FUDGE: LLNL nuclear data infrastructure

Presented at CSEWG

Bret Beck

Nov 2021

LLNL-PRES-829231

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

Overview of FUDGE

- For Updating Data and Generating Evaluations (FUDGE)
- Use to read/write, plot, modify and process GNDS data
  - Processing for Monte Carlo and multi-group transport.
- User interface is python 3.6+.
  - Computationally intensive stuff written in C and C++.
    - E.g., heating cross sections, multi-grouping distributions.
    - Users do not interact directly with the C and C++ codes, the python interface handles that.
- FUDGE is like a toolkit but we are writing many scripts that are included in the bin directory
- FUDGE can convert ENDF-6 (and LLNL ENDL) files into GNDS, and GNDS to ENDF-6.

FUDGE processing example for transport codes

- FUDGE command to process a GNDS file for deterministic and Monte Carlo transport at 3 temperatures (MeV/k)

```
bin/processProtare.py -mc -mg -t 2.58522e-08 -t 1e-07 -t 1e-4 gnds.file.xml
```

- Output (i.e., processed) file contains
  - evaluation data
  - Reconstructed cross sections (if needed)
  - Coulomb + nuclear elastic μ cutoff (if needed)
  - average product data (needed for energy deposition - KERMA)
  - pdf/cdf data for distributions (e.g., P(E'|E))
  - heated cross sections (at 3 temperatures in example above)
  - Common grid heated cross section for Monte Carlo (ditto)
  - multi-group data (ditto)
  - TNSL data, if present, are processed

URR processing is done separately, still needs to be integrate in.

GNDS 2.0 support in FUDGE

- GNDS 2.0 has not been officially released but it is most likely complete except for a few minor requests.
- FUDGE is up-to-date to handle GNDS 2.0
  - Except for TNSL (see Caleb) and that's because the specification are still being tweaked.
- Internally, FUDGE looks a lot like GNDS 2.0
  - I will show examples later
- Since GNDS 2.0 is not finalized, FUDGE writes the format as '2.0.LLNL_4'.
  - We have generated '2.0.LLNL_4' file for ENDF/B-VIII.0 that the OECD/NEA will make available (see Michael Fleming)

FUDGE creates many “nodes” automatically

- For example, in GNDS 2.0 the top node “reactionSuite” has the following child nodes:

externalFiles	styles	PoPs
resonances	reactions	orphanProducts
sums	productions	incompleteReactions
fissionComponents	applicationData	

- All of these are automatically created by the “reactionSuite” constructor (i.e., “__init__”) except for the “resonances” node and I will try to fix that.
- All child nodes but “PoPs”, “resonances”, “sums” and “resonances” have an ”add” method to add an object.
- Adding an instance to reactions example
  - reactionSuite.reactions.add(reaction)

GNDS 2.0 documentation node support in FUDGE

- The documentation node in GNDS 2.0 allows one to record a lot of information (thanks mainly to Dave Brown).
- Child nodes are:

  authors, contributors, collaborations, dates, copyright, acknowledgements, keywords, relatedItems, title, abstract, body, computerCodes, experimentalDataSets, bibliography and endfCompatible.

- All child nodes are automatically created by the documentation constructor
- Next slide show how to populate title and abstract nodes.

Documentation example

```
from xData.Documentation import documentation

doc = documentation.Documentation( doi="12.23134/8821.322" )

doc.title.body = "This is the title."

doc.abstract.body = "Pythagoras' theorem is $a^2 + b^2 = c^2$."
doc.abstract.markup = "latex"

print( doc.toXML() )
```

```
<documentation doi="12.23134/8821.322">
  <title><![CDATA[This is the title.]]></title>
  <abstract markup="latex">
     <![CDATA[Pythagoras's theorem is $a^2 + b^2 = c^2$.]]></abstract></documentation>
```

Note, only nodes that have been populated are written to a file.

GNDS 2.0 map file supported in FUDGE and GIDI+

```
<map library="Example" format="2.0">

<protare projectile="n"  target="O16"  evaluation="fromJoe"
                         path="fromJoe/n-008_O_016.xml"/>
<protare projectile="n"  target="U235"  evaluation="fromJoe"
                         path="fromJoe/n-092_U_235.xml"/>

<protare projectile="n"  target="U235"  evaluation="Ian"
                         path="fromIan/n-092_U_235.xml"/>
<protare projectile="n"  target="U238"  evaluation="Ian"
                         path="fromIan/n-092_U_238.xml"/>

<TNSL projectile="n"  target="OinBeO"  evaluation="ENDF/B-8.0"
                         path="tsl/tsl-OinBeO.xml">
                         standardTarget="O16" standardEvaluation="ENDF/B-8.0"/>

<import production.map></map>
```

A map file supports nesting which is used by EMU.

