LLNL's FUDGE and GIDI+ Code Packages: for Managing, Processing and Accessing GNDS 2.0 Nuclear Data Libraries

B. R. Beck, C. M. Mattoon, and G. Gert
Nuclear Data and Theory Group, NACS/PLS

Presented at CSEWG
Nov. 2022

LLNL-PRES-841930

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

FUDGE (For Updating Data and Generating Evaluations) is an open-source code for managing nuclear data libraries

- FUDGE is a set of Python modules and scripts for viewing, translating, modifying and processing nuclear data
- Requires Python-3.6 or later, numpy
  - matplotlib, PyQT5 for interactive visualization
  - Computationally intensive tasks implemented in C / C++
- Supports installation either via “git clone/Makefile” or “pip install”.
- FUDGE-6.0 released September 2022 with GNDS 2.0 support.
- FUDGE API has gone through some changes, so user scripts may need to be updated!

FUDGE-6 is available from https://github.com/LLNL/fudge.

Some API changes in latest version

- Replace the following

```
from fudge.gnds import reactionSuite, covariances
RS = reactionSuite.readXML( “file.xml” )
CS = covariances.covarianceSuite.readXML( “file-covar.xml”, reactionSuite=RS )
```

- with

```
from fudge import reactionSuite
RS = reactionSuite.read( “file.xml” )
CS = RS.loadCovariances()
```

- Can get a generic GNDS file with

```
from fudge import GNDS_file
gnds = GNDS_file.read( “file.xml” )
```

  GNDS_file.read supports reading any GNDS map, PoPs, reactionSuite or covarianceSuite file. User is responsible for checking return type.

- Other changes:
  - class names capitalized (e.g., class “product” changed to “Product”)
  - string constants converted to enums
  - Some modules moved
  - and more

Adding a reaction to a reactionSuite

```
from fudge import reactionSuite as reactionSuiteModule
from fudge.reactions import reaction as reactionModule

reactionSuite = reactionSuiteModule.ReactionSuite( 'n', 'H1', 'demo' )

reaction = reactionModule.Reaction( ... )
#
# Populate reaction with cross section and products.
#
reactionSuite.reactions.add( reaction )
```

FUDGE classes create as many child nodes as reasonable

```
>>> from fudge import reactionSuite
>>> protare = reactionSuite.ReactionSuite('n', 'H1', 'demo')
>>> print(protare.toXML(showEmptySuites=True, showEmptyText=True))
<reactionSuite projectile="n" target="H1" evaluation="demo" format=”2.0" projectileFrame="lab" interaction="nuclear">
 <externalFiles/>
 <styles/>
 <PoPs name="protare_internal" version="1.0" format="1.10">
   <documentation> <authors></authors> <contributors></contributors> <collaborations></collaborations>
     <dates></dates> <copyright/> <acknowledgements></acknowledgements> <keywords></keywords>
     <relatedItems></relatedItems> <title/> <abstract/> <body/> <computerCodes></computerCodes>
     <bibliography></bibliography> <endfCompatible/></documentation>
   <aliases/> <gaugeBosons/> <leptons/> <baryons/> <chemicalElements/> <unorthodoxes/></PoPs>
 <reactions/> <orphanProducts/> <sums><crossSectionSums/><multiplicitySums/></sums>
 <fissionComponents/> <productions/> <incompleteReactions/> <applicationData/></reactionSuite>
```

FUDGE includes Python scripts to help with some common nuclear data tasks.

- Translate ENDF-6 data into GNDS:
  - endf2gnds.py /path/to/evaluation.endf evaluation.xml
- Run physics quality checks on GNDS data file:
  - checkGNDS.py evaluation.xml
- Extract outgoing spectrum for specified product at specified projectile energy
  - energySpectrum.py evaluation.xml <product> <incidentEnergy> # More on this later.
- Process data for Monte Carlo and/or deterministic transport
  - processProtare.py -mc -mg -up -t 293.6 -t 300 --temperatureUnit K evaluation.xml proc.xml
- Generate ACE file (after Monte Carlo processing with processProtare.py)
  - python -m brownies.LANL.toACE.toACE proc.xml proc.ace -i 1
- and more! Build map files, summarize processed files, comparison plots, etc.

