separate directory with descriptive titles. To use them in a simulation, they will be copied to the `/dchain-sp/data/` folder and renamed to `fendla2_175-r4` (the file name the release version of DCHAIN packaged with PHITS points to), overwriting the previous version of the file. As a backup and diagnostic measure, it would also be a good idea to have a file somewhere which states the descriptions for the current files in the `/data/` folder.

Thus, at a very high level and using the codes developed here, the process of preparing an ENDF library for DCHAIN is as follows:

1. Download the desired ENDF-formatted data library

2. Move the ENDF-formatted library to a desired location

3. Update this path and the output path in gen_bat_scripts_for_ENDF-to-DCHAIN.py

4. Run gen_bat_scripts_for_ENDF-to-DCHAIN.py, which generates the batch script convert_ENDF_<*library-name*>_to_DCHAIN_lib.bat

5. Run convert_ENDF_<*library-name*>_to_DCHAIN_lib.bat in its directory

6. If desired, run check_ENDF_DCHAIN_folder_diff.py to verify the batch script successfully converted all of the original ENDF files to the DCHAIN-format

7. Run compile_DCHAIN_data_lib.py to condense the many DCHAIN library files into the single file used by DCHAIN.

1.2 Conversion of ENDF decay data to DCHAIN library format

The decay data used comes from JENDL/DDF-2015 or the JENDL Decay Data File 2015. Section 2 details use of ENDF/B-VIII.0 libraries instead and what (minor) differences exist between use of JENDL and ENDF-B decay libraries.

Before me, there was written a Python script (not so originally titled `main.py`) which automatically converted ENDF-formatted decay data to the format used by DCHAIN. It appears to be written in Python 2.7, so I have converted it to Python 3 syntax using *2to3*, naming the updated version convert_decay_data_ENDF-to-DCHAIN.py. While my general style is to handle all file writing within Python, the old script simply prints all of the desired output and relies on the user to redirect the output in the terminal to a file as shown below. Note that it is useful here to give the output file a descriptive name; it only needs to be titled "`spd-dcylib`" when being used by DCHAIN.

```
python convert_decay_data_ENDF-to-DCHAIN.py > OUTPUT_FILENAME
```

To maximize convenience, this script should be kept in the directory which contains other directories of ENDF decay data files. The specific name of the directory (located in the same folder as the Python script) which contains the ENDF files to be converted must be specified within the Python script. Also, unless the output file's path is also specified, this output text file will be generated in this same folder too. Relative paths can be used as shown below.

```
python convert_decay_data_ENDF-to-DCHAIN.py > ../..
    /DCHAIN_data_files/jendlddf2015-pure
```

Given that I am essentially treating the old code a bit like a black box for this purpose, similarly to how I treated the other Fortran code and the PREPRO package, it makes sense to make a single short batch script which can be used to automate this process a bit more easily. To do this, I first modified the Python script to accept specification of the directory name containing the ENDF files when executed via command line. Then, I wrote the simple batch script below, titled `run_decay_conversion_script_ENDF-to-DCHAIN.bat`, which calls the Python script, passing to it the first argument (`%1`) provided when invoking the batch script, and then placing the output in the DCHAIN-formatted results folder with a filename specified by the second argument (`%2`) provided to the batch script.

```
python convert_decay_data_ENDF-to-DCHAIN.py %1 > ../..
    /DCHAIN_data_files/%2
```

run_decay_conversion_script_ENDF-to-DCHAIN.bat

So, to convert the ENDF files in the `ENDF-formatted_libraries/jendl-ddf-2015/` folder into a single file called `jendlddf2015-pure` located in the `DCHAIN-formatted_libraries/` folder, the batch script would be invoked as shown below.

```
run_decay_conversion_script_ENDF-to-DCHAIN.bat jendl-ddf-2015
    jendlddf2015-pure
```

1.3 Conversion of ENDF $\beta+/E.C.$ ratios to DCHAIN library format

Unlike the previous two sections, preexisting codes to extract this information from ENDF files are not available to me. Thus, I must develop my own code to perform this action. This process has ended up being quite more involved than originally intended.

First, it seems that the JENDL DDF library uses data taken from the Evaluated Nuclear Structure Data Files (ENSDF), a 2010 version to be specific. From the ENSDF archive repository, one can find much more recent releases of the ENSDF library. Additionally, without delving too deeply into the specifics of the ENDF format, there are actually two pieces of information in the ENDF-formatted JENDL files which can yield information on the positron intensities: the positron emission intensities themselves and the 511 keV annihilation photon intensities produced from those positrons annihilating.

One may think using the ENSDF files is the most obvious route in order to have the most up-to-date data available; however, it does not come without its own challenges. First, it has its own file formatting syntax substantially different from ENDF. Additionally, all decays are listed in terms of their daughter nuclides. For instance, information on the $\beta+/E.C.$ decay of ^{18}F is found under the section for ^{18}O (the daughter product) in the "P" *parent record*. This can complicate searching for a specific decay. Due to this complication and specifics of the ENSDF format itself, the parent in each decay does not have its isomeric/metastable state listed; only its half life, spin, parity, ground state Q value, and energy of the decaying level are reported. This leaves it to the individual to attempt to define which decays are associated with which isomeric states of a given isotope.

