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 IAEA
May 2023

LLNL-PRES-??????

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-3.7 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-6.2 released May 2023 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

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:

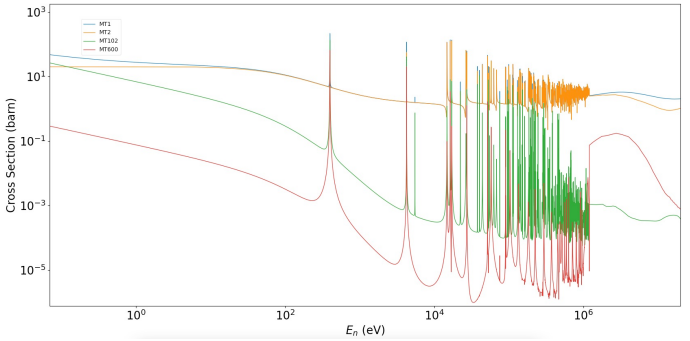
```
processProtare.py evaluation.xml proce.xml @options.input
```

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

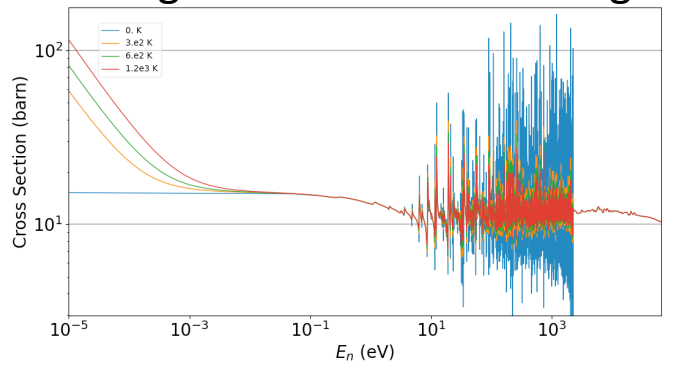
Some processProtare.py steps

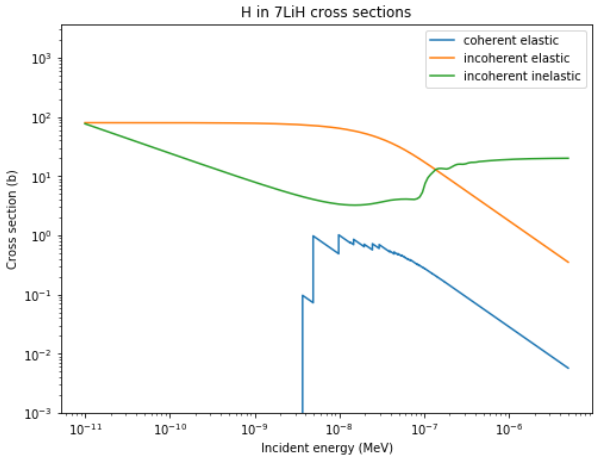
- resonance reconstruction if needed
- Doppler broadening: uses kernel broadening method
- Converting TNSL parameters to double-differential cross sections (including new LTHR=3 mixed elastic option)