fudgeScripts.py: Script provides synopsize of FUDGE scripts.

```
fudgeScripts.py
addFlux.py          - Add a flux definition (label and f(T,E,mu) data) to a fluxes file (e.g., fluxes.xml).
addMultigroup.py    - Adds a multi-group boundary definition (i.e., label and the multi-group boundaries) to a
                      groups file (e.g., groups.xml).
buildMapFile.py     - Creates a map file from a list of GNDS reactionSuite and map files.
checkGNDS.py        - Reads GNDS files and runs all of FUDGE physics tests on each.
checkMap.py         - Checks a GNDS map file and its contents for consistency.
convertMapFile.py   - Converts a GNDS map file from one format to another.
crossSections.py    - Outputs the cross section for each reaction and total for a GNDS reactionSuite.
energyBalance.py    - For each reaction of a protare, writes available energy, each product's outgoing energy,
                      energy balance, etc. to files.
energySpectrum.py   - For the specified projectile energy and product, outputs energy spectra by reaction and
                      also summed spectra.
gnds2gnds.py        - Converts a GNDS file to a GNDS, allowing the new file to have different parameters
                      (e.g., format, energyUnit).
peek.py             - Prints an outlines of the reactions, and their energy domaian and products for a GNDS
                      reactionSuite file.
processProtare.py   - Processes a GNDS reactionSuite file for Monte Carlo and/or deterministic transport at
                      various temperatures.
temperatures.py     - Prints the list of temperatures in a GNDS reactionSuite and labels for each processed
                      style for each temperature.
```

There are more scripts and many more to come.

processProtare.py serves as the main driver for processing nuclear data libraries

- Supports generating multi-temperature data for Monte Carlo transport, deterministic transport or both
  - Processed results are also stored in GNDS, either in XML or in hybrid XML/HDF5 (hybrid option reduces file size and improves load times)
  - Many command line options (processProtare.py --help for details)
  - Simplify processing options by creating a standard options file and using the '@' parameter:

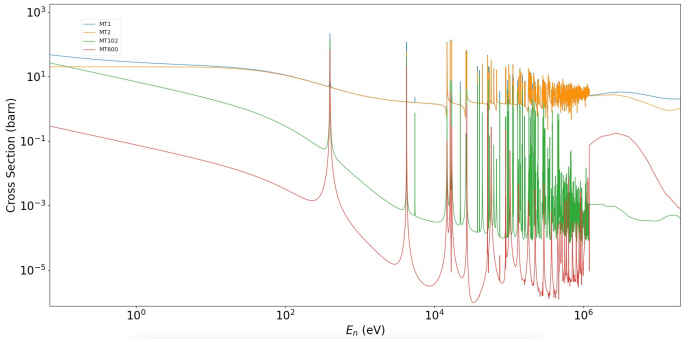
```
processProtare.py evaluation.xml proce.xml @options.input
```

```
cat options.input
--energyUnit eV --temperatureUnit K
-t 293.6
-t 300
-mc –mg –up
…
```