Despite these troubles though, the ENSDF data is usually extremely reliable and consistent with NNDC's decay information (likely because it pulls from a recent ENSDF release). Unfortunately though, one of the key constraints of this problem is that there must be $\beta+/E.C.$ ratio data for *every* isotope in the decay library, which, at least initially in this project, is from JENDL and consists of 3237 isotopes/isomers, 1286 of which undergo some sort of positron emission or electron capture. After parsing the ENSDF library, positron decay intensity information (found on the "E" $\beta+/E.C.$ *record*) was found to only be available for 864 of these 1286 decays. ENSDF does mention quite a few more cases of $\beta+/E.C.$ decays as indicated by the *identification records*, but for those cases only the *parent record* is present, meaning a Q value is known while positron intensities are unknown.

Thus, some decay information must be extracted from the ENDF-formatted JENDL files. Parsing the ENDF format is quite a bit less straightforward than the ENSDF format, though both are reasonably well-documented. So, within the massive standard that is the ENDF format, the only information of concern here is contained in MF=8 MT=457, the section specifically for radioactive decay. Unlike ENSDF, the ENDF files are organized by parent nuclide, with one file per isotope/isomer. So, one can parse the file for a given nuclide and check it's RTYP (decay mode) entries to see if any electron capture or positron emissions occur. If so, the file can be further searched for decay radiation information (denoted by STYP). For positron emission intensity, one can either sum the intensities of all emitted positrons (STYP=2.0) or half the total 511 keV annihilation photon intensity (STYP=9.0).

Unfortunately though, this process is not necessarily so straightforward. I will not go into the minutiae here, but do be mindful of the branching ration BR, normalization

factor FD, intensity of radiation produced RI, intensity of positrons produced RIS (the E.C. case is an exception for this card), and discrete energy of radiation produced E. Essentially, all intensities (RI and RIS) must be multiplied by the normalization factor FD to obtain the average number of radiations/positrons produced from each decay in that given branch. That value must then be divided by the branching ratio of that decay channel (RTYP, e.g. 2.0, 2.4) to obtain the total number of radiations/positrons emitted per decay of the parent. While that handles normalization, that is not the only complication.

All of the $\beta+/E.C.$ decay radiation energies listed, E, are actually the full decay energy available. For emitted positrons, this is the end-point energy plus the threshold energy, two electron masses or 1.022 MeV as shown in Equation 1.

$$E = E_{end-point} + 2m_e = E_{end-point} + 1.022 \text{ MeV} \tag{1}$$

As an example, the ENDF file for the decay of ^{18}F lists the prominent (96.73% intensity) positron emission as having $E = 1.6555$ MeV. Subtracting 1.022 MeV yields 633.5 keV, which is the end-point energy of that positron as listed on NNDC. Unfortunately, this does present some issues, some of which I believe result in errors present in the current JENDL DDF library.

Under the section for STYP=2. emissions ($E.C./\beta+$), energies below 1.022 MeV are listed. While the energies above 1.022 MeV can be assumed to be actually emitted positrons, all of these listed emissions appear to impact the listed 511 keV annihilation photon intensities, sometimes causing erroneous values. For instance, for ^{52}Mn, two prominent emissions are listed: 1.5976 MeV ($I_{\beta+} = 29.6\%$) and 84.4 keV ($I_{\beta+} = 96.86\%$). The first one is characteristic of the only positron emitted (since $E > 1.022$ MeV) while the second is not listed on ENDF whatsoever (though is likely related on an E.C. decay). Regardless, the file's stated in 511 keV annihilation photon intensity is stated to be 252.9%, double the sum of these two intensities. This is, however, not the correct annihilation photon intensity. Since only the higher energy of those two emissions is actually associated with a photon intensity, the actual annihilation photon intensity should only be 59.2%. While this issue may be resolved in a future JENDL release, one should be aware that this issue may still exist.

Anyways, aside from these errors and a few miscellaneous places where JENDL appears to have missing, extraneous, or faulty data values, there is one more hurdle: There are still cases where intensity values are not stated. At present, in DCHAIN if a decay does not have intensity information present but does have a Q-value (which is present for all JENDL and ENSDF decays), the positron emission fraction is estimated from the decay's Q-value, Q_i, and the parent's mass A using the functions listed in Equations 2 and 3 (executed in the `gammsp.f` module).

$$Q_{th}(MeV) = 1.022 + (5 \times 10^{-3}) \cdot A \tag{2}$$

$$I_{\beta+} = \begin{cases} 0.0 & Q_i < Q_{th} \\ \dfrac{Q_i}{2 \cdot Q_{th}} - \dfrac{1}{2} & Q_{th} < Q_i < 2 \cdot Q_{th} \\ \dfrac{Q_i - 2 \cdot Q_{th}}{10 - 2 \cdot Q_{th}} & Q_i > 2 \cdot Q_{th} \end{cases} \tag{3}$$

Rather than leaving this to DCHAIN, I have opted here to incorporate this calculation into the scripts generating the EC ratio library files to be used by DCHAIN, meaning there will be no isotopes missing positron emission fractions in those library files.

Figure 1 shows the relationship between positron emission probability for a given $\beta+/E.C.$ decay, Q, and A. The blue line indicates the line below which all decays are exclusively 100% electron capture (0% positron emission); note that there are some 100% E.C. decays lying near but above this line. All points above the red line have a 99% positron emission probability or greater. Both of these lines were empirically generated.

