RECONR Documentation

Jeremy Lloyd Conlin

Mar 22, 2021

CONTENTS:

The RECONR module is used to reconstruct resonance cross sections from resonance parameters and to reconstruct cross sections from ENDF nonlinear interpolations schemes. The output is written as a pointwise-ENDF (PENDF) file with all cross sections on a unionized energy grid suitable for linear interpolation to within a specified tolerance. Redundant reactions are reconstructed to be exactly equal to the sum of their reconstructed and linearized parts at all energies.

This is the documentation for the modern RECONR module[1]. Please see the following pages for more information.

[1] Much of this documentation was taken from the NJOY2016 manual and adapted as needed for the modern module in NJOY21.

CHAPTER

ONE

OVERVIEW

RECONR module is used to reconstruct resonance cross sections from resonance parameters and to reconstruct cross sections from ENDF nonlinear interpolation schemes. The output is written as a pointwise-ENDF[1] file with all cross sections on a unionized energy grid suitable for linear interpolation to within a specified tolerance. Redundant reactions (for example, total inelastic, charged-particle reactions) are reconstructed to be exactly equal to the sum of their reconstructed and linearized parts at all energies.

1.1 Linearization

In the evaluation file, the cross sections are provided using a handful of interpolations and often uses multiple interpolation regions to represent a cross section across the full energy range.

RECONR linearizes the cross sections and combines the interpolation regions into a single region for each reaction. Modern RECONR uses the broken-stick algorithm [[PTCdL17]] to perform the linearization—which really is the same algorithm that is used in Legacy NJOY. This algorithm is implemented in our twig component. This is sometimes known as Segmented Regression.

1.1.1 Broken Stick Algorithm

The linearization process begins with a sorted grid of energy values. For cross section linearization, the initial energy grid comes from the *Energy Grid Unionization*. For linearization of reconstructed resonances, the initial energy grid is the peaks and half-height energies of the resonances.

The linearization "stack" is primed with two starting values; the first two values of the initial energy grid. The stack is said to be inverted because the lower energy is at the top.

This interval or panel is now divided into two parts, and the cross section computed at the intermediate point is compared to the result of linear interpolation between the adjacent points. If the two values do not agree within various criteria, the top of the stack is moved up one notch, and the new value is inserted. The code then repeats the checking process for the new (smaller) interval at the top of the stack. The top of the stack rises until convergence is achieved for the top interval. The top energy and cross section are then saved on a scratch file, the stack index is decremented, and the checks are repeated. This process is continued with the top of the stack rising and falling in response to the complexity of the cross section until the entire panel ΔE has been converged. The stack is then reprimed with the bounds of the next panel. The process continues until the entire energy range has been processed.

The convergence criterion used for linearization is that the linearized cross section at the intermediate point is within the fractional tolerance `err` of the actual cross section specified by the ENDF law; `err` is one of the *input arguments* for RECONR.

[1] Often refered to as a PENDF file.

When linearizing with this procedure, one often enters into the situation—particularly with a discontinunity—where adjacent energy values become so close they will be rounded to the same number when written out[2]. Modern RECONR avoid this by declaring the linearization "converged" when adjacent energy points are within a relative difference of 1E-7, regardless of the specified tolerance, `err`.

This process is described graphically in Fig. 1.1.

1.2 Energy Grid Unionization

Having a common energy grid for all cross sections is important to be able to simply add cross sections together—whether adding the reconstructed cross sections to the background or adding partial cross sections to obtain a redundant cross section. The `unionizeEnergyGrid` *function* performs this task. It is called twice during a RECONR run; after the cross sections have been linearized, and after the resonances have been reconstructed. The second call just appends energy values to the first call.

1.3 Resonance Reconstruction

The actual resonance reconstruction is performed using the resonanceReconstruction repository, one of the components that make up NJOY21. The details of how the reconstruction is performed is not given here.

`resonanceReconstruction` provides functions that return cross section values for a given energy. The linearization of the reconstructed cross section is done using these functions and the linearization strategy described *earlier*.

`resonanceReconstruction` supports the following resonance formalisms:

- Resolved
  - Single- and Multi-level Breit Wigner (LRF=1 and 2)
  - Reich-Moore (LRF=3)
  - R-Matrix Limited (LRF=7)
- Unresolved
  - Energy independent (Case A)
  - Energy dependent fission widths (Case B)
  - Energy dependent (Case C)

  Only the fully energy dependent parameters were implemented since the other two formats can be translated to the more general case of fully energy dependent parameters.