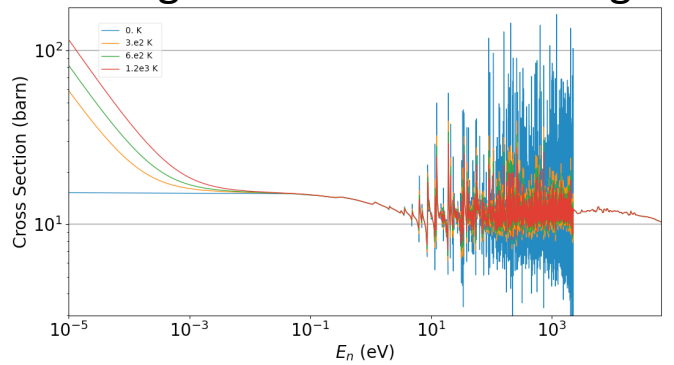
Some processProtare.py steps

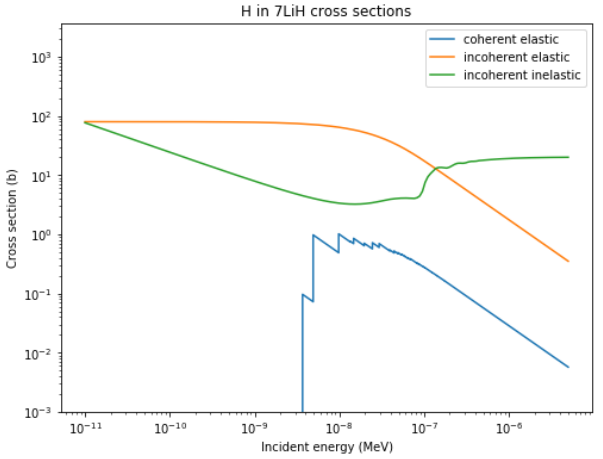
- resonance reconstruction if needed
- Doppler broadening: uses kernel broadening method
- Converting TNSL parameters to double-differential cross sections (including new LTHR=3 mixed elastic option)