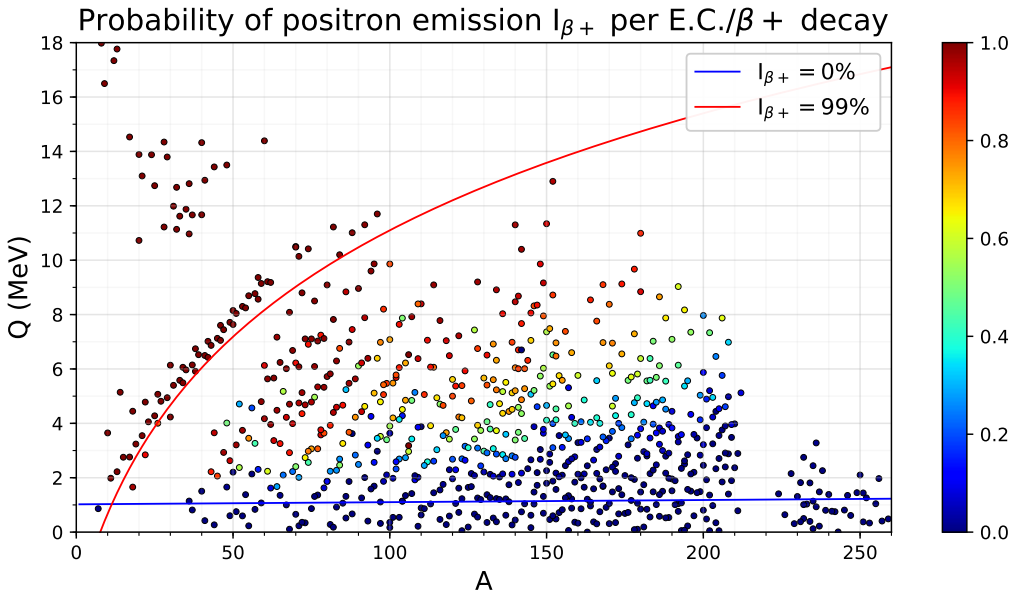


Figure 1: Relationship between positron emission probability $I_{\beta+}$, A, and Q

To test the validity of these functions currently used within DCHAIN to estimate missing positron emission intensities from Q and A, a colormap of Equations 2 and 3 applied over this phase space can be overlaid on Figure 1, producing Figure 2.

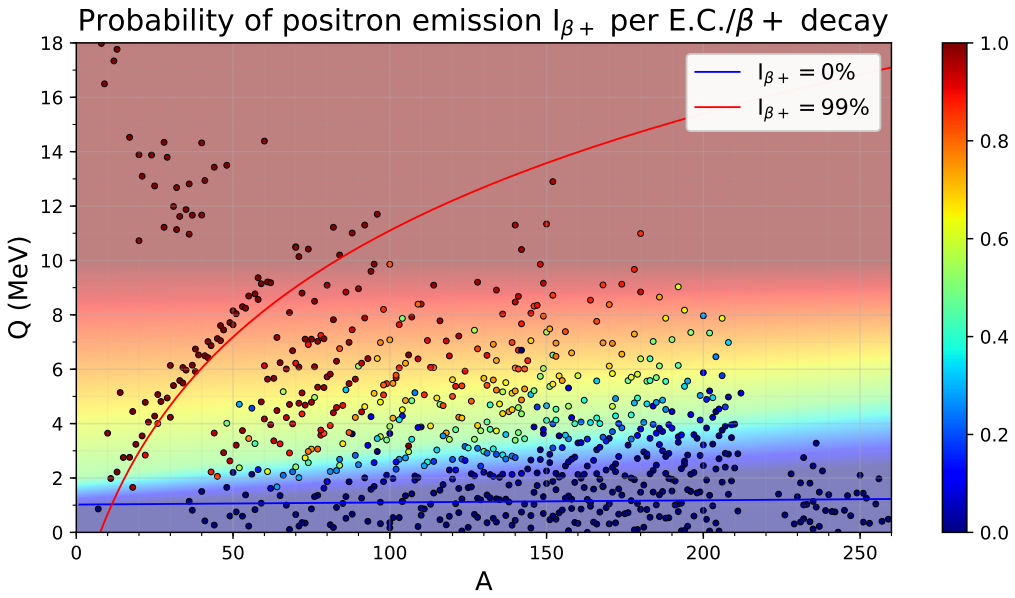


Figure 2: Same as Figure 1 but with DCHAIN's $I_{\beta+}$ estimation function overlaid

From this exercise we see that this function, while not terrible, is certainly lacking in some places. It performs best for decays where electron capture is favored over positron emission. For heavier nuclides, it seems to overestimate $I_{\beta+}$ with Q values above a few MeV. It does an especially poor job of capturing heavily favored positron emissions from the lighter nuclides.

An attempt has been made to generate a new empirical function that more accurately predicts the existing data. Provided Q and A should be the variables which impact $I_{\beta+}$, one must note that the impact of Q on $I_{\beta+}$ is much more significant than that of A. Thus, the function decided on was a logistic function with Q as the primary variable (Equation 4), but the other coefficients within the main function have light dependencies on A (Equations 5 through 7). Since the change in $I_{\beta+}$ appeared more steep for decays of lower Q-values, a multiplier on the steepness of the logistic curve was adopted for decays of lower Q-value. The fitting parameters in these equations, denoted as c with subscripts, are tabulated in Table 2 and were optimized to minimize deviation with known values of $I_{\beta+}$, first manually/visually and then refined using the GRG Nonlinear Solver in Microsoft Excel (with Multistart enabled).

$$I_{\beta+} = \begin{cases} 0.0 & Q_i < Q_{th} \\ \left(1 + e^{-k\cdot(Q_i - Q_x)}\right)^{-1} & Q_i > Q_{th} \end{cases} \tag{4}$$

where:

$$Q_x = c_{x1} \cdot A^2 - c_{x2} \cdot A + c_{x3} \tag{5}$$

$$k = c_{k1} \cdot m \cdot A^{-c_{k2}} \qquad \text{if } k > 5, \text{ then } k = 5 \tag{6}$$

$$m = \begin{cases} c_{m1} \cdot A + c_{m2} & Q_i < Q_x \\ 1.0 & Q_i > Q_x \end{cases} \tag{7}$$