Note: The Adler-Adler formalism (LRF=4) is not supported in modern RECONR. The last evaluation that used the Adler-Adler formalism was ^{233}U from ENDF/B-VI.8, which is now 20+ years old. If there is a need to process evaluations with the Adler-Adler formalism, please use Legacy NJOY.

Once the cross sections have been reconstructed from the parameters given on the evaluation file, they are added to the background cross section.

[2] In ENDF, the precision is limited to seven significant digits.

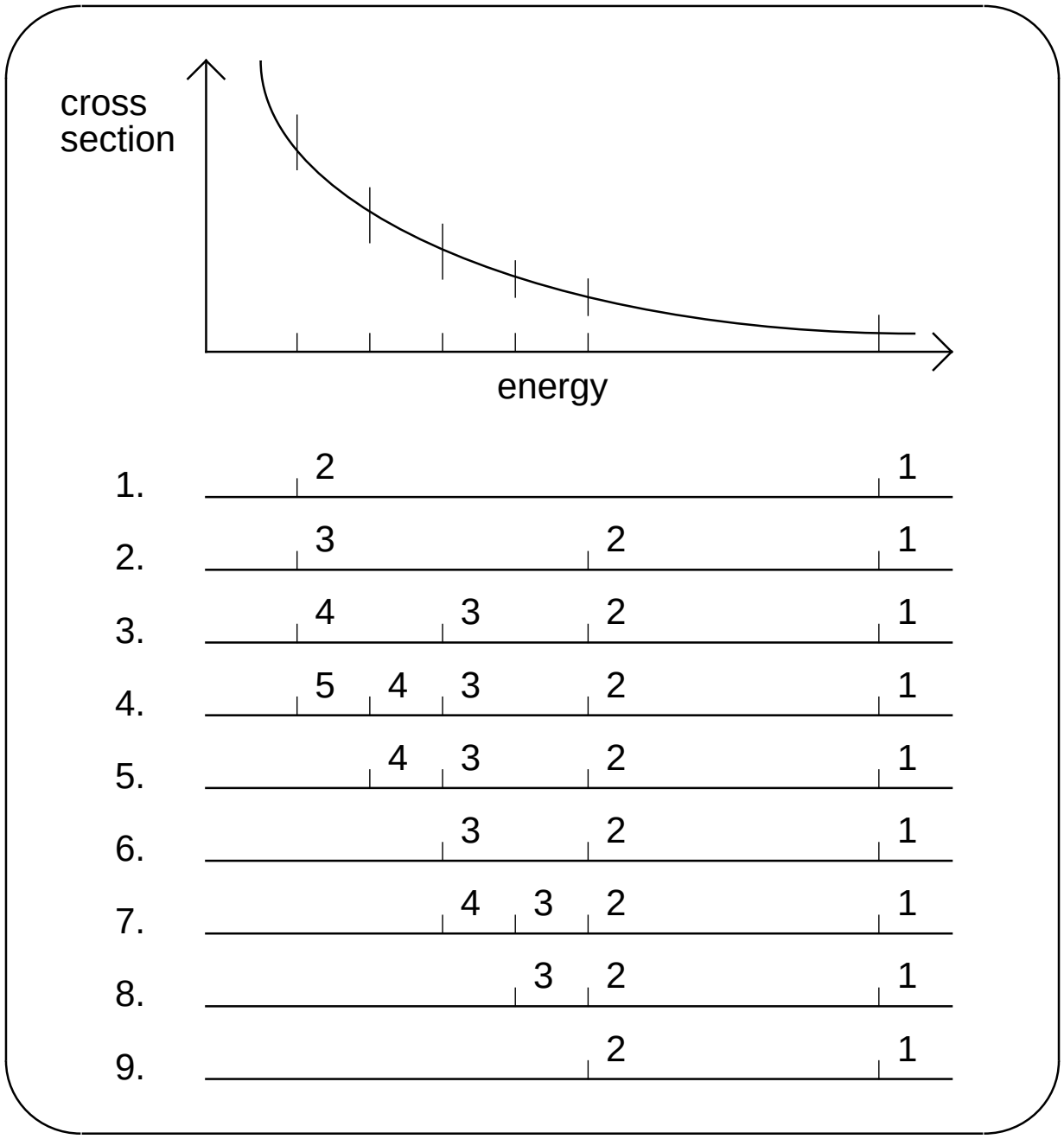

Fig. 1.1: Line 1 shows the two initial points (the lower energy is higher in the stack). In line 2, a new point has been calculated at the midpoint, but the result was not converged, and the new point has been inserted in the stack. In line 3, the midpoint of the top panel has been checked again, found to be not converged, and inserted into the stack. The same thing happens in line 4. In line 5, the top panel is found to be converged, and the top point (5) has been written out. The same thing happens in line 6. In line 7, the top panel is tested and found to be not converged. The midpoint is added to the stack. Finally, in line 8, the top panel is found to be converged, and the top point is written out. This leaves two points in the stack (see line 9). Note that the energy points come off the stack in the desired order of increasing energy, and that only one point has to be moved up in the stack as each new result is inserted.

1.4 Summation of Redundant Reactions

RECONR will calculate the redundant cross sections and ignores the redundant cross sections on the evaluation file. Since all the cross section values have already been linearized and calculated on the same energy grid, summing the cross sections is as simple as adding vectors of data.

Note: The rules RECONR uses to determine which partial reactions make up a redundant reaction are those given in the ENDF manual, table 14 [[THB18]].

1.5 Truncation of Cross Section Arrays

Many (most?) of reactions are threshold reactions and thus don't span the same energy range. Since RECONR uses the same unionized energy grid for all reactions, many reactions have many zeros before a non-zero cross section value is given. Modern RECONR will truncate all the reactions that leading zeros so as not to bloat the size of the processed evaluation. This is done in the *truncate* method.

CHAPTER

TWO

RUNNING RECONR

The output of RECONR is an ENDF-like file that is generally referred to as a *PENDF* file, for pointwise-ENDF.

2.1 Input Description