n + Cl35 resonance reactions



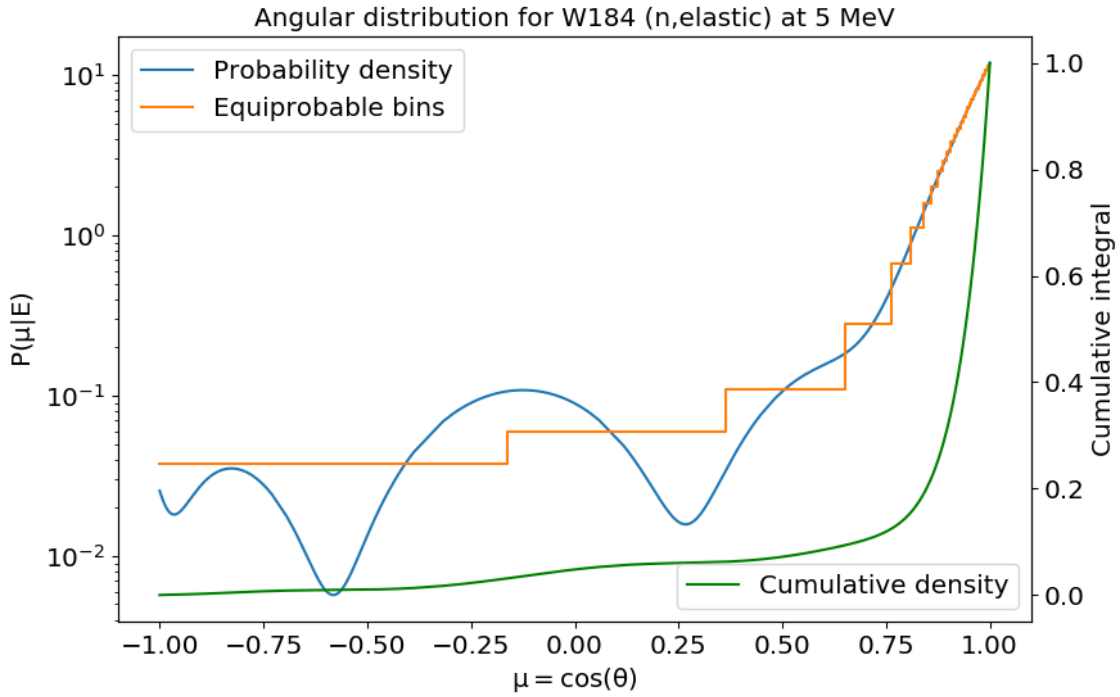
Heating U235 elastic scattering





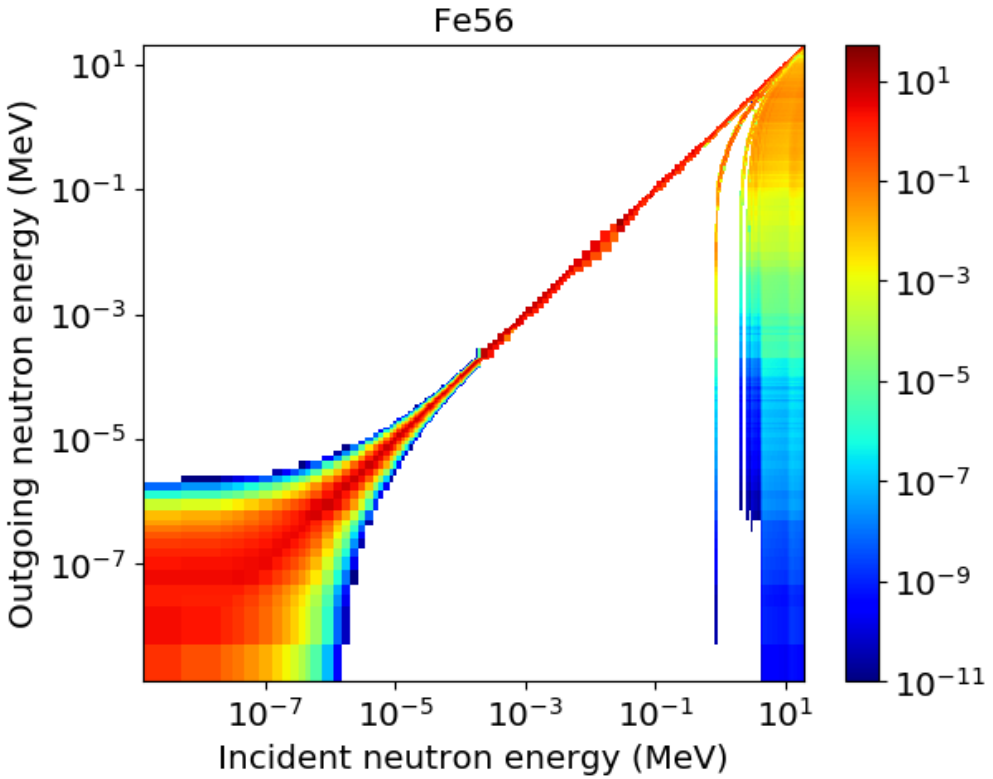
Additional steps for efficient Monte-Carlo sampling

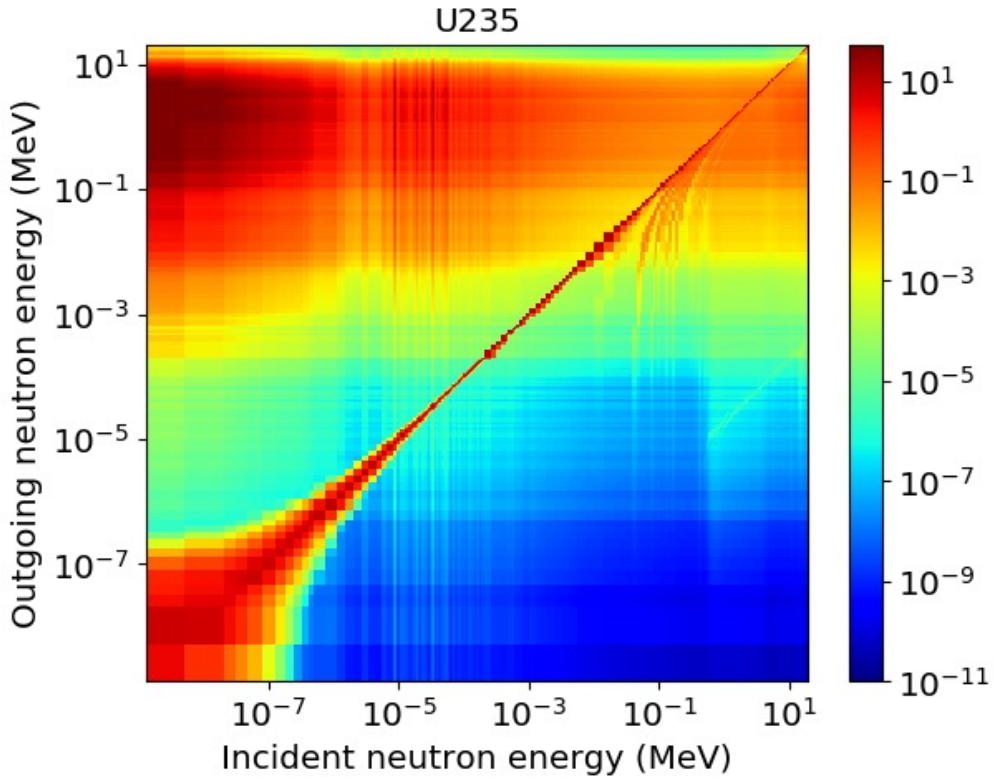
- Linearizing all functions and generating a ‘union grid’ of incident energies for all reaction cross sections – faster cross-section lookup
- Pre-compute cumulative probability density functions (CDFs) for faster sampling of distributions