Table 2: New $I_{\beta+}$ model parameters

Variable	Value
c_{x1}	10^{-4}
c_{x2}	2.14×10^{-4}
c_{x3}	2.2216
c_{k1}	168.18
c_{k2}	1.146
c_{m1}	2.94×10^{-3}
c_{m2}	2.818

The resulting new empirical function overlaid with the existing data is shown in Figure 3. While it is not perfect, it does appear to be a clear improvement from the previous function. The boundaries where either decay mode become overwhelming more probable than the other align much more closely with the data in this model. It generally leans toward overestimating $I_{\beta+}$, which is the more desirable direction for it to err given its ultimate impact is with the number of annihilation photons that will be generated, providing a more conservative estimate. An interesting scientific pursuit would be to develop a physics-based model describing this phenomenon, but this empirical function as-is is more than suitable for its intended use case here. The script used to generate these plots and perform this troubleshooting can be found in Appendix A.10.

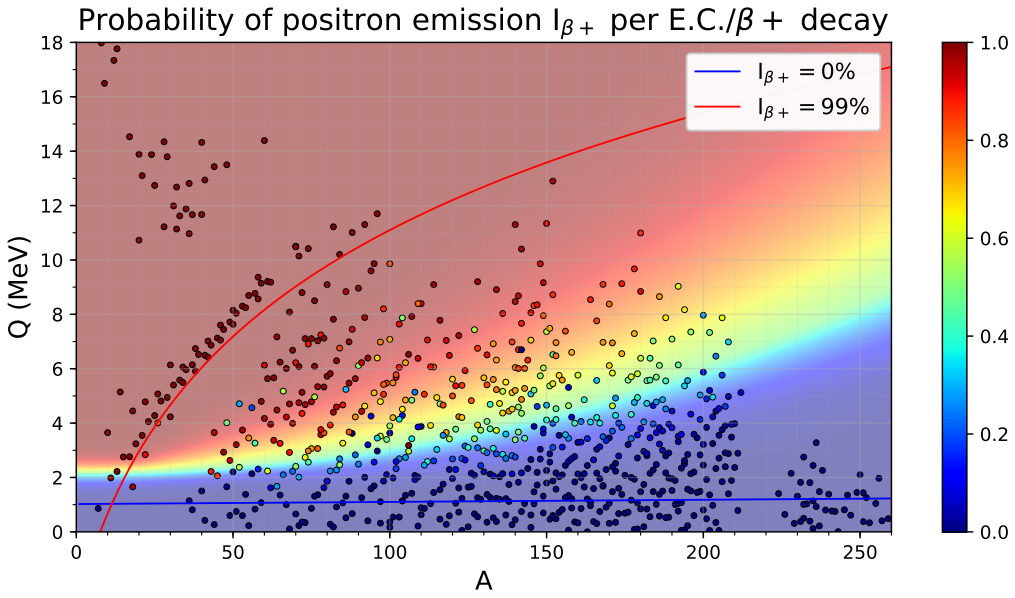
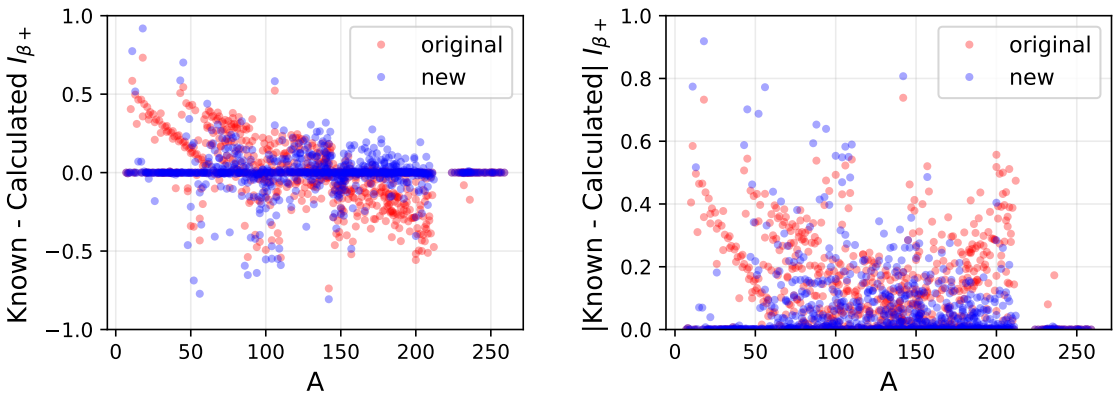


Figure 3: Same as Figure 1 but with the new $I_{\beta+}$ estimation function overlaid

To quantify the improvement of this function, comparison of known and calculated values is warranted. Figure 4 shows the differences of known and calculated values from both models. It is clear that there are systematic issues in the original model while deviations in the new model appear more random, though more concentrated at low-to-intermediate masses.



(a) Known - Calculated

(b) |Known - Calculated|

Figure 4: For data values with known positron emission fractions/rates $I_{\beta+}$, the differences (normal and absolute) between the known and calculated values are shown for both the new and old models for predicting $I_{\beta+}$.

The findings from these plots are quantified in Table 3. While looking at the mean and median differences when just subtracting the calculated value from the known one doesn't really paint a clear picture, looking at these values for the absolute differences makes a much stronger case for the new model. Since calculated values can deviate above and below the known value, both models "on average" do a good job of predicting $I_{\beta+}$. However, when considering this absolute value, or the magnitude of the deviations, the issues from the systematic trends in the old model become much more evident. Use of the new model better prevents inaccurate results while the old model is more likely to produce results systematically under or over predicting the positron emission rates (and, thus, annihilation photons).