RECONR follows the same rules as all the other modules. Please see the more general information about NJOY inputs if unfamiliar. RECONR input is given in six different cards. Each card has one or more arguments and is (optionally) ended with the forward slash: `/`.

Note: The input for modern RECONR is the same as for Legacy RECONR, but modern RECONR use all of the arguments. Listed here are only the required arguments. If other arguments are given, they are silently ignored.

2.1.1 Card 1

`nendf` Unit for ENDF tape input. See input/output unit.

`npend` Unit for PENDF tape output. See input/output unit.

2.1.2 Card 2

`tlabel` Single-quote delimited string of 66 ASCII characters or less, ending with a forward slash. The value of `tlabel` will be used as the TPID for the PENDF file.

2.1.3 Card 3

`mat` ENDF Material number to be reconstructed. This is a positive four-digit number.

`ncards` Number of cards of descriptive text (default=0)

`ngrid` Number of user specified energy grid points to be added (default=0)

2.1.4 Card 4

err Fractional reconstruction tolerance used to linearize cross sections and reconstruct resonance parameters.

Please note that Legacy RECONR does have limited capabilities for reconstruction at a particular temperature. Modern RECONR does not and reconstructs everything at zero Kelvin.

2.1.5 Card 5

cards Single-quote delimited string of 66 ASCII characters or less, ending with a forward slash. The value of the argument is added to MF=1/MT=451. This card must be repeated `ncards` times where `ncards` is given in Card 3.

2.1.6 Card 6

enode A list of energy grid points with units of electon volt. These energy grid points will be added to the generated list of energy points.

Cards 3, 4, 5, and 6 are repeated for every material that should be processed. While this is generally not done anymore, the capability still exists.

RECONR is terminated with a final card that is just `0 /`

2.2 Examples

2.2.1 Simple Input

```
reconr
 20 30                            / Card 1
 'A point-wise ENDF file (PENDF)' / Card 2
 2631                             / Card 3
 0.001                            / Card 4
 0                                / Terminate RECONR execution
stop
```

2.2.2 Input with Descriptive Text and User-defined Energies

```
reconr
 20 30                            / Card 1
 'A point-wise ENDF file (PENDF)' / Card 2
 2631 2  3                        / Card 3
 0.001                            / Card 4
 'Material 2631 processed '       / Card 5.1
 'with modern RECONR'             / Card 5.2
 1.0, 2.0, 3.0                    / Card 6
 0                                / Terminate RECONR execution
stop
```