Additional steps for deterministic transport: generating multi-group cross sections and transfer matrices

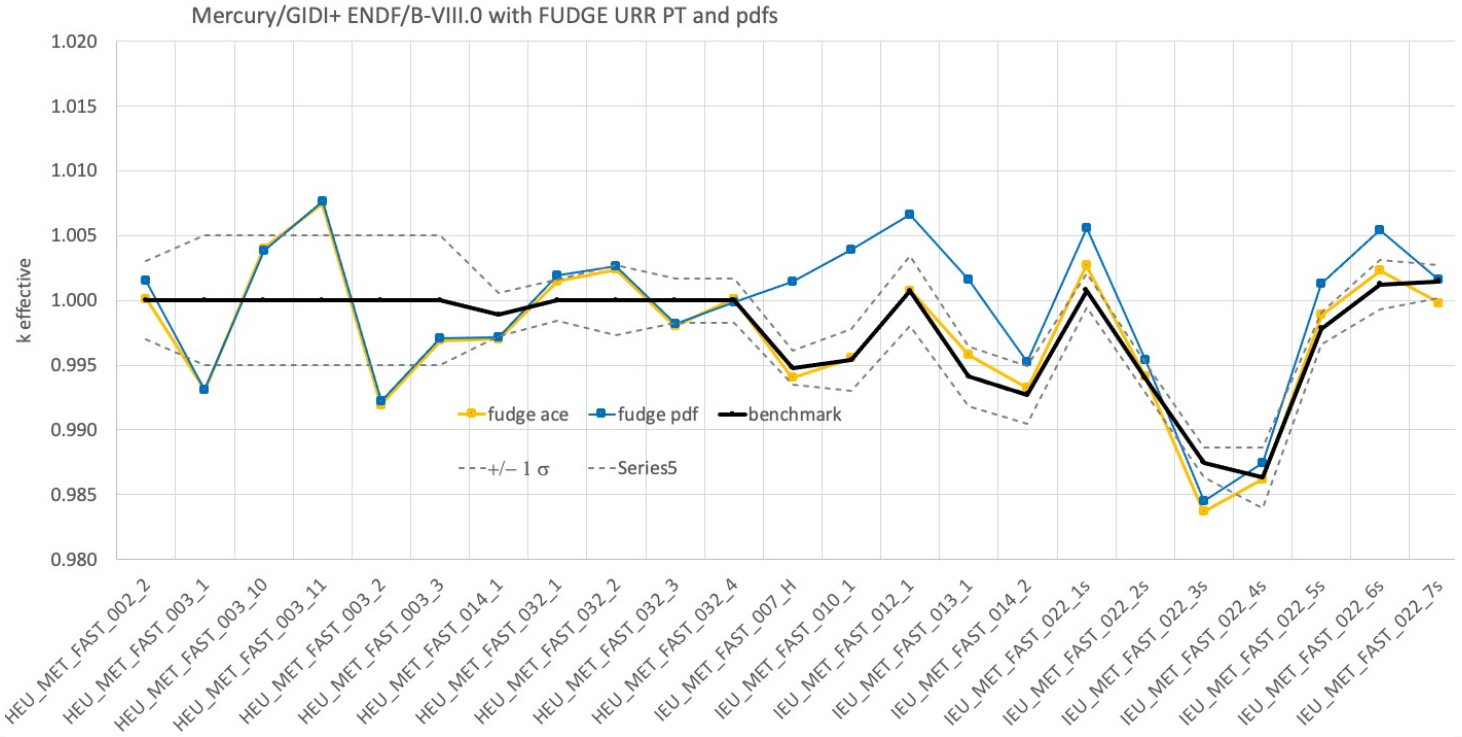
- Transfer matrices





Improved URR probability tables

- A temperature unit error was recently fix in FUDGE's URR code and we are now getting good results for FUDGE produced URR probability tables.
- Still need to automate in FUDGE (e.g., add to processProtare.py)



See Caleb Mattoon and Marie-Anne Descalle for details.