Table 3: Assessing model performance: differences between known and predicted positron emission rates $I_{\beta+}$

	Known - Calculated		|Known - Calculated|	
	Old model	New model	Old model	New model
Mean	-0.73%	0.0011%	12.39%	6.74%
Median	0.00%	0.00%	7.32%	1.27%

1.4 Assembly of DCHAIN's activity-to-dose ratio database

In the older version of DCHAIN, the `dose_rate_coeff.dat` file is composed of a row for each nuclide present in the decay database containing the nuclide name and a number next to it. This number was a fluence-to-dose conversion factor with units of $\mu Sv \cdot m^2 \cdot MBq^{-1} \cdot hr^{-1}$ Modern fluence-to-dose conversion factors have units of effective dose times area ($pSv \cdot cm^2$), and since $Bq = \#/s$, the time units in the denominator can be added or removed at will. The meaning or derivation process of the old conversion coefficients is not clear, so I decided to add some additional conversion coefficients and then allow the user to select which coefficients they would like to use. While all of the ICRP coefficients are in units of just $pSv \cdot cm^2$, it seems that $\mu Sv \cdot m^2 \cdot MBq^{-1} \cdot hr^{-1}$ are actually the

standard units for this quantity in Japan. Thus, it makes more sense to convert all of the ICRP values to these units. This has the added benefit of not needing to modify the old values or the source code's math. Thus, to convert the ICRP 116 photon fluence to effective dose conversion coefficients in $pSv \cdot cm^2$ to DCHAIN's $\mu Sv \cdot m^2 \cdot MBq^{-1} \cdot hr^{-1}$, the unit conversions shown below in Equation 8 must be employed.

$$\frac{\mu Sv \cdot m^2}{MBq \cdot hr} = \frac{pSv \cdot cm^2}{\text{decay}} \cdot \left(\frac{10^{-6}\mu Sv}{1pSv}\right)\left(\frac{10^{-4}m^2}{1cm^2}\right)\left(\frac{1\frac{\text{decay}}{s}}{1Bq}\right)\left(\frac{10^6 Bq}{1MBq}\right)\left(\frac{3600s}{1hr}\right) \quad (8)$$

Then, to obtain the final dose results, which are in units of $\mu Sv \cdot m^2 \cdot hr^{-1}$, DCHAIN just multiplies the activities of each nuclide (converted from Bq to MBq) by these conversion coefficients.

The script used to generate this file can be found in Appendix A.11. In the 2019 release of DCHAIN, this file had just been the original database plus missing nuclides with values of zero. In the current one, seven additional fluence-to-dose conversion coefficients have been added from which the user may choose. The first six are photon fluence to effective dose E conversion coefficients published in ICRP 116 [3] for the various standard geometries: AP, antero-posterior; PA, postero-anterior; LLAT, left lateral; RLAT, right lateral; ROT, rotational; and ISO, isotropic. The seventh is fluence to ambient dose equivalent $H^*(10)$ conversion coefficients from ICRP 74 [4] for photons 10 MeV and lower and from Pelliccioni [5] above 10 MeV. (Note that there is a discontinuity at this transition.)

To generate the data for this file, one must evaluate the per decay fluence to dose conversion coefficients for each nuclide. This is done using the decay data libraries used by DCHAIN which already contain the most comprehensive gamma emission databases available. For nuclides where no discrete gamma rays are listed, the dose conversion coefficient is evaluated from the average gamma emission energy. For nuclides with discrete spectra listed, dose conversion coefficients are found for each gamma emission energy and then summed, weighted by their individual intensities (average number emitted per decay) to obtain the total dose conversion coefficient per decay. Since the ICRP's minimum photon energy is 10 keV, all photons below this threshold are ignored. Following the guidance of ICRP 116, a cubic interpolation of the tabulated conversion coefficients is performed on a log-log scale. In this code, splines are used instead of Lagrangian polynomials.

As for the specific method of implementation, I have opted to just add new columns to the `dose_rate_coeff.dat` file. Now, the columns are as follows: Nuclide, old coefficient, AP, PA, LLAT, RLAT, ROT, ISO, $H^*(10)$, and Source (specifying library of origin for this data). In `rddrlb`, the code just reads in the nuclide and coefficient from each row separately, specifying what column numbers should be read. Thus, this can be easily updated to accommodate the new file by adding an input parameter allowing the user to select what set of coefficients should be used which then just changes the columns which will be read for the coefficient value. In fact, the `rddrlb` code already seems to have code built for selecting an arbitrary column number built into it; it had just been hard-coded to stop after reading the second column.

A new input variable `IDOSECF` has been added to control what dose rate coefficients are used. `IDOSECF = 0` employs the old defaults while the consecutive integer values 1 through 7 correspond to AP, PA, LLAT, RLAT, ROT, ISO, and $H^*(10)$, respectively.

1.5 Automation of the decay data library compilation

If any of the decay data files are added/removed/changed, the three files reliant on decay data will need to be updated: `lib_ec`, `spd-dcylib` (and it's human readable sibling, `spd-dcytxt`, if desired), and `dose_rate_coeff.dat`. The previous three sections have covered the generation of each of these files independently, but, in practicality, it makes sense to combine these processes into a single automated step to be executed. [Does it though? Perhaps if this is a frequent occurrence, but at present carefully updating each file manually sounds like the better approach. If this does become more routine and my procedure for assembling these libraries is clearly something that can be generalized and automated, then this avenue will be worth pursuing.]

