ENDF Python Interface

Release 0.1.1

Paul Romano

May 30, 2023

CONTENTS:

CHAPTER ONE

USER'S GUIDE

1.1 Getting Started

1.1.1 What is `endf`?

endf is a Python package for reading and interpreting ENDF-6 and ACE format nuclear data files. Compared to other packages that provide functionality for working with ENDF and ACE files, this package has numerous advantages:

- Easily installable through `pip`
- Thoughtful API design targeting Python first
- Offers both a low-level interface for working with the raw data in an ENDF file as well as a more intuitive high-level interface
- Fast file-loading performance thanks to optimized read routines
- Fully documented, tested, and type-hinted

1.1.2 Installation

The `endf` package can be installed by running:

```
python -m pip install endf
```

1.2 Detailed Usage

The *endf* package provides both low-level and high-level interfaces to ENDF files. The low-level interface can be used when you simply need access to the raw data in an ENDF file and you know exactly what you are looking for (usually with a copy of the ENDF-6 format manual open next to you). The high-level interface can be used if you are a normal human and you just want to, say, look at a cross section for a specific reaction.

1.2.1 Low-level Interface

To read an ENDF file with a single material, simply pass the filename to the `Material` class:

```
import endf

mat = endf.Material('n-092_U_235.endf')
```

The `section_text` attribute holds a dictionary mapping (MF, MT) to the section of text from the ENDF file:

```
>>> print(mat.section_text[12, 75])
 9.223500+4 2.330248+2          2          2         25          0922812 75
 4.386000+5 0.000000+0          0          0          3          1922812 75
 0.000000+0 1.000000+0 1.000000+0                                 922812 75
```

The `section_data` attribute also holds a dictionary with keys that are (MF, MT) pairs, but the values are dictionaries that hold the individual pieces of data from the ENDF file:

```
>>> mat.section_data[3, 16]
{'ZA': 92235,
'AWR': 233.0248,
'QM': -5298000.0,
'QI': -5298000.0,
'LR': 0,
'sigma': <Tabulated1D: 39 points, 1 regions>}
```

Tabulated1D Objects

As the above example shows, whenever a TAB1 record is encountered in an ENDF file, it is represented as a `Tabulated1D` object. The raw *x* and *y* values can be accessed through the `x` and `y` attributes. For example, if you wanted to make a log-log plot of a cross section:

```
import matplotlib.pyplot as plt

xs = mat.section_data[3, 16]['sigma']
plt.loglog(xs.x, xs.y, marker='.', label='(n,2n)')
plt.show()
```

The `Tabulated1D` acts like a function; you can pass it a given value of the independent variable and it will return the corresponding value of the dependent variable, interpolating between tabulated points where necessary. For example, if you wanted to know the cross section at 9.5 MeV:

```
>>> xs(9.5e6)
0.7428240476190476
```

Passing a list, array, or other iterable of values will return a `numpy` array of corresponding values:

```
>>> energies = numpy.linspace(6.0e6, 8.0e6, 10)
>>> xs(energies)
array([0.183841  , 0.25245628, 0.3152985 , 0.3550485 , 0.3947985 ,
       0.43081767, 0.463106  , 0.49255778, 0.52108125, 0.55290937])
```

1.2.2 High-level Interface

While this form is more useful, it still may be a little too "raw". The `Material` class has an `interpret()` method that returns a class based on the ENDF sublibrary type. For example, incident-neutron data will result in an instance of the `endf.IncidentNeutron` class. These "interpreted" classes provide a much more intuitive interface to data within an ENDF (or ACE) file.

Incident Neutron Data

The `IncidentNeutron` class provides the high-level interface to ENDF incident neutron sublibrary files. You can get an instance of this class either by calling the `endf.Material.interpret()` method:

```
mat = endf.Material('n-092_U_235.endf')
u235 = mat.interpret()
```

or by directly passing a filename to the `endf.IncidentNeutron.from_endf()` method:

```
>>> u235 = endf.IncidentNeutron.from_endf('n-092_U_235.endf')
>>> u235
<IncidentNeutron: U235, 85 reactions>
```