LLNL also has multi-group boundaries and flux files

- Multi-group boundaries format uses GNDS <group> node to store the label and boundaries for a group.

```
<group label="LLNL_gid_23">
  <grid index="0" label="energy" unit="MeV" style="boundaries">
   <values>2.0908e-6 2.0908e-4 1.8817e-3 .010245 .07002 0.27097 .7527 15.754</values>
     </grid></group>
```

/path/to/FUDGE/bin/addMultigroup.py -h

- Flux stored as f(T,E,mu) using a GNDS 3d function.

```
<XYs3d label="LLNL_fid_1">
  <axes>
   <axis index="3" label="temperature" unit="MeV/k"/>
   <axis index="2" label="energy_in" unit="MeV"/>
   <axis index="1" label="mu" unit=""/>
   <axis index="0" label="flux" unit="1/s"/></axes>
  <XYs2d outerDomainValue="0.0">
   <Legendre outerDomainValue="0.0"><values>85</values></Legendre>
   <Legendre outerDomainValue="21.0"><values>85</values></Legendre></XYs2d></XYs3d>
```

/path/to/FUDGE/bin/addFlux.py -h

Plotting example for ‘n + U230’ via pyQt5 and matplotlib

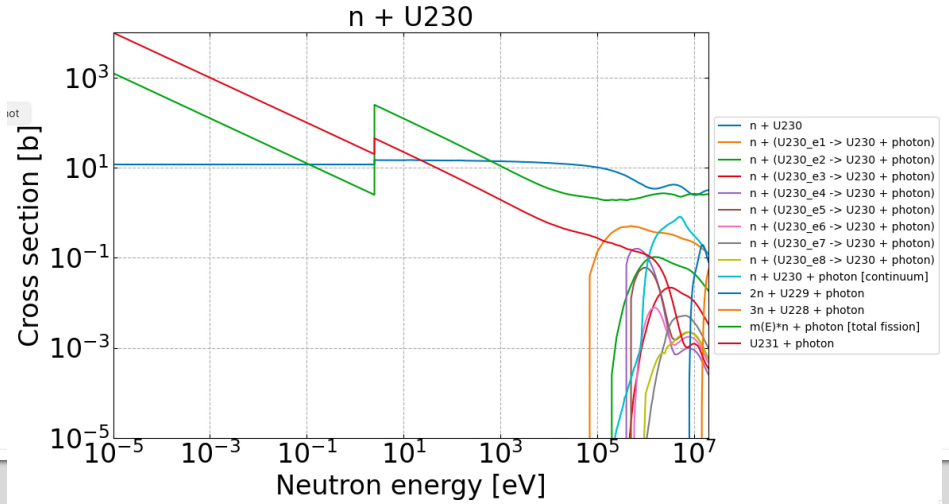
```
from fudge import map as mapModule

map = mapModule.Map.readXML( 'ENDF-VIII.0/all.map' )
protare = map.findAllOf('n', 'U230')[0].protare()

crossSections = []
for reaction in protare.reactions:
  crossSection = reaction.crossSection.toPointwiseLinear( lowerEps=1e-7 )
  crossSection.plotLegendKey = str( reaction )
  crossSections.append( crossSection )

crossSection.multiPlot( crossSections, rangeMin = 1e-5, xylog = 3,
  title='n + U230', xLabel='Neutron energy [eV]', yLabel='Cross section [b]' )
```

Plot is interactive



Check for missing gamma data in ENDF-VIII.0

```
from fudge import map as mapModule
from fudge.productData.distributions import unspecified

map = mapModule.Map.readXML( 'ENDF-VIII.0/all.map' )

neutrons = map.findAllOf( 'n' )   # First argument is projectile

for mapProtare in neutrons:
    protare = mapProtare.protare()
    for reaction in protare.reactions:
        for product in reaction.outputChannel.products:
            if product.id == 'photon':
                distribution = product.distribution[0]
                if isinstance(distribution, unspecified.form ):
                    print( product.toXLink() )
```

This example ignores nesting of products, but the code I used did include nesting.

FUDGE scripts

- We are developing scripts to make it easier to examine and process GNDS files.