Below listed are the steps involved in preparing the decay data files.

1. In the terminal (and in the File Explorer if desired), navigate to the directory: `\Induced_Activity_Project\DCHAIN_data_formatting\ENDF-to-DCHAIN_decay_library_conversion\ENDF-formatted_libraries`

2. Make sure there is a folder, `<endf-decay-lib-folder-name>`, present containing the ENDF decay data files to be processed.

3. To generate the updated decay library file (`spd-dcylib`), execute the following command in the terminal: `run_decay_conversion_script_ENDF-to-DCHAIN.bat <endf-decay-lib-folder-name> <output-dchain-decay-lib-file-name>`

   - The output file, `<output-dchain-decay-lib-file-name>`, will be placed into the folder: `\Induced_Activity_Project\DCHAIN_data_formatting\DCHAIN_data_files`
   - **Note:** All output files generated here are saved to the above location.
   - more details on this process can be found in Section 1.2.

4. To generate the human-readable version of the decay library, `spd-dcytxt`, (and a NumPy array of all the nuclides present in the library used by later codes) and list which nuclides are exclusive to the current decay database versus those exclusive to the library that is shipped with DCHAIN in PHITS v3.10, run the parse_DCHAIN_decay_data_file.py script, making sure the filename of the decay library to be parsed, `<output-dchain-decay-lib-file-name>`, is correctly specified in the Python code.

5. The generation of the $\beta+/E.C.$ ratio library file, `lib_ec`, is slightly more complicated. Briefly, it involves cobbling together data for these ratios from multiple sources.

   - Navigating to the `Induced_Activity_Project\DCHAIN_data_formatting\ENDF-to-DCHAIN_decay_library_conversion` directory, there should be subdirectories for ENDF and ENSDF formatted files. Within each of those directories, one can place folders containing libraries of decay data in each of those respective formats.
   - To extract intensities, where available, from an ENDF library and calculate missing intensities from the nuclide's A and the decay's Q-value, run the parse_ENDF_files_for_EC_ratios.py script, making sure the `ENDF_folder_path`

variable is set correctly to the desired location. Additionally, changing of the output ENDF EC ratio library's filename at the end of the script may be desired.

  - Note: this code can be ran in two modes, one which sums positron intensities and another which sums annihilation photon intensities. The positron mode is generally slightly more reliable than the annihilation photon mode, but both can be ran without creating overlapping files. Just make sure that the `search_mode` is set to which ever is desired, and run the code twice, once in each mode, if desired.

- To extract only explicitly listed intensities from an ENSDF library, run the parse_ENSDF_files_for_EC_ratios.py script, ensuring the path to and name of the desired ENSDF library directory is correctly specified in the Python code. Additionally, changing of the output ENSDF EC ratio library's filename at the end of the script may be desired.
- To get the best of both worlds, higher reliability of ENSDF results and completeness of ENDF results (given that the ENDF decay library is, definitionally, complete for this application), the two can be pulled from to form a final hybrid EC ratio library. This is done by running the compare_EC_lib_methods.py script. Make sure the file names of the EC libraries generated from the ENDF and/or ENSDF libraries in the scripts mentioned previously are correctly specified in the `new_ec_lib_fnames` variable. Then, using the `dataset_priorities` variable, rank the priority of when to use each library. Note that if an ENDF-generated file is listed first, it will likely just pull exclusively from it since it will contain all of the nuclides present in the decay library. Generally, the ENSDF library should be ranked first.
- This process results in a $\beta+/E.C.$ ratio library file, `lib_ec`, containing as much accurate data as possible.

6. Finally, an updated dose rate coefficient library can be constructed by running the generate_dose_rate_coeff_lib.py script.

2 Relevant ENDF/B-VIII.0 libraries

While this work initially sought to use the most recent JENDL libraries, the recently released (Feb. 2018) ENDF/B-VIII.0 libraries should not be ignored. Specifically relevant to this work, ENDF/B-VIII.0 contains specific sublibraries for decay and neutron reactions. While conversion of these files to their DCHAIN-formatted counterparts was relatively straightforward, there were a few minor roadbumps encountered.

While processing the decay data files using the convert_decay_data_ENDF-to-DCHAIN.py script as discussed in Section 1.2, several hurdles were encountered. First, the ENDF/B-VIII.0 decay library contains nearly every isotope ever measured, meaning the arrays containing lists of element symbols had to be updated to up to Z=111 (the maximum in ENDF/B-VIII.0). Additionally, these ENDF files appear to be missing the sequence number NS located at the end of each line. While the ENDF-6 format manual explicitly states that this value is a relic of the past when data was stored on physical paper cards, it does not mention that its inclusion is optional, which makes its exclusion from the ENDF/B-VIII.0 library somewhat perplexing. Regardless, since it is a relic of the past,

it is not that important. While the convert_decay_data_ENDF-to-DCHAIN.py script does indeed attempt to read the NS values, it does not actually do anything with them. Therefore, every attempt to read columns 76-80 for the NS value in the script has been placed inside a try/except statement as to allow their reading when present but not to crash the code when absent.