FUDGE supports generating map files to assemble reactionSuite files into a complete library

- map files are similar to MCNP's xsdir, but they support importing other map files

```
buildMapFile.py --library Test -o test.map neutrons/* all.map
```

```
<map library="Test" format="2.0" checksum="a0cf6d97b19b3a4c2affdc4d5c06d60daf1e3172" algorithm="sha1">
  <protare projectile="n" target="H1" evaluation="ENDF/B-8.0" path="neutrons/n-001_H_001.xml"
      interaction="nuclear" checksum="a7de427c92cf738d255b7abce0d740736866dcd0"/>
  <protare projectile="n" target="O16" evaluation="ENDF/B-7.1" path="neutrons/n-008_O_016.xml"
      interaction="nuclear" checksum="a1e8bcb99c64c9b574dea1e95623fa6901075ec1"/>
  <protare projectile="n" target="Th227" evaluation="JENDL-7.1" path="neutrons/n-090_Th_227.xml"
      interaction="nuclear" checksum="18eedda8ecd27bfec471e4f2f9a2816d713ddfb8"/>
  <TNSL projectile="n" target="tnsl-Al27" evaluation="ENDF/B-8.0" path="neutrons/tsl-013_Al_027.xml"
      checksum="..." standardTarget="Al27" standardEvaluation="ENDF/B-8.0"/>
  <TNSL projectile="n" target="HinCH2" evaluation="ENDF/B-8.0" path="neutrons/tsl-HinCH2.xml"
      checksum="..." standardTarget="H1" standardEvaluation="ENDF/B-8.0"/>
  <import path="all.map" checksum="80eb39043d8e884f1e74b56f22aed9183015b984"/></map>
```

buildMapFile.py uses GNDS_file.py type and preview functions, and calculates checksums.

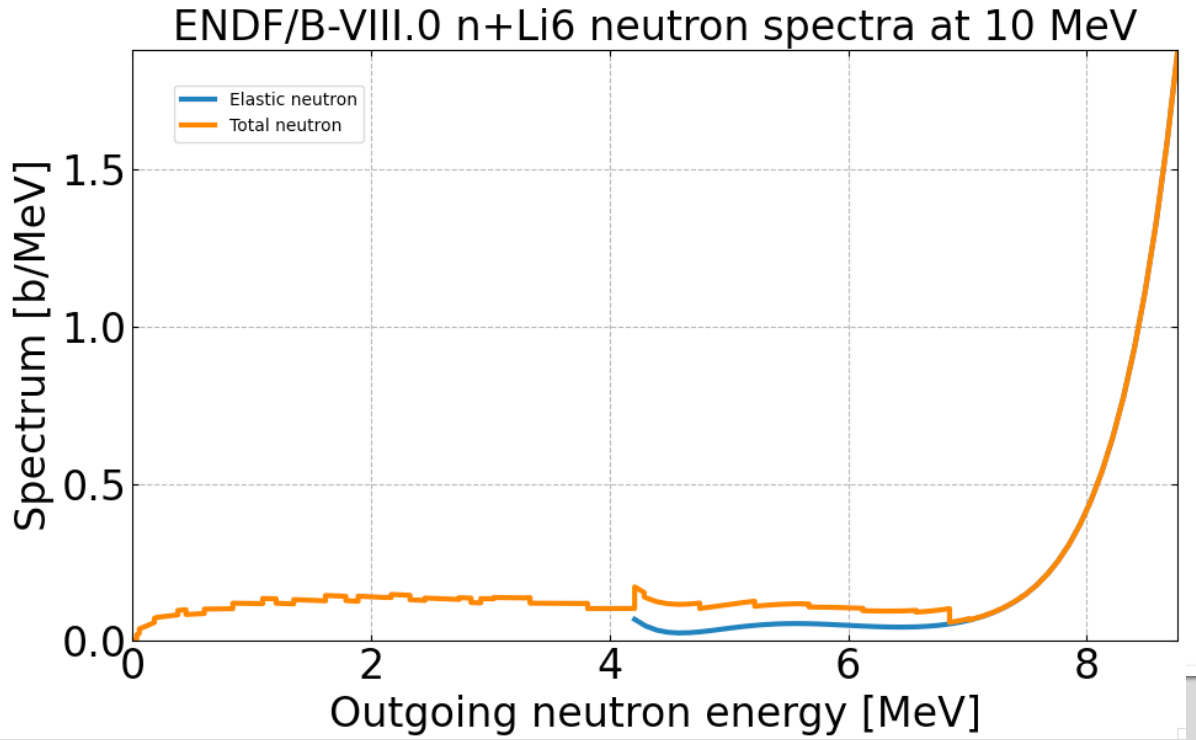
energySpectrum.py

- Outputs the energy spectra for the specified product and incident projectile energy for each reaction and several reaction sums.
  - Outputs *.spec (reaction cross section * pdf), *._pdf and *._cdf files.