- Some examples include:
  - processProtare.py: Processes for Monte Carlo and multi-group transport
  - peek.py: Prints each reaction and brief information about each reaction's products.
  - temperatures.py: List all temperature data in a GNDS file
  - checkGNDS.py: Check the physics in a GNDS file (e.g., positive cross sections, normalize distributions)
  - diffGNDS.py: Does a partial diff of two GNDS files
  - buildMapFile.py: Creates a map file from a list of GNDS files
  - checkMap.py: Check a map file and all it reference

Some of what are we working on in FUDGE

- Improving URR probability table building
  - We are working with BNL on this (Dave Brown and Matteo Vorabbi)
  - Currently, not automatic in FUDGE
- Adding a multi-group sums node to reduce load time for deterministic transport codes
  - Initially, this will go under the “applicationData” node
- Additional ACE support
  - Currently supports GNDS to ACE for neutrons transport including TNSL
- Update codes for GPUs where possible and beneficial (e.g., heating cross sections, calculating multi-group transfer matrices)

FUDGE refactoring

- We have been doing some needed refactoring of FUDGE
- Latest release includes most of the refactoring but still have more to do
- Hope to complete refactoring by January 2022 and do another release
- That said, anyone interested in FUDGE should get the latest release and play with it since a few small changes should be needed to update scripts for the January release

Other talks related to FUDGE

- Caleb Mattoon will be talking on TNSL data and FUDGE
  - Talk Wednesday at 10:00 am EST
  - “TNSL Implementation and Testing in FUDGE”

- Kyle Wendt will be talking on EMU
  - Talk today at 17:35 EST
  - “An Uncertainty Quantification Toolkit for GNDS Formatted Libraries”
  - Evaluated Means and Uncertainties (EMU)
  - Generates realizations from mean and covariance data
  - Uses FUDGE to read/write and modify data.
  - Also generates processed files

GIDI+

- A collection of packages for transport codes to read GNDS files
  - Mainly written in C++
- GIDI+ consists of the following packages:
  - PoPI: Property of Particles Interface
    - Reads PoPs data
  - GIDI: General Interaction Data Interface
    - Read in a GNDS reactionSuite file
    - Supports map files
    - Includes support for multi-group data including collapsing
    - Currently, only reads GNDS data needed by transport codes
  - MCGIDI: Monte Carlo GIDI
    - For use in Monte Carlo transport codes
    - Extracts needed data from a GIDI instance
    - Has cross section, energy deposition, etc. lookup functions by temperature and energy
    - Has reaction, distribution sampling functions.
  - plus other packages rarely called directly by transport codes

What are we working on in GIDI+

- Decreasing load times
  - ‘lazy reading’
  - Summed data for deterministic transport codes
- Direct sampling of elastic TNSL data
  - from S(E,T) for coherent elastic
  - from DebyeWaller - W’(T) – for incoherent elastic
- Updating the version in GEANT4
  - Current version is many years old and does not read GNDS
  - We have funding to work on improving diagnostic gammas
    - Dave Brown and BNL leading effort
- Add physical unit support
  - Already in FUDGE

Code releases

- We are releasing all codes under https://github.com/LLNL
  - FUDGE
    - https://github.com/LLNL/fudge
    - Version 5.0
    - Python 3.6+
    - Pip install – instructions on github site
    - BSD license (will switch to MIT license)
    - GNDS format is 2.0.LLNL_4 which should be the same as 2.0 if there are no further changes to GNDS 2.0 specifications (except for TNSL data)
  - GIDI+
    - https://github.com/LLNL/gidiplus
    - Version 3.22.??
    - C++11
    - GNDS format is 2.0.LLNL_4 (and 2.0?)
- Releasing all codes under MIT license, except currently FUDGE

Final comment

- Latest releases of FUDGE and GIDI+ will be available in a few days.
- Plan to do another release around 1 Jan. 2022.
- If you find any issues with FUDGE or GIDI+ please let us know
- We plan to also release EMU (see Kyle Wendt)
- We need you to look at the latest 2.0* including
  - The specification document
  - the NEA files
  - What FUDGE is doing (i.e., it is doing something wrong)

GIDI+ main user packages

- PoPI
  - Properties of Particle Interface
  - C++ API to read and allow access to GNDS PoPs data
- GIDI
  - General Interaction Data Interface
  - C++ API to read and access to GNDS data
  - Developed to give access for transport codes
  - Follows outline of GNDS
  - Has Map and Protare classes
- MCGIDI
  - Monte Carlo General Interaction Data Interface
  - C++ API for use in Monte Carlo transport codes
  - Extracts data from a GIDI::Protare
  - Stores data in more suitable way for Monte Carlo transport
  - Cross section look up by temperature and projectile energy for host code
  - Samples a reaction for a protare
  - Samples outgoing particle data for a reaction

GIDI+ (or gidiplus)

- To use GIDI and MCGIDI requires additional packages. GIDI, MCGIDI and these additional packages are dubbed GIDI+.
  - pugixml-1.8
    - Third party XML parser
    - Written in C++
  - statusMessageReporting
    - Handles message passing between C packages
    - Written in C
  - numericalFunctions
    - Supports 1d numerical functions including addition, multiplication
    - Written in C
  - PoPI
  - GIDI
  - MCGIDI

statusMessageReporting and numericalFunctions are also used by FUDGE.