Most incident neutron data is collected into the `reactions` attribute, which is a dictionary that maps the MT value to an instance of the `Reaction` class:

```
>>> u235.reactions
{1: <Reaction: MT=1 (n,total)>,
 2: <Reaction: MT=2 (n,elastic)>,
 4: <Reaction: MT=4 (n,level)>,
 5: <Reaction: MT=5 (n,misc)>,
 16: <Reaction: MT=16 (n,2n)>,
 17: <Reaction: MT=17 (n,3n)>,
 18: <Reaction: MT=18 (n,fission)>,
 ...
 835: <Reaction: MT=835 (n,a35)>}
```

To look at data for a specific reaction then, you can index the `reactions` attribute:

```
>>> u235.reactions[16]
<Reaction: MT=16 (n,2n)>
```

Alteratively, you can pass the MT value as an index to `IncidentNeutron` directly:

```
>>> u235[16]
<Reaction: MT=16 (n,2n)>
```

or even use the name of the reaction as an index:

```
>>> u235['n,2n']
<Reaction: MT=16 (n,2n)>
```

The `Reaction` class has several attributes, including `xs` (cross section), `products` (reaction products), `q_reaction` (reaction Q-value) and `q_massdiff` (mass-difference Q value). The `xs` attribute is a dictionary mapping a temperature to the integral cross section:

```
>>> n2n = u235['n,2n']
>>> n2n.xs
{'0K': <Tabulated1D: 39 points, 1 regions>}
```

For data that originates from an ENDF file, the cross sections are always present at 0 K. However, the `IncidentNeutron` class can also be used for ACE data. In that case, cross sections can be present at temperatures other than 0 K.

The `products` attribute gives a list of reaction products as `Product` objects:

```
>>> n2n.products
[<Product: neutron, emission=prompt, yield=2.0>,
 <Product: U234, emission=prompt, yield=1.0>,
 <Product: photon, emission=prompt, yield=tabulated>]
```

The yield of a given product is accessed through the `yield_` attribute:

```
>>> photon = n2n.products[-1]
>>> photon.yield_
<Tabulated1D: 39 points, 1 regions>
```

1.2.3 ACE Files

Working with ACE files is conceptually similar to ENDF files. A low-level interface provides access to the raw data within an ACE file (the *NXS*, *JXS*, and *XSS* arrays) and the same high-level interface classes, e.g., `endf.IncidentNeutron` can be used to more easily inspect data. If you have an ACE file with a single table within it, you can load it with the `endf.ace.get_table()` function, which returns a `endf.ace.Table` object:

```
>>> table = endf.ace.get_table('80198.710nc')
>>> table
<ACE Table: 80198.710nc at 293.6 K>
```

Raw access to the underlying arrays in the ACE file is provided via the `nxs`, `jxs`, and `xss` attributes:

```
>>> table.nxs
array([     0, 172118,  80198,   3180,      34,      25,      97,       4,
            0,      0,     80,    198,       0,       0,       0,       0,
            0])
```

Note that each of these arrays is prepended with an extra zero at the beginning so that the indexing follows the Fortran 1-based indexing that is referenced in the ACE format manual.

As with the `Material` class, the `Table` class has a `interpret()` that returns a corresponding high-level interface class. For continuous-energy neutron data, this method will return an `IncidentNeutron` object:

```
>>> hg198 = table.interpret()
>>> hg198
<IncidentNeutron: Hg198, 38 reactions>
```

From this point, the interface follows exactly as is shown in *High-level Interface*.

CHAPTER

TWO

API REFERENCE

2.1 Low-level Interface

Material	ENDF material with multiple files/sections

2.1.1 endf.Material

class endf.**Material**(*filename_or_obj*)

ENDF material with multiple files/sections

Parameters

filename_or_obj (Union[str, bytes, PathLike, TextIO]) – Path to ENDF file to read or an open file positioned at the start of an ENDF material

MAT

ENDF material number

Type

int

sections

List of (MF, MT) sections

Type

List[Tuple[int, int]]

section_text

Dictionary mapping (MF, MT) to corresponding section of the ENDF file.

Type

dict

section_data

Dictionary mapping (MF, MT) to a dictionary representing the corresponding section of the ENDF file.