- energySpectrum.py --tid Li6 ENDFB-VIII.0/neutrons.map n 10 --outputDir energySpectrum

index
info.txt

000_002.spec
000_002._pdf
000_002._cdf
…
050-091_004.spec
050-091_004._pdf
050-091_004._cdf
total_001.spec
total_001._pdf
total_001._cdf



energyBalance.py (new)

- Outputs detailed energy curves as a function of projectile's energy for each reaction (available energy, each product's outgoing energy, excess energy, etc.).
- energyBalance.py ENDF-VIII.0/neutrons/n-007_N_014.xml Out
  - Writes output into sub-directories of Out/n+N14

Sub-directories

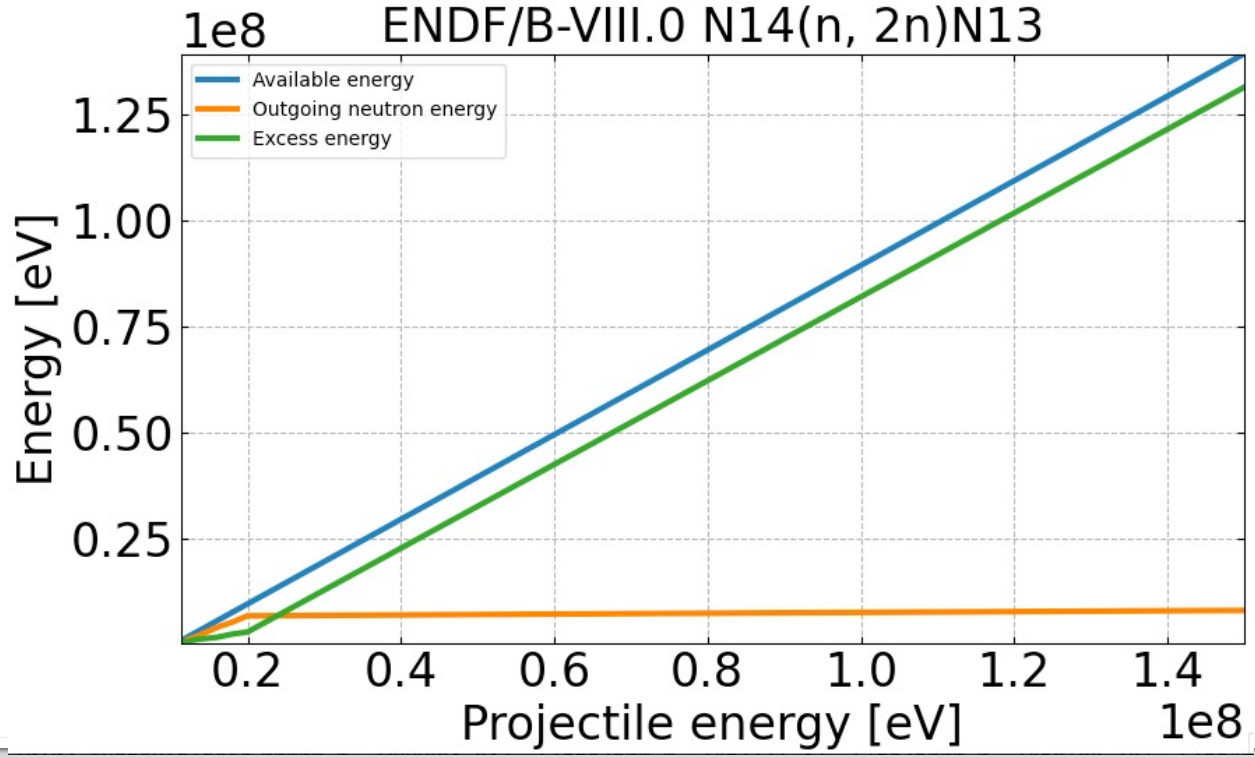
index
0000_002/
0001_051/
0002_052/
…
0059_105/
0060_107/
s004_050-091/
s103_600-649/
s104_650-699/
s105_700-749/
s107_800-849/

Files in sub-directory 0028_016/

availableEnergy.dat

_N13_averageProductEnergy.dat
_n_averageProductEnergy.dat
totalProductEnergy.dat

crossSection.dat
energyBalance.dat



What is GIDI+?