n + Cl35 resonance reactions



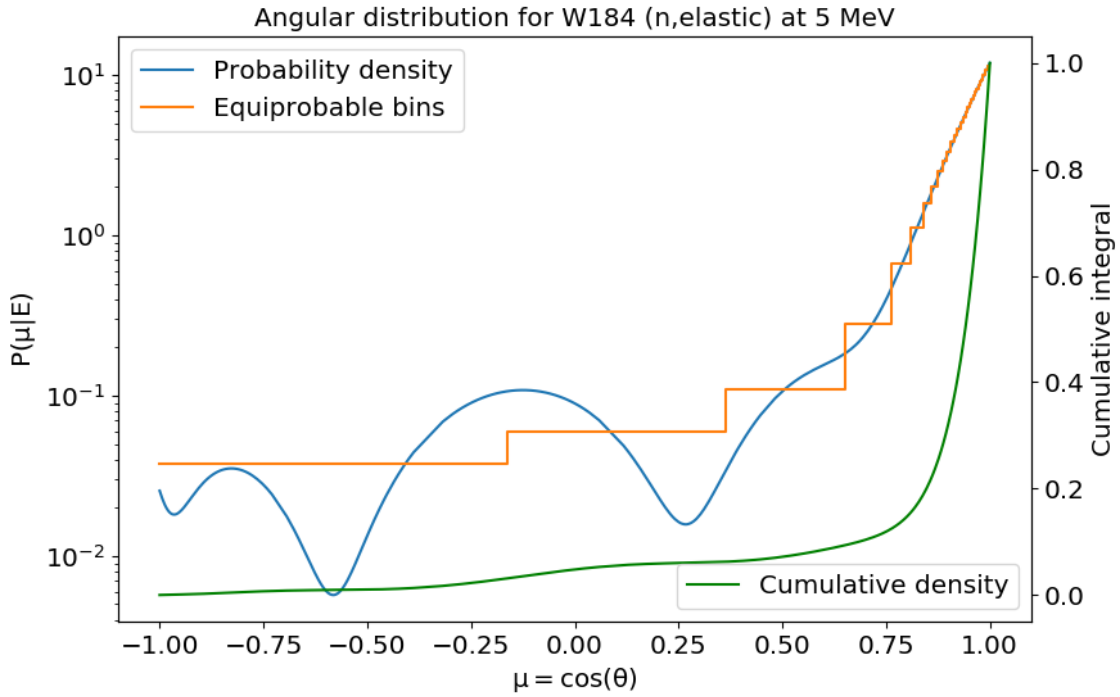
Heating U235 elastic scattering





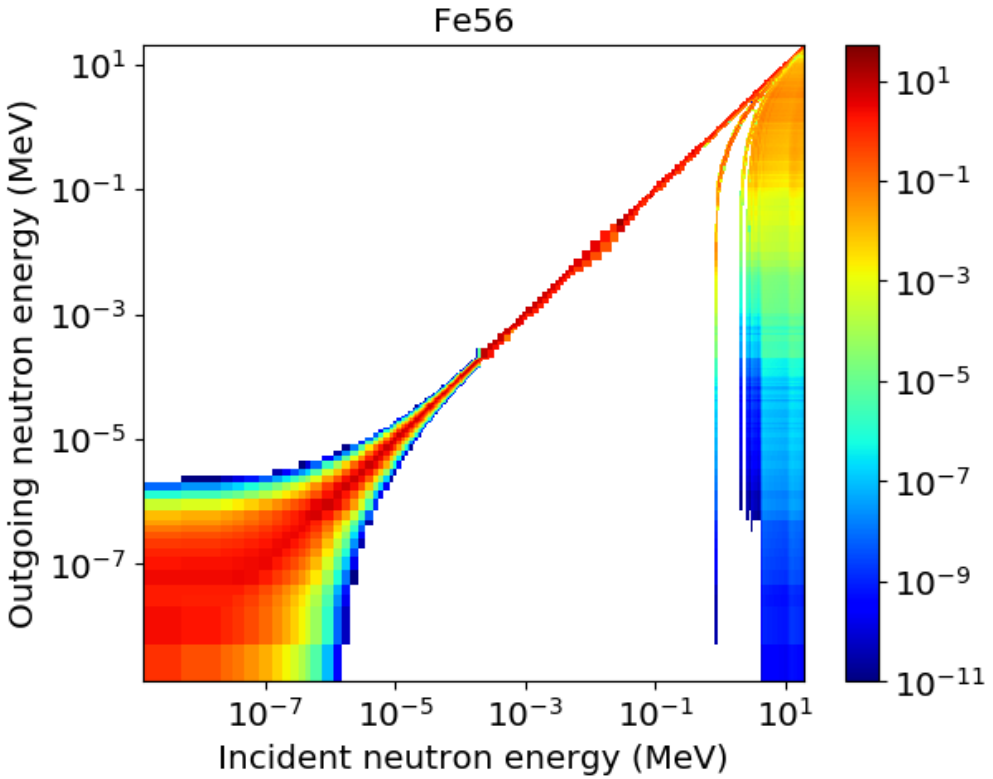
Additional steps for efficient Monte-Carlo sampling