2.2.3 Multi-Material Input

```
reconr
 -40 -41
 'A PENDF file with multiple materials' /
 125 1 0                    / Card3 #1
 0.001                      / Card4 #1
 'H1 from ENDF/B-VII.1'     / Card5 #1
 2631 1                     / Card3 #2
 0.001                      / Card4 #2
 'Fe-56 from ENDF/B-VII.1'  / Card5 #2
 9228 2 0                   / Card3 #3
 0.001                      / Card4 #4
 'U-235 from ENDF/B-VII.1'  / Card5 #4.1
 'Another TEXT Record    '  / Card5 #4.2
 0                          / End RECONR
stop
```

CHAPTER

THREE

CODE DESCRIPTION

All modern NJOY components are being written in C++. They can be used stand-alone, but are also used in connection with NJOY21.

The RECONR module is implemented as a class with a single public method, the call operator, which is called by NJOY21 and the input parameters are passed to it.

```
class RECONR {
/* private methods */
public:
   void operator()( const nlohmann::json& njoyArgs,
                    std::ostream& output,
                    std::ostream& error,
                    const nlohmann::json& other ){
   }
};
```

Note: All NJOY21 modules have the same argument list for the call operator.

The arguments to `operator()` are:

`njoyArgs` This argument is a JSON object translation of the original input arguments. When RECONR is executed, this JSON object is echoed to the output

`output` Where informational messages are printed. You can specify what file this is written to using the `--output` command-line option in NJOY21. When not specified, these messages are written to `stdout`.

`error` Where error messages are printed. You can specify what file this is written to using the `--error` command-line option in NJOY21. When not specified, these messages are written to `stderr`.

`other` This argument is currently unused in RECONR. It is a place holder for if/when we decide to give additional data to *all* NJOY21 modules.

The call operator is analogous to the `reconr` subroutine in Legacy NJOY. It then makes a series of calls to other methods to perform all the calculations that are performed.

The data needed to perform the operations done in RECONR is contained in a `ResonanceReconstructionDataDelivery` *ResonanceReconstructionDataDelivery* object. After the input evaluation file is read in with the `getEvaluated` function, the data is passed to the `ResonanceReconstructionDataDelivery` constructor to:

3.1 RECONR Methods

Listed here are the different methods (i.e., functions) called by the `RECONR` class to perform the operations of the RECONR module.

`getEvaluated` Read the evaluated file; i.e., ENDF or (eventually) GNDS. This returns a `std::variant` which is then passed to `findR2D2`.

`findR2D2` Create an instance of the *ResonanceReconstructionDatatDelivery* object.

`linearizeXS` Linearize all of the background cross sections and photon production cross sections. It uses the interpolation library to interpolate between cross section values.

`unionizeEnergyGrid` Create an energy grid used as a common energy grid for reconstruction. It uses these energy values:

- Cross section energies,
- Photon production energies,
- Resonance range boundaries, and
- Resonance energies.

It is in this method that the infamous `sigfig` method is used to avoid discontinuities. When two adjacent energy values are found, the first value is nudged down and the second value is nudged up so that they are no longer identical. `sigifig` does the nudging.

I tried to remove the need for `sigfig`, but it caused problems at the interface of the resonance regions, giving incorrect answers.

`reconstructResonances` This is an overloaded function that, well, reconstructs the resonances based on the formalism. These functions call to resonanceReconstruction to do the actual work. The function creates a linear-linear interpolation table for each of the reactions that are reconstructed and adds them to the `ResonanceReconstructionDataDelivery` instance.

`reconstructUnresolved` This method reconstructs the unresolved resonances, also using resonanceReconstruction. The function creates a linear-linear interpolation table for each of the reactions that are reconstructed and adds them to the `ResonanceReconstructionDataDelivery` instance.

`reconstructCrossSections` After all the resonances have been reconstructed and the energy grid unionized, `reconstructCrossSections` creates (E, σ) pairs (actually stored as a `std::pair< std::vector< double >, std::vector< double > >`).

`combineUnresolved` Combine the unresolved cross sections with the background cross sections. This is all dependent on the value of `LSSF` flag.

`combineReconstructed` This will add the reconstructed cross sections to the background cross sections.