Type

dict

Methods

interpret()	Get high-level interface class for the ENDF material

endf.Material.interpret

`Material.interpret()`

Get high-level interface class for the ENDF material

Return type
`Any`

Returns

- *Instance of a high-level interface class, e.g.,*
- *endf.IncidentNeutron*.

get_materials	Return a list of all materials within an ENDF file.

2.1.2 endf.get_materials

`endf.get_materials`(*filename*)

Return a list of all materials within an ENDF file.

Parameters
filename (`Union[str, bytes, PathLike]`) – Path to ENDF-6 formatted file

Return type
`List[Material]`

Returns
A list of ENDF materials

2.2 High-level Interface

IncidentNeutron	Continuous-energy neutron interaction data.
Tabulated1D	A one-dimensional tabulated function.
Reaction	A nuclear reaction
Product	Secondary particle emitted in a nuclear reaction

2.2.1 endf.IncidentNeutron

`class endf.IncidentNeutron(`*atomic_number*, *mass_number*, *metastable=0*`)`

Continuous-energy neutron interaction data.

This class stores data derived from an ENDF-6 format neutron interaction sublibrary.

Parameters

- **`atomic_number`** (*`int`*) – Number of protons in the target nucleus
- **`mass_number`** (*`int`*) – Number of nucleons in the target nucleus
- **`metastable`** (*`int`*) – Metastable state of the target nucleus. A value of zero indicates the ground state.

`atomic_number`

Number of protons in the target nucleus

Type
int

`atomic_symbol`

Atomic symbol of the nuclide, e.g., ‘Zr’

Type
str

`mass_number`

Number of nucleons in the target nucleus

Type
int

`metastable`

Metastable state of the target nucleus. A value of zero indicates the ground state.

Type
int

`name`

Name of the nuclide using the GNDS naming convention

Type
str

`reactions`

Contains the cross sections, secondary angle and energy distributions, and other associated data for each reaction. The keys are the MT values and the values are Reaction objects.

Type
dict

Methods

from_ace(filename_or_table[, metastable_scheme])	Generate incident neutron continuous-energy data from an ACE table
from_endf(filename_or_mat)	Generate incident neutron data from an ENDF file

endf.IncidentNeutron.from_ace

`classmethod IncidentNeutron.from_ace`(*filename_or_table*, *metastable_scheme='mcnp'*)

Generate incident neutron continuous-energy data from an ACE table

Parameters

- **ace_or_filename** – ACE table to read from. If the value is a string, it is assumed to be the filename for the ACE file.
- **metastable_scheme** (*{'mcnp', 'nndc'}*) – Determine how ZAID identifiers are to be interpreted in the case of a metastable nuclide. Because the normal ZAID (=1000*Z + A) does not encode metastable information, different conventions are used among different libraries. In MCNP libraries, the convention is to add 400 for a metastable nuclide except for Am242m, for which 95242 is metastable and 95642 (or 1095242 in newer libraries) is the ground state. For NNDC libraries, ZAID is given as 1000*Z + A + 100*m.

Return type

IncidentNeutron

Returns

Incident neutron continuous-energy data

endf.IncidentNeutron.from_endf

`classmethod IncidentNeutron.from_endf`(*filename_or_mat*)

Generate incident neutron data from an ENDF file

Parameters

filename_or_mat (Union[str, bytes, PathLike, *Material*]) – Path to ENDF-6 formatted file or material object

Return type

IncidentNeutron

Returns

Incident neutron data

2.2.2 endf.Tabulated1D

`class endf.Tabulated1D`(*x*, *y*, *breakpoints=None*, *interpolation=None*)

A one-dimensional tabulated function.

This class mirrors the TAB1 type from the ENDF-6 format. A tabulated function is specified by tabulated (x,y) pairs along with interpolation rules that determine the values between tabulated pairs.

Once an object has been created, it can be used as though it were an actual function, e.g.:

```
>>> f = Tabulated1D([0, 10], [4, 5])
>>> [f(xi) for xi in numpy.linspace(0, 10, 5)]
[4.0, 4.25, 4.5, 4.75, 5.0]
```

Parameters

- **x** (*Iterable of float*) – Independent variable
- **y** (*Iterable of float*) – Dependent variable
- **breakpoints** (*Iterable of int*) – Breakpoints for interpolation regions
- **interpolation** (*Iterable of int*) – Interpolation scheme identification number, e.g., 3 means y is linear in ln(x).

x

Independent variable

Type
Iterable of float

y

Dependent variable

Type
Iterable of float

breakpoints

Breakpoints for interpolation regions

Type
Iterable of int

interpolation

Interpolation scheme identification number, e.g., 3 means y is linear in ln(x).