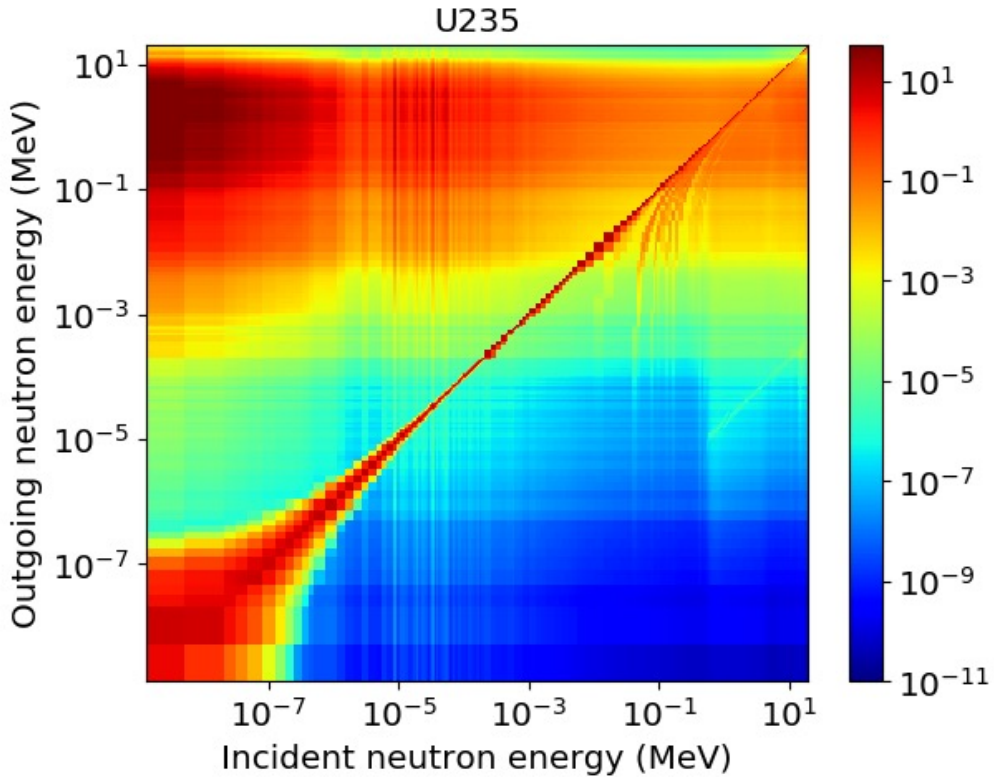
- Linearizing all functions and generating a ‘union grid’ of incident energies for all reaction cross sections – faster cross-section lookup
- Pre-compute cumulative probability density functions (CDFs) for faster sampling of distributions



