# PyNE : Python For Nuclear Engineering

Anthony M. Scopatz[1], Paul K. Romano[2], Paul P.H. Wilson[3], Kathryn D. Huff[3]
[1] *University of Chicago,* [2] *Massachusetts Institute of Technology,* [3] *University of Wisconsin*
*pyne-dev@googlegroups.com*

## INTRODUCTION

PyNE, or 'Python for Nuclear Engineering'[1], is a nascent free and open source C++/Cython/Python package for performing common nuclear engineering tasks. This is intended as a base level tool kit - akin to SciPy or Biopython - for common algorithms in the nuclear science and engineering domain.

The remainer of this paper is composed of a discussion of the difficulties which prevented PyNE from being written earlier, a listing of the first cut capabilities, and a description of why PyNE has thus far been successful and what future features are currently planned.

## BACKGROUND

While PyNE may be considered long overdue from an external perspective, the nuclear industry poses a uniquely high barrier to entry for free software. That closed source solutions are often easier to develop is an irony that increases the novelty of this package.

An initial hurdle for PyNE is the myriad of arcane file formats (both plain text and binary) which have been industry standard since the 1960s. Though obtuse specifications are not solely a nuclear problem, the scale of the number of formats is much greater. Partly to blame is that some of these formats were enshrined in international law at a time when Fortran 2 was the language of choice.

Thus the emerging value of PyNE is that it allows new users to shortcut the tedious and error prone process of writing their own tools to parse these data and output file formats. The current unfortunate state of affairs is due in part to huge institutional inertia on the part of the maintainers of such formats. This manifests as a reluctance to develop or refactor new and existing codes in a modern open source manner. PyNE seeks to increase human efficiency via a shared set of solutions rather than having every developer around the world replicate the same parsing routines.

Another major challenge for the community of PyNE developers is maintaining the BSD license while explicitly avoiding any code which may be subject to export control. Many nuclear engineering codes are open source in the sense that the source code is distributed to developers who are then free to modify it. However, due to the fear of illicit use for the development of weapons, these same codes are then heavily export controlled and redistribution in any form is against their licenses and is often illegal.

Redistribution concerns are not limited to source code. Basic nuclear data, while fundamentally un-copywritable under Western jurisprudence, may also be considered sensitive and subject to export control laws. PyNE has developed a three-tiered strategy to alleviate the data burden of the individual user based on the level of openness of the data.

In spite of the above administrative concerns, PyNE's place in the nuclear software ecosystem requires that it have a general architecture. Large portions of the code base are written in pure C or C++ and are built as Python-independent shared libraries. This enables other, compiled nuclear engineering codes to leverage PyNE. Hence, the Cython layer has wrappers for C++ standard library containers (maps, sets, lists, etc) which implement the Python collections interface of the appropriate type. Because of the high degree of factorization in PyNE, these wrappers could easily be reused by other projects.

## CURRENT CAPABILITIES

The following is an overview of the current capabilities of PyNE. This covers briefly each major module and its relevance to nuclear engineering. This is not meant to be a tutorial. For such information please refer to the PyNE user's guide [1].

### Nuclide Naming

There are a plethora of ways to represent nuclide names. The `pyne.nucname` module may be used to convert between these various naming schemes. Currently the following naming conventions are supported: zzaaam, human readable names, MCNP, Serpent, NIST, and CINDER. This module may convert between any of them and is implemented in C.

### Basic Nuclear Data

The `pyne.data` module aims to provide quick access to very high fidelity nuclear data. Usually values are taken from the `nuc_data.h5` library. This library is generated by the new `nuc_data_make` utility at install time. Current data includes atomic masses, decay data, neutron scattering lengths, and simple cross section data. 63-group neutron cross sections, photon cross sections, and fission product yields are also added when CINDER is available on the machine of the user. This module is implemented in C++.

### The Material Class

Materials are the primary container for radionuclides throughout PyNE. They map nuclides to mass weights, though they also contain methods for converting to and from atom frac-

[1] http://pyne.github.com

tions. In many ways this class takes inspiration from numpy arrays and python dictionaries. Materials are implemented in C++ and support both text and HDF5 persistence. This class may be found within `pyne.material`.

### CCCC Formats

The `pyne.cccc` module contains a number of classes for reading various cross section, flux, geometry, and data files with specifications given by the Committee for Computer Code Coordination. The following types of files can be read using classes from this module: ISOTXS, DLAYXS and SPECTR, with plans to support BRKOXS, RTFLUX, ATFLUX, RZFLUX, and MATXS.

### ACE Format Cross Sections

One of the most common data formats used to represent pointwise linearly-interpolatable cross sections is the ACE (A Compact ENDF) format. This format was introduced by LANL for use in the MCNP [2] code and is now used by the Serpent [3] and OpenMC [4] Monte Carlo codes as well. While there are a variety of resources that allow a user to inspect and plot data directly from ENDF, there has generally been no good means of looking at processed data in the ACE format that is actually used in Monte Carlo simulations. As such, a module has been added to PyNE to parse and store data from ASCII or binary ACE files. A front-end graphical user interface is also in development that will give the user an easy interface to view and compare both cross sections and secondary energy and angle distributions. Figure 1 shows cross section data for Pu-239 being plotted within the GUI.