Type
Iterable of int

n_regions

Number of interpolation regions

Type
int

n_pairs

Number of tabulated (x,y) pairs

Type
int

Methods

from_ace(ace[, idx, convert_units])	Create a Tabulated1D object from an ACE table.
integral()	Integral of the tabulated function over its tabulated range.

endf.Tabulated1D.from_ace

classmethod Tabulated1D.**from_ace**(*ace*, *idx=0*, *convert_units=True*)

Create a Tabulated1D object from an ACE table.

Parameters

- **ace** (*openmc.data.ace.Table*) – An ACE table
- **idx** (*int*) – Offset to read from in XSS array (default of zero)
- **convert_units** (*bool*) – If the abscissa represents energy, indicate whether to convert MeV to eV.

Returns

openmc.data.Tabulated1D – Tabulated data object

endf.Tabulated1D.integral

Tabulated1D.**integral**()

Integral of the tabulated function over its tabulated range.

Returns

numpy.ndarray – Array of same length as the tabulated data that represents partial integrals from the bottom of the range to each tabulated point.

2.2.3 endf.Reaction

class endf.**Reaction**(*MT*, *xs=None*, *products=None*, *q_reaction=0.0*, *q_massdiff=0.0*, *redundant=False*)

A nuclear reaction

This class represents a single reaction channel for a nuclide with an associated cross section and, if present, a secondary angle and energy distribution.

Parameters

- **mt** (*int*) – The ENDF MT number for this reaction.
- **xs** (*dict*) – Microscopic cross section for this reaction as a function of incident energy; these cross sections are provided in a dictionary where the key is the temperature of the cross section set.
- **products** (*list*) – Reaction products
- **q_reaction** (*float*) – The reaction Q-value in [eV].
- **q_massdiff** (*float*) – The mass-difference Q value in [eV].
- **redundant** (*bool*) – Indicates whether or not this is a redundant reaction

MT

The ENDF MT number for this reaction.

Type
int

products

Reaction products

Type
list

q_reaction

The reaction Q-value in [eV].

Type
float

q_massdiff

The mass-difference Q value in [eV].

Type
float

redundant

Indicates whether or not this is a redundant reaction

Type
bool

xs

Microscopic cross section for this reaction as a function of incident energy; these cross sections are provided in a dictionary where the key is the temperature of the cross section set.

Type
dict

Methods

from_ace(table, i_reaction)	Generate incident neutron continuous-energy data from an ACE table
from_endf(MT, material)	Generate reaction from ENDF file

endf.Reaction.from_ace

classmethod Reaction.**from_ace**(*table*, *i_reaction*)

Generate incident neutron continuous-energy data from an ACE table

Parameters

- **table** (*Table*) – ACE table to read from
- **i_reaction** (int) – Index of the reaction in the ACE table

Returns
Reaction data

endf.Reaction.from_endf

`classmethod Reaction.from_endf(MT, material)`

Generate reaction from ENDF file

Parameters

- **MT** (int) – MT value of the reaction
- **material** (*Material*) – ENDF

Return type

Reaction

2.2.4 endf.Product

`class endf.Product(name='neutron', yield_=None, distribution=None, applicability=None)`

Secondary particle emitted in a nuclear reaction

Parameters

- **name** (str) – The particle type of the reaction product. Defaults to 'neutron'.
- **yield** – Yield of secondary particle in the reaction.
- **distribution** – Distributions of energy and angle of product.
- **applicability** – Probability of sampling a given distribution for this product.

`applicability`

Probability of sampling a given distribution for this product.

Type

Iterable of openmc.data.Tabulated1D

`decay_rate`

Decay rate in inverse seconds

Type

float

`distribution`

Distributions of energy and angle of product.