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

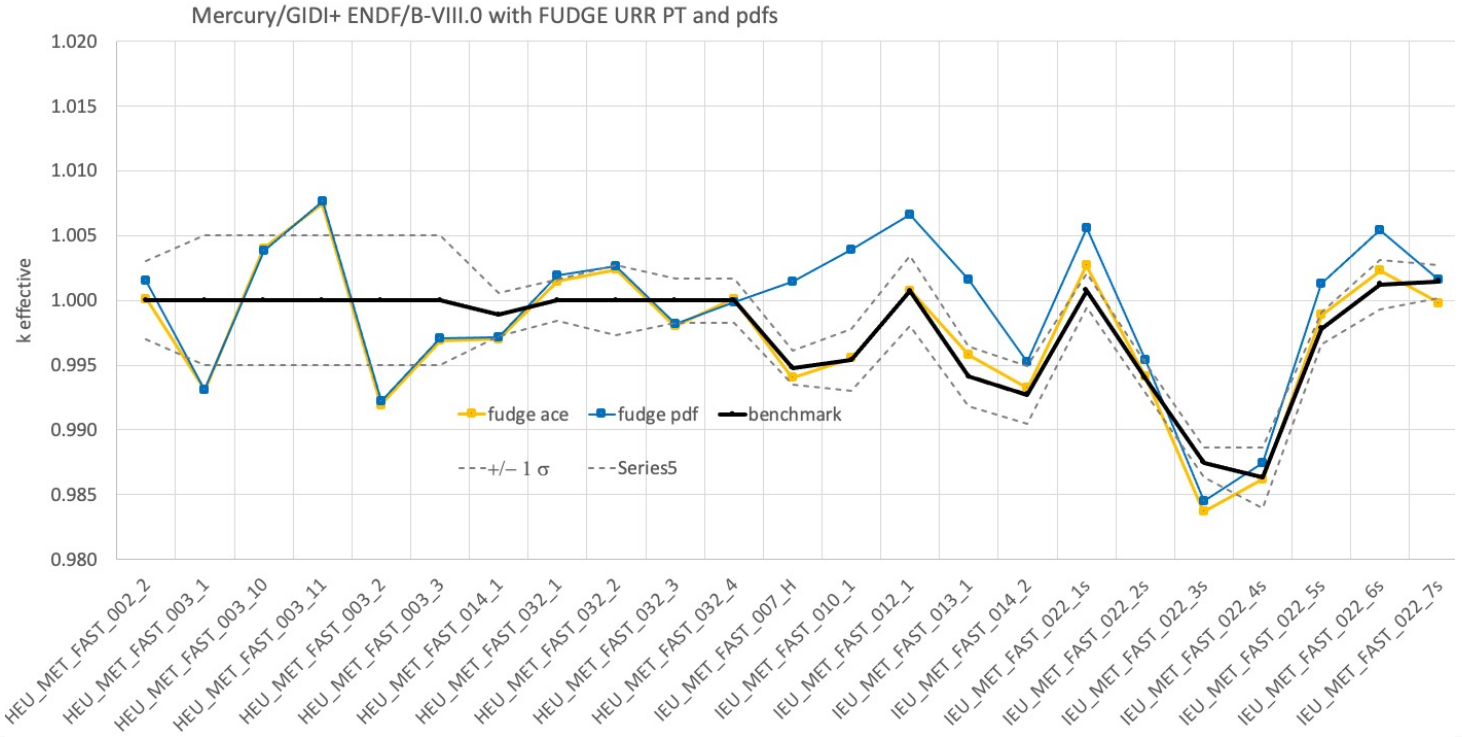
- Transfer matrices





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.