- GIDI+ is a collection of (mainly) C++ APIs (i.e., sub-packages) for reading and sampling from GNDS data as needed by transport codes.
- Main C++ APIs are:
  - PoPI: access to GNDS PoPs data.
  - GIDI: access to GNDS reactionSuite data.
  - MCGIDI: provides data lookup and sampling for Monte Carlo transport codes.
  - HAPI: Interface between GNDS data and GIDI that allows data to be in XML, HDF5 or Hybrid formats.
- Non-C++ sub-packages
  - numericalFunctions: C library for manipulating numerical 1d data (e.g., adding, multiplying).
    - Also used by FUDGE
- Third party code packages
  - pugixml:
    - Used for parsing XML files.
  - HDF5:
    - Used for reading hdf5 files.

What transport codes uses GIDI+

- Ardra:
  - LLNL deterministic transport code
  - Routinely updated with latest GIDI+

- Mercury:
  - LLNL Monte Carlo transport code
  - Routinely updated with latest GIDI+

- FUTURE plans:
  - GEANT4
    - Worldwide development coordinated by CERN
    - Written in C++
    - GEANT4 has a very old version of GIDI+ that was written in C. Does not support GNDS
    - Plan, as part of GRIN project, to update GEANT4 to the latest GIDI+.
    - We are meeting with GEANT4 people to update GEANT4

Changes to speed up loading a GNDS file into FUDGE and GIDI

- Storing a processed GNDS file in hybrid format
  - Structure is stored in XML
  - Floats and integers in a “values” node are stored in an HDF5 file.

```
n-008_O_016.xml                 # (549 MB)

      versus

n-008_O_016.xml                 # Structure in XML (36 MB)
HDF5/n-008_O_016.h5             # Float and integer data (344 MB)
```

- FUDGE and GIDI support lazy parsing
  - Basically, a large data node is not parsed into FUDGE or GIDI classes until it is accessed by the user.

- For multi-group data, FUDGE calculates multi-group sums and stores the results within the applicationData node. When accessing multi-group summed data, GIDI will read for the pre-sum data if present and requested.

EMU: Evaluations with Means and Uncertainties

- Suite for generating realizations of GNDS formatted nuclear data
  - Built on top of FUDGE for reading and writing GNDS
    - Library agnostic, only need a translator to/from GNDS
    - Translator must also **correctly** translate covariance data.
- Manages the workflow for multi-material UQ studies
  - Generation of raw nuclear data realization
  - Batch processing of realizations into heated and grouped data
  - Feeding in multi-material variations into application codes
- Python scripting framework to make interaction with sampled libraries extremely simple.
- Developer is Kyle Wendt

Some future plans

- Improve documentation/tutorials. Jupyter notebook tutorials are proving popular at LLNL
- Integrate URR probability tables processing into processProtare.py
  - Now getting good agreement with NJOY / FRENDY URR probability table results.
- More efficient processing
  - Some codes run in parallel, but many are still serial (add more threading and GPU coding).
- Adding more scripts to FUDGE
- Support direct sampling of TNSL S(T,α,β) data
- Some refactoring of FUDGE still possible
- And more

Summary

- GNDS, FUDGE and GIDI+ are replacing legacy formats and codes as the standard toolkit for nuclear data users at LLNL
- New version of FUDGE and GIDI+ supports the GNDS 2.0 standard and are available at
  - https://github.com/LLNL/fudge
    - version 6.0.0
    - Two ways to install: “pip install” or “git clone and make –s”
  - https://github.com/LLNL/
    - version 3.25.7
    - Requires C++11
    - Builds with Makefile
- We plan to release new versions every 3 months to github.com.
- All codes released under MIT license, except currently FUDGE BSD.
- For questions please contact beck6@llnl.gov, mattoon1@llnl.gov or gert1@llnl.gov

Lawrence Livermore
National Laboratory