As an additional complication, for stable nuclides, the JENDL library simply excludes the MF=8 MT=457 portion of the files while the ENDF/B library includes the section but states the nuclides have half-lives of 0.0 and leaves all other fields filled with zeros as shown below for ^{14}N. There was a minor bug in the old Python 2.7 script which would cause stable nuclides to have the information of the previous radioactive nuclide associated with it, but that issue was easily resolved.

```
 7.014000+3 1.388278+1          0          0          1          0  71 8457
 0.000000+0 0.000000+0          0          0          6          0  71 8457
 0.000000+0 0.000000+0 0.000000+0 0.000000+0 0.000000+0 0.000000+0  71 8457
 1.000000+0 1.000000+0          0          0          6          0  71 8457
 0.000000+0 0.000000+0 0.000000+0 0.000000+0 0.000000+0 0.000000+0  71 8457
```

ENDF/B-VIII.0 MT=457 section for N-14

3 Construction of composite libraries

While Section 1 detailed conversion of libraries from a given source into formats accepted by DCHAIN, it did not cover preparation of what will ultimately be the final libraries shipped with the updated DCHAIN. Since the goal, ultimately, is to provide the best libraries possible, limiting oneself to a single source is counterproductive. Instead, it is beneficial to combine information from multiple libraries into a composite library, meaning no nuclides will be excluded from the final product.

The presently available/processed decay libraries are listed below.

- Old DCHAIN decay library
- JENDL/DDF-2015 decay library
- ENDF/B-VIII.0 decay library

In addition to these, the following decay libraries and/or methodologies were used to generate electron capture to positron emission ratios:

- Old DCHAIN $\beta+/E.C.$ library
- ENSDF-extracted positron emission intensities
- JENDL/DDF-2015 positron emission intensities
- JENDL/DDF-2015 annihilation photon intensities
- ENDF/B-VIII.0 positron emission intensities
- ENDF/B-VIII.0 annihilation photon intensities
- Calculation from Q and A as shown in Section 1.3.

The script decay_lib_master_assembly.py was written to allow generation of composite versions of the decay-related library files used by DCHAIN quickly and easily. It allows the user to combine the previously listed libraries while selecting which of them to use and how to rank their usage (in other words, which libraries have higher priority for having values placed into the final library).

The `spd-dcylib`, `spd-dcytxt`, and `dose_rate_coeff.dat` files consist of data for all nuclides present, regardless of stability or decay mode. When assembling these files, the user can rank their preference of decay libraries to be used. If data for a given nuclide is not available in the most preferred library, the code will check the next preferred library, and so on. Below are the configurations considered with the present decay data libraries and the string of characters denoting that ranking.

- `[EJO]` - ENDF/B-VIII.0 > JENDL/DDF-2015 > Old-DCHAIN-lib
- `[JEO]` - JENDL/DDF-2015 > ENDF/B-VIII.0 > Old-DCHAIN-lib

The `lib_ec` and (new) `lib_ec_with_sources` files are assembled from the provided $\beta+/E.C.$ libraries. Due to their generally lower reliability, the libraries generated from the 511 keV annihilation photon emissions have been excluded from these libraries. Additionally, the user can decide on whether ratios calculated using the methodologies outlined in Section 1.3 should be lower priority or higher priority that their non-first-choice libraries. Below are the considered combinations for these $\beta + /E.C.$ libraries and the character strings denoting the specific rank ordering.

- `[ESJO]` - ENDF/B-VIII.0$^{(\beta+)}$ > ENSDF > JENDL/DDF-2015$^{(\beta+)}$ > Old-DCHAIN
- `[SEJO]` - ENSDF > ENDF/B-VIII.0$^{(\beta+)}$ > JENDL/DDF-2015$^{(\beta+)}$ > Old-DCHAIN
- `[JSEO]` - JENDL/DDF-2015$^{(\beta+)}$ > ENSDF > ENDF/B-VIII.0$^{(\beta+)}$ > Old-DCHAIN
- `[SJEO]` - ENSDF > JENDL/DDF-2015$^{(\beta+)}$ > ENDF/B-VIII.0$^{(\beta+)}$ > Old-DCHAIN

In the script, the user can select a "common" name to all of the files generated when running the script at once as well as the decay library and electron capture ratio library identifying character strings (or whatever other names they'd rather assign). The files generated are then of the form `<common-name><lib-rank-ID><DCHAIN-filename>`. For example, if a common name of "`Composite-`", a decay library ranking ID of "`EJO`", and an electron capture ratio library ranking ID of "`ESJO`" are selected, the following files will be generated upon running the script:

- `Composite-EJO_spd-dcylib`
- `Composite-EJO_spd-dcytxt`
- `Composite-EJO_dose_rate_coeff.dat`
- `Composite-ESJO_lib_ec`
- `Composite-ESJO_lib_ec_with_sources`

References

[1] H. N. Ratliff, N. Matsuda, S. ichiro Abe, T. Miura, T. Furuta, Y. Iwamoto, and T. Sato, "Modernization of the DCHAIN-PHITS activation code with new features and updated data libraries," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 484, pp. 29–41, 2020. doi: https://doi.org/10.1016/j.nimb.2020.10.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168583X20304225

[2] H. Takada and K. Kosako, "Development of the DCHAIN-SP code for analyzing decay and build-up characteristics of spallation products," 1999. [Online]. Available: https://inis.iaea.org/search/search.aspx?orig_q=RN:30031678

[3] ICRP, "Conversion coefficients for radiological protection quantities for external radiation exposures. ICRP publication 116," *Annals of the ICRP*, vol. 40, no. 2-5, pp. 1–257, 2010. doi: 10.1016/j.icrp.2011.10.001. [Online]. Available: https://doi.org/10.1016/j.icrp.2011.10.001

[4] ICRP, International Commission on Radiological Protection, and International Commission on Radiation Units, *ICRP Publication 74: Conversion Coefficients for Use in Radiological Protection Against External Radiation.* Elsevier Health Sciences, 1996, vol. 23.