`summateUnresolved` This sums up the unresolved cross sections and—depending on the `LSSF` flag—adds it to the background cross section.

`summateReactions` Add up all the partial cross sections to calculate the redundant (e.g., total) cross section. To determine the definition of what is redundant and what is not, it uses the ENDF definition of a redundant cross section—Table 14 in the current edition as of this writing.

Note: NJOY21 ignores redundant cross sections as given in the evaluation file. It will simply recaluclate the redundant cross section from its partials. If there are no partials, it treats the redundant reaction as any other reaction.

---

summateProductions Similar to `summateReactions`, this adds up the partial photon production cross sections to create the redundant photon production cross sections.

truncateReactions All the cross sections are evaluated on the same energy grid. For some reactions (i.e., threshold), the cross section is zero when the energy is less than the Q-value. This function will remove energy, cross section values from the beginning of a reaction when the cross section value is zero.

3.1.1 ResonanceReconstructionDataDelivery

The `ResonanceReconstructionDataDelivery` object[1] is designed to hold all the data needed for RECONR. Rather than passing a number of arguments to a variety of functions, only the `ResonanceReconstructionDataDelivery` object needs to be passed and it can be queried for all the data it posses.

This object is built using a factory builder pattern. The Factory performs the following operations

- Collect all cross sections (e.g., MF=3) and photon production cross sections (e.g., MF=13), converting them to interpolation tables using our interpolation library.
- Storing the resonance parameters (e.g., from MF=2).
- Storing the min/max energy values for the resonance ranges (resolved and unresolved).

Once the Factory has collected all the data from the evaluation file (e.g., ENDF), it passes the data to the constructor for `ResonanceReconstructionDataDelivery`.

3.1.2 Processed Evaluation

This class takes the ResonanceReconstructionDataDelivery object and writes an ENDF or (eventually) GNDS file.

3.1.3 Reactions

So there are really three different reaction classes.

Reaction Contains data for a standard (i.e., from an ENDF MF=3 section or a GNDS `reaction`) reaction.

UnresolvedReaction Contains data for unresolved reactions.

PhotonProduction Contains data for photon production reactions (i.e., from an ENDF MF=13 section).

They each contain a variety of parameters read from the evaluation file as well as containing a representation of the cross sections. The representation changes as RECONR progress. It begins as an interpolation table and ends as pairs of energy, cross section values.

3.1.4 Additional Components

RECONR takes advantage of a number of other components. Some of these components are libraries that we have developed with the bigger NJOY21 project, others components are third-party libraries that are available with a compatible Open Source license. Listed here are some of those components.

Internally Developed Components While these components/libraries are internally developed as part of the NJOY21 project, they are generally available on the NJOY GitHub site.

- ENDFtk is used to read and write ENDF-formatted data.

[1] affectionately known as the R2D2 object and even has an `R2D2` alias.

- resonanceReconstruction contains all the capabilities for performing the reconstruction of unresolved and resolved resonance.
- elementary is used to create reaction identifiers given the products and reactants of a nuclear reaction.
- interpolation is a library that can make interpolation tables that can then be, ahem, interpolated using various interpolation schemes.
- constants is a library providing mathematical and physical constants.
- dimwits is a library providing dimensional analysis with units.

Third-Party Components

- Catch is a testing frame work that we use to test our code.
- Range-v3 a wonderful little tool that has been (mostly) adopted into the C++20 standard. As of this writing, we are using a rather old version of this library.
- JSON for Modern C++ is used for working with JSON data.

CHAPTER

FOUR

BIBLIOGRAPHY

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[PTCdL17] Norman Poh, Santosh Tirunagari, Nicholas Cole, and Simon de Lusignan. Probabilistic broken-stick model: a regression algorithm for irregularly sampled data with application to egfr. *Journal of Biomedical Informatics*, 76:69 – 77, 2017. URL: http://www.sciencedirect.com/science/article/pii/S1532046417302253, doi:https://doi.org/10.1016/j.jbi.2017.10.006.

[THB18] A. Trkov, M. Herman, and D. A. Brown. ENDF-6 formats manual: data formats and procedures for the evaluated nuclear data files ENDF\/B-VI, ENDF\/B-VII and ENDF\/B-VIII. Technical Report BNL-203218-2018-INRE, National Nuclear Data Center, Brookhaven National Laboratory, February 2018.

INDEX