Type

Iterable of AngleEnergy

`emission_mode`

Indicate whether the particle is emitted immediately or whether it results from the decay of reaction product (e.g., neutron emitted from a delayed neutron precursor). A special value of 'total' is used when the yield represents particles from prompt and delayed sources.

Type

{'prompt', 'delayed', 'total'}

`name`

The particle type of the reaction product

Type

str

yield_

Yield of secondary particle in the reaction.

Methods

2.3 ACE File Interface

ace.Table	ACE cross section table
ace.TableType	Type of ACE data table.

2.3.1 endf.ace.Table

class endf.ace.**Table**(*name*, *atomic_weight_ratio*, *kT*, *pairs*, *nxs*, *jxs*, *xss*)

ACE cross section table

Parameters

- **name** (*str*) – Full ACE table identifier, e.g., '92235.70c'.
- **atomic_weight_ratio** (*float*) – Atomic mass ratio of the target nuclide.
- **kT** (*float*) – Temperature of the target nuclide in [MeV]
- **pairs** (*list of tuple*) – 16 pairs of ZAIDs and atomic weight ratios. Used for thermal scattering tables to indicate what isotopes scattering is applied to.
- **nxs** (*numpy.ndarray*) – Array that defines various lengths with in the table
- **jxs** (*numpy.ndarray*) – Array that gives locations in the `xss` array for various blocks of data
- **xss** (*numpy.ndarray*) – Raw data for the ACE table

data_type

Type of the ACE data

Type

TableType

temperature

Temperature of the target nuclide in [K]

Type

float

zaid

ZAID identifier of the table, e.g., 92235

Type

int

nxs

Array that defines various lengths with in the table

Type

numpy.ndarray

jxs

Array that gives locations in the `xss` array for various blocks of data

Type

numpy.ndarray

xss

Raw data for the ACE table

Type

numpy.ndarray

Methods

interpret(**kwargs)	Get high-level interface class for the ACE table

endf.ace.Table.interpret

`Table.interpret(**kwargs)`

Get high-level interface class for the ACE table

Parameters

****kwargs** – Keyword-arguments passed to the high-level interface class

Return type

Any

Returns

- *Instance of a high-level interface class, e.g.,*
- *endf.IncidentNeutron*.

2.3.2 endf.ace.TableType

`class endf.ace.TableType(value)`

Type of ACE data table.

Methods

from_suffix(suffix)	Determine ACE table type from a suffix.

endf.ace.TableType.from_suffix

classmethod TableType.**from_suffix**(*suffix*)

Determine ACE table type from a suffix.

Parameters
`suffix` (*str*) – Single letter ACE table designator, e.g., 'c'

Return type
TableType

Returns
TableType – ACE table type

ace.get_table	Read a single table from an ACE file
ace.get_tables	Get all tables from an ACE-formatted file.
ace.get_libraries_from_xsdir	Determine paths to ACE files from an MCNP xsdir file.

2.3.3 endf.ace.get_table

endf.ace.**get_table**(*filename*, *name=None*)

Read a single table from an ACE file

Parameters

- **filename** (*str*) – Path of the ACE library to load table from
- **name** (*str, optional*) – Name of table to load, e.g. '92235.71c'

Returns
endf.ace.Table – ACE table with specified name. If no name is specified, the first table in the file is returned.

2.3.4 endf.ace.get_tables

endf.ace.**get_tables**(*filename*, *table_names=None*, *verbose=False*)

Get all tables from an ACE-formatted file.

Parameters

- **filename** (*str*) – Path of the ACE library file to load.
- **table_names** (*None, str, or iterable, optional*) – Tables from the file to read in. If None, reads in all of the tables. If str, reads in only the single table of a matching name.
- **verbose** (*bool, optional*) – Determines whether output is printed to the stdout when reading a Library

`endf.ace.`**`tables`**

List of *`Table`* instances

Type

list

2.3.5 endf.ace.get_libraries_from_xsdir

`endf.ace.`**`get_libraries_from_xsdir`**(*path*)

Determine paths to ACE files from an MCNP xsdir file.

Parameters

`path` (`Union`[`str`, `bytes`, `PathLike`]) – Path to xsdir file

Return type

`List`[`Path`]

Returns

List of paths to ACE libraries

INDEX