[5] M. Pelliccioni, "Overview of Fluence-to-Effective Dose and Fluence-to-Ambient Dose Equivalent Conversion Coefficients for High Energy Radiation Calculated Using the FLUKA Code," *Radiation Protection Dosimetry*, vol. 88, no. 4, pp. 279–297, 04 2000. doi: 10.1093/oxfordjournals.rpd.a033046. [Online]. Available: https://doi.org/10.1093/oxfordjournals.rpd.a033046

A Programs / Scripts

A.1 gen_bat_scripts_for_ENDF-to-DCHAIN.py

This Python script is called on the command line to convert a GENDF file generated by GROUPIE to the DCHAIN format. This script replaces the old Fortran script since it could not handle any libraries aside from JENDL/AD-2017. This Python script must be provided with 3 command line arguments: "i" input GENDF filename/path (from GROUPIE), "o" output DCHAIN-formatted library file name/path, and "libname" descriptive name of the library from which this file is sourced.

A.2 gen_bat_scripts_for_ENDF-to-DCHAIN.py

This Python script is used to generate the large batch script which automatically converts a folder of ENDF-formatted data files to a separate folder of DCHAIN-formatted data files. A sample of the produced script is shown in Appendix A.2.1 immediately below the following Python script.

A.2.1 convert_ENDF_to_DCHAIN_lib.bat (sample)

Below is a snippet of the batch script `convert_ENDF_to_DCHAIN_lib.bat` that was generated by gen_bat_scripts_for_ENDF-to-DCHAIN.py for an example isotope data file for ^{105}Ag downloaded from the JENDL library. This code is simply repeated for every different ENDF file present in the source ENDF library directory.

Note that the "`action_zone`" directory is simply where all of the input files for PREPRO and the compiled Fortran script are kept. The batch script essentially copies each ENDF file from its own directory to this one, creates a DCHAIN-formatted version, and moves that final output file to a separate directory of DCHAIN-formatted files.

```
copy /y "C:\path\to\ENDF-formatted_libraries\jendl-ad2017_0K\Ag105.nt0" "C:\path\to\
    action_zone\ENDFB.IN"
endf2c
linear
recent
del LINEAR.OUT
sigma1
del RECENT.OUT
activate
del SIGMA1.OUT
fixup
del ACTIVATE.OUT
dictin
del FIXUP.OUT
groupie
move /y "C:\path\to\action_zone\GROUPIE.OUT" "C:\path\to\action_zone\fort.50"
gendf-dchain.exe
move /y "C:\path\to\action_zone\fort.60" "C:\path\to\DCHAIN-formatted_libraries\
    jendl-ad2017_0K\Ag105_nt0.dchainlib"
```

A.3 check_ENDF_DCHAIN_folder_diff.py

This script simply scans the output DCHAIN folder and sees if any files from the original ENDF folder were not successfully converted.

A.4 compile_DCHAIN_xs_data_lib.py

This script sorts and compiles the many individual DCHAIN-formatted data files for each isotope/isomer into a large single file used by the actual DCHAIN program.

A.5 convert_decay_data_ENDF-to-DCHAIN.py

This script compiles the many individual ENDF-formatted decay data files for each isotope/isomer into a large single file used by the actual DCHAIN program. This script was originally written in Python 2.7 before my time on the PHITS team. I have just converted it to Python 3 using *2to3*, shuffled some lines around for convenience, and added functionality for command line specification of the directory where the code will be looking for ENDF files.

A.6 parse_DCHAIN_decay_data_file.py

This script parses the DCHAIN decay data file to determine what isotopes are present in it. This list of isotopes can then be checked against the $E.C./\beta+$ ratio file and the dose coefficient file to see which, if any, isotopes are missing from them that are present in the decay data library. Additionally, this script generated the more human-readable version of the DCHAIN decay data library. It also saves a NumPy array of all of the nuclides present in the decay library which can be used by other codes.

A.7 parse_ENDF_files_for_EC_ratios.py

This script parses ENDF decay data files, specifically the MF=8 MT=457 portions, to extract the fraction of $E.C./\beta+$ decays which result in positron emission. This can be done using emitted positron intensities or annihilation photon intensities. For isotopes undergoing this decay mode whose intensities cannot be found, they are derived from Q and A.

A.8 parse_ENSDF_files_for_EC_ratios.py

This script parses ENSDF decay data files to extract the fraction of $E.C./\beta+$ decays which result in positron emission. In this code, only fractions explicitly stated are considered.

A.9 compare_EC_lib_methods.py

This script compares various $E.C./\beta+$ ratio libraries (old, ENDF, ENSDF, etc.) and generates a table containing the values from all listed files for use in a diagnostic Excel spreadsheet (using cell conditional formatting to easily locate outliers). Additionally, the code is used to assemble a hybrid EC library file which will end up having all of the same isotopes present as the desired decay data library but can pull from multiple lists. For instance, while ENSDF data is typically the most reliable, it does not contain all of the isotopes from the JENDL DDF. So, this script pulls what information it can from the ENSDF-derived EC library and fills in the gaps from the other EC libraries (JENDL-derived and the original one).

A.10 study_IBp_vs_Q_vs_A_relationship.py

This script is a tool used to study the relationship between positron emission probability from a $E.C./\beta+$ decay and that isotope's A and the decay's Q-value. It was used to generate the new $I_{\beta+}$ function.

A.11 generate_dose_rate_coeff_lib.py

This script generates the dose rate coefficient library used by DCHAIN such that all of the isotopes within it match those contained in the decay data library being used.

A.12 decay_lib_master_assembly.py

This script assembles composite decay libraries by combining already existing ones according to the user's preferences. It's usage is discussed in Section 3.