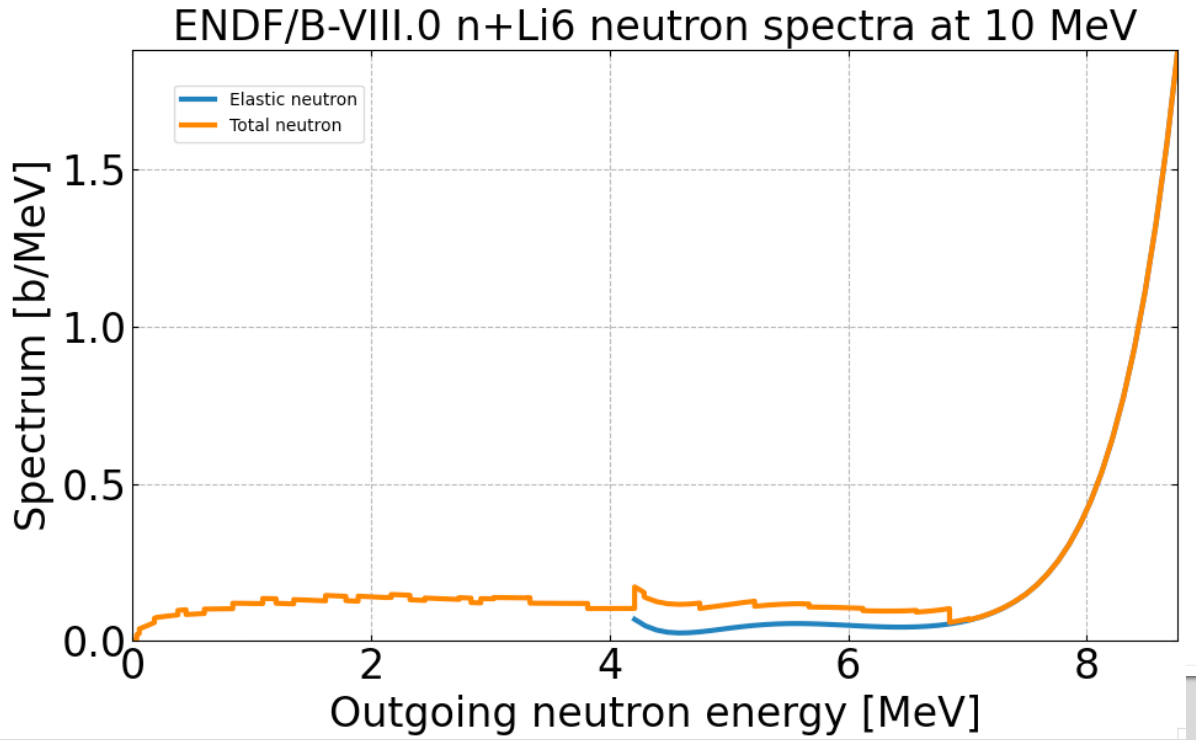
energySpectrum.py

- 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

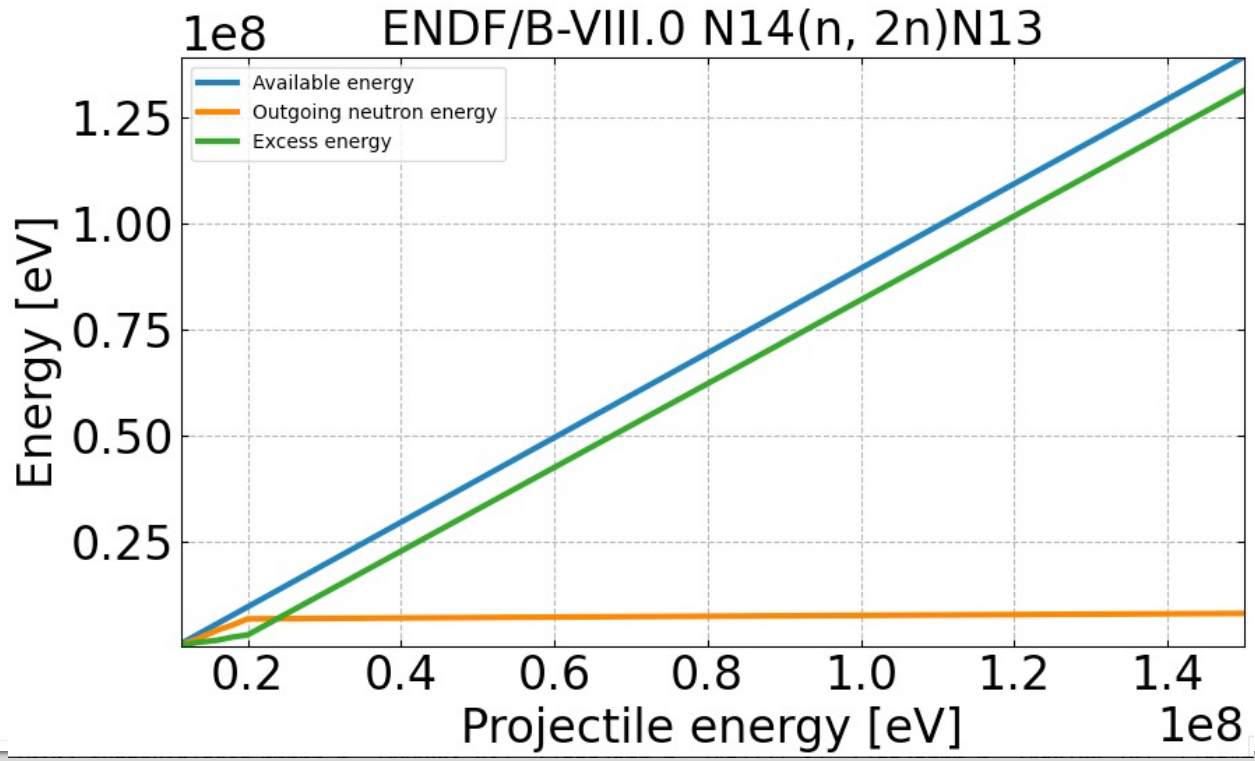
index
0000_002/
0001_051/
0002_052/
…
0059_105/
0060_107/
s004_050-091/
s103_600-649/
s104_650-699/
s105_700-749/
s107_800-849/