PoPI C++ API

- Property of Particles Interface (PoPI)
- Implements the PoPs part of GNDS
- Uses strings for particle IDs as defined in GNDS
  - (e.g., “O16”, “n”, “U235”, “u235_e6”)
- Current LLNL PoPs files
  - pops.xml (currently only defines ground state nuclei)
  - metastables_alias.xml (e.g., “Am242_m1” for “Am242_e2”)
  - LLNL_alias.xml (e.g., “92235” for “U235”)

Simple PoPI example

```
PoPI::Database pops( "pops.xml" );
pops.addFile( "metastables_alias.xml" )

PoPI::Particle const &O16 = pops.get<PoPI::Particle>( "O16" );
            //                O16 = pops.particle( "O16" );

std::cout << "O16 -> " << O16_3.ID( ) << std::endl;
std::cout << "mass = " << O16_3.massValue( "amu" ) << std::endl;
```

```
O16 -> O16
mass = 15.9949
```

GIDI C++ API

- General Interaction Data Interface (GIDI)
- A C++ API for reading a GNDS reactionSuite.
  - Uses PoPI to read the PoPs part.
- Retrieving and collapsing multi-group data for use in deterministic codes (or Monte Carlo but that is better handled by MCGIDI).
- Protare is a virtual class. Actual classes are ProtareSingleton, ProtareComposite and ProtareTNSL.
- Support reading/writing GNDS 1.10 and 2.0(?) but, like FUDGE, uses 2.0 internally.

Simple GIDI example

```
GIDI::Map map( “test.map”, pops );

GIDI::Construction::Settings construction( GIDI::Construction::e_all,
                                           GIDI::Construction::e_nuclearAndAtomic );
GIDI::Protare *protare = map.protare( construction, pops, PoPI::IDs::neutron, "O16" );

GIDI::Styles::TemperatureInfos temperatures = protare->temperatures( );

GIDI::Settings::MG settings( protare->projectile( ).ID( ), label, true );
GIDI::Vector crossSection = protare->multiGroupCrossSection( settings, particles );
```

typedef std::vector<Styles::TemperatureInfo> TemperatureInfos;

```
(venv-3.7.2) # temperatures.py ~/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
/g/g16/beck6/Git/GIDI_plus/GIDI/Test/Data/MG_MC3Ts/neutrons/n-008_O_016.xml
  temperature 0.0 K: eval
       temperature [K]         heated           griddedCrossSection    URR_probabilityTables      heatedMultiGroup       SnElasticUpScatter
   ------------------------------------------------------------------------------------------------------------------------------------------
                 300.1       heated_000           MonteCarlo_000                                  MultiGroup_000
               1.16e+04       heated_001           MonteCarlo_001                                  MultiGroup_001
               1.16e+06       heated_002           MonteCarlo_002                                  MultiGroup_002
```

MCGIDI C++ API for GNDS

- Monte Carlo GIDI (MCGIDI)
- A C++ API for Monte Carlo transport codes
- Can do LLNL model A and B (MCNP) upscatter for outgoing particles
- Supports broadcasting for MPI and GPUs
- Implemented in LLNL's Monte Carlo transport code Mercury

Simple MCGIDI example

```
double energy = 14.1, crossSection, reactionCrossSection;

MCGIDI::DomainHash domainHash( 4000, 1e-8, 10 );
MCGIDI::Protare *MCProtare = MCGIDI::protareFromGIDIProtare( *protare, pops, MC
                        particles, domainHash, temperatures, reactionsToExclude );

int hashIndex = domainHash.index( energy );
crossSection = MCProtare->crossSection( URR_infos, hashIndex, temperature, energy );
reactionCrossSection = MCProtare->reactionCrossSection( ir, URR_infos, hashIndex,
                        temperature, energy );

int reactionIndex = MCProtare->sampleReaction( URR_infos, hashIndex,
                                temperature, energy, crossSection, rng, rngState );
reaction->sampleProducts( MCProtare, energy, input, products );
```