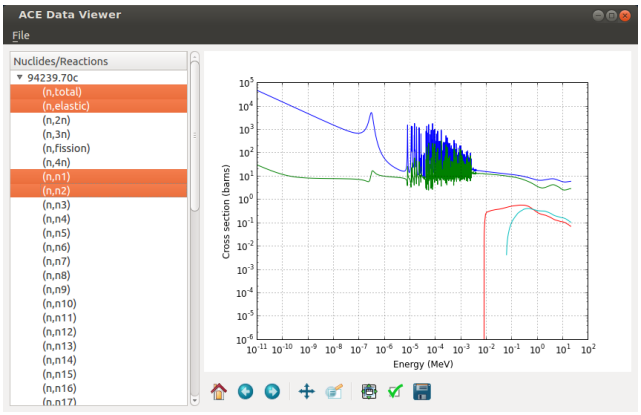


Fig. 1: Front-end graphical user interface for plotting ACE format data in PyNE.

### Cross Section Interface

The `pyne.xs` sub-package provides a top-level interface for computing (and caching) multigroup neutron cross sections. These cross sections will be computed from a variety of available data sources. Nominally these are stored in the `nuc_data.h5` HDF5 library which was generated by `nuc_data_make` at installation time. The library is searched for the following data sets in the following order of preference:

1. CINDER 63-group cross sections [5],
2. A two-point fast/thermal interpolation (using 'simple_xs' data from KAERI [6]),
3. or physical models implemented in this sub-package.

In the future, this package will support generating multigroup cross sections from user-specified pointwise data sources (such as ENDF or ACE files). In future versions of PyNE, this package may also include SCALE multigroup cross sections, if available [7].

### ORIGEN 2.2 Support

The `pyne.origen22` module provides an interface for reading, writing, and merging certain ORIGEN 2.2 [8] input and output files. Specifically, tapes 4, 5, 6, and 9 are currently supported. Other decks could be supported in the future with relative ease if the need or interest arises.

### Serpent Support

Serpent [3] is a continuous energy Monte Carlo reactor physics code. Pyne contains support for reading in Serpent's three types of output files: `*.res`, `*.dep`, and `*.det`. These are all in Matlab `*.m` format and are read in as Python dictionaries of numpy arrays and Materials. They may be optionally written out to corresponding `*.py` files and imported later.

### MCNP5 Support

MCNP5 [2] is a general purpose continuous energy Monte Carlo radiation transport code. Pyne contains support for reading some data from MCTAL files and for reading and writing surface source files (SSW/SSR). Additions are underway for reading all MCTAL data and mesh tally (MESHTAL) data.

### NJOY Module

Recently the developers of PyNjoy [9], a set of Python bindings for processing nuclear data using NJOY, agreed to have their work integrated with the PyNE project as a new NJOY module. This will help to encourage active development and wider participation in the project.

The `pyne.njoy` module provides a simple method of generating nuclear data in a variety of formats from raw ENDF data without the user having to worry about the exact format of the NJOY input files. Thus, with a single Python script, it is possible to process thousands of nuclides at once rather than managing many NJOY input files.

## CONCLUSIONS

Over the past year, the PyNE development team has formed a community which has successfully overcome many initial hurdles. In addition to the regular burden of starting a new open source project, the nuclear domain contains special concerns about export control and non-proliferation. By maintaining a conservative development strategy, only methods and data which are truly open make it into the PyNE code base. Even with these restraints the number of tools possible is very large.

With over 22000 lines of C++, Cython, and Python code, PyNE has just begun to implement what is in its purview. The more modules that PyNE has, the more utilities may be composed using it. Noting the possibility for 'scope explosion,' the development team is committed to adhering to modern software quality development standards. At a minimum, all contributions to PyNE must be tested, documented, and follow the official Python coding standard.

Future work may include more NJOY-like capabilities, a fully-featured MCNP interface, a common nuclear materials database, various infinite medium diffusion solvers, deep geologic repository modeling functionality, and Bateman equation solvers.

## ACKNOWLEDGEMENTS

The authors would like to recognize additional code contributions by Christopher Dembia, Robert Flanagan, and Eric Relson. Moreover, we would like to thank Seth Johnson, Joshua Peterson, Rachel Slaybaugh, Nick Touran, and Morgan White for their inspiration, guidance, and testing.

## REFERENCES

1. A. SCOPATZ et al., "PyNE User's Guide", 2012, http://pyne.github.com/usersguide/.
2. X-5 MONTE CARLO TEAM, "MCNP - A General Monte Carlo N-Particle Transport Code, Version 5", LA-UR-03-1987, Los Alamos National Laboratory (2008).
3. J. LEPPÄNEN, "Development of a New Monte Carlo Reactor Physics Code", P640, VTT Technical Research Centre of Finland (2007).
4. P. K. ROMANO and B. FORGET, "The OpenMC Monte Carlo Particle Transport Code", *Ann. Nucl. Energy*, Accepted.
5. W. B. WILSON, S. T. COWELL, T. R. ENGLAND, A. C. HAYES, and P. MOLLER, "A Manual for CINDER'90 Version 07.4 Codes and Data", LA-UR-07-8412, Los Alamos National Laboratory (2008).
6. KOREA ATOMIC ENERGY RESEARCH INSTITIUTE, "Table of Nuclides", 2012, http://atom.kaeri.re.kr/.
7. S. M. BOWMAN, "SCALE 6: Comprehensive Nuclear Safety Analysis Code System", *Nucl. Technol.*, **174**, 126 (2011).
8. A. G. CROFF, "ORIGEN2: A Versatile Computer Code for Calculating the Nuclide Compositions and Characteristics of Nuclear Materials", *Nucl. Technol.*, **62**, 335 (1983).
9. A. HÉBERT, "Towards DRAGON Version4", *Topical Meeting on Advances in Nuclear Analysis and Simulation* (2006).