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

- Our users have asked us to include all data, including processed data, for one protare (e.g., ”n+O16” or “p+Li6”) into ”one” file. This includes:
  - Original evaluated data
  - The following data at 25 temperature (was 23 but we have been asked to include 0 and 77 K).
    - Data for Monte Carlo transport
    - Data for determinist transport (multi-group data)
    - We also leave in all data needed for processing the two above (e.g., cross section heated data).
  - Ergo, there are at least 76 child nodes in each reaction’s cross section.
- Therefore, a processed protare file can be huge
  - For example, 1.3GB for ”n+U238”.
- We have done 3 things to speed up reading in the files (presented in the next few slides)
  - LLNL deterministic transport codes tells us that reading “summed” (discussed later) is now faster than our legacy binary formatted data.
  - Monte Carlo transport codes also read in protares in parallel

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

- We can store GNDS file in a 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)
```

```
<XYs1d interpolation="log-log">
        <axes>
          <axis index="1" label="energy_in" unit="MeV"/>
          <axis index="0" label="radius" unit="fm"/></axes>
        <values href="HDF#/dData" startIndex="1818252" count="166"/></XYs1d></scatteringRadius>
```

  - For details see Caleb Mattoon.

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

- 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.
  - They are still parsed by the XML parser but this is relatively fast.
  - For example, the three nodes in red below are not “parsed” into FUDGE or GIDI until accessed:

```
<XYs1d label="heated_000">
        <axes>…</axes>
        <values href="HDF#/dData" startIndex="2575918" count="181928"/></XYs1d>
<Ys1d label="MonteCarlo_000">
        <axes>…</axes>
        <values href="HDF#/dData" startIndex="2757846" count="482542"/></Ys1d>
<gridded1d label="MultiGroup_000">
        <axes>…</axes>
        <array shape="230" compression="flattened">
          <values valueType="Integer32" label="starts">0</values>
          <values valueType="Integer32" label="lengths">230</values>
          <values href="HDF#/dData" startIndex="3240619" count="230"/></array></gridded1d>
```

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

- 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 (and all reactions enabled). Otherwise, GIDI will recompute the multi-group summed data which takes much longer since many lazy-parsed nodes need to be parsed.

```
<applicationData>
    <institution label="LLNL::multiGroupReactions">
        <reaction label="total" ENDF_MT="1"> … </reaction></institution>
    <institution label="LLNL::multiGroupDelayedNeutrons">
        <products> … </products></institution>
    <institution label="LLNL::multiGroupDelayedNeutrons">
        <outputChannel genre="NBody"> … </outputChannel></institution></applicationData>
```

- We have put in an issue to make this part of GNDS 3.0.

EMU: Realization sampling

- Kyle Wendt has developed a python code to create realizations from mean and covariance data.
- The code is dubbed EMU (Evaluated Means and Uncertainties).
- EMU uses FUDGE to access GNDS data and process the realizations.
- Users at LLNL use EMU for sensitivity studies.
- EMU will be released on github.com.

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 6.2.0
    - 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 (especially FUDGE).
- All codes released under MIT license, except currently FUDGE BSD.
- For questions please contact beck6@llnl.gov, mattoon1@llnl.gov or gert1@llnl.gov

Issue with TENDL2021 “n + Be9” protare

```
MT  level       energy           QI
875     0          0.0  -1684000.0
876     1     745400.0  -2429400.0
877     2    1096000.0  -2780000.0
878     3    1365000.0  -3049000.0
879     4    3020000.0  -4704000.0
880     5    3906000.0  -5590000.0
881     6    4696000.0  -6380000.0
882     7    5076000.0  -6760000.0
883     8    6256000.0  -7940000.0
884     9    9599000.0 -11283000.0
885    10   10126000.0 -11810000.0
886    11     713000.0  -2397000.0
887    12    2316000.0  -4000000.0
888    13    1666000.0  -3350000.0
889    14     -18600.0  -1665400.0
Negative excitation level = -18600.0 for ZA = 4008 and levelIndex = 14 is not allowed
890    15     779600.0  -2463600.0
    Negative excitation level = -18600.0 for ZA = 4008 and levelIndex = 14 is not allowed
WARNING: read ENDF error: len( info.doRaise ) > 0
```

Issue with TENDL2021 “n + Be9” protare

```
ReactionSuite: n + Be9
    PoPs
        chemicalElements
            chemicalElement:
/reactionSuite/PoPs/chemicalElements/chemicalElement[@symbol='Be']
                Isotope Be8
                    WARNING: Discrete level 11 is out of order
                    WARNING: Discrete level 12 is out of order
                    WARNING: Discrete level 13 is out of order
                    WARNING: Discrete level 14 is out of order
                    WARNING: Discrete level 15 is out of order
...
```

I modified FUDGE to translate Be9(n,2n)Be8_e* in a better way - IMHO.

```
<reaction label="n + Be9" ENDF_MT="2">
<reaction label="2n + (Be8 -> 2He4)" ENDF_MT="875">
<reaction label="n + n + (Be8_e1 -> 2He4)" ENDF_MT="876">
<reaction label="2n + (Be8_e2 -> 2He4)" ENDF_MT="877">
<reaction label="2n + (Be8_e3 -> 2He4)" ENDF_MT="878">
<reaction label="2n + (Be8_e4 -> 2He4)" ENDF_MT="879">
<reaction label="2n + (Be8_e5 -> 2He4)" ENDF_MT="880">
<reaction label="2n + (Be8_e6 -> 2He4)" ENDF_MT="881">
<reaction label="2n + (Be8_e7 -> 2He4)" ENDF_MT="882">
<reaction label="2n + (Be8_e8 -> 2He4)" ENDF_MT="883">
<reaction label="2n + (Be8_e9 -> 2He4)" ENDF_MT="884">
<reaction label="2n + (Be8_e10 -> 2He4)" ENDF_MT="885">
<reaction label="2n + (Be8_e11 -> 2He4)" ENDF_MT="886">
<reaction label="2n + (Be8_e12 -> 2He4)" ENDF_MT="887">
<reaction label="2n + (Be8_e13 -> 2He4)" ENDF_MT="888">
<reaction label="2n + (Be8_e14 -> 2He4)" ENDF_MT="889">
<reaction label="2n + (Be8_e15 -> 2He4)" ENDF_MT="890">
<reaction label="Be10 + photon [inclusive]" ENDF_MT="102">
<reaction label="H1 + Li9 [inclusive]" ENDF_MT="103">
<reaction label="H2 + Li8 [inclusive]" ENDF_MT="104">
<reaction label="H3 + Li7 [inclusive]" ENDF_MT="105">
<reaction label="He4 + He6 [inclusive]" ENDF_MT="107">
```

Lawrence Livermore
National Laboratory