PyNE Progress Report

Cameron R. Bates[1,2], Elliott Biondo[3], Kathryn Huff[2], Kalin Kiesling[3], Anthony Scopatz[3]

Robert Carlsen[3], Andrew Davis[3], Matthew Gidden[3], Tim Haines[3], Joshua Howland[2], Blake Huff[2], Kevin Manalo[4], Arielle Opotowsky[3], Rachel Slaybaugh[2], Eric Relson[3], Paul Romano[5], Patrick Shriwise[3], John D. Xia[6], Paul Wilson[3], and Julie Zachman[3]

[1] *Lawrence Livermore National Laboratory, 7000 East Ave L-188, Livermore, CA 94550*
[2] *The University of California, Berkeley, 2521 Hearst Ave, Berkeley, CA 94709*
[3] *The University of Wisconsin-Madison, 1500 Engineering Drive, Madison, WI 53706*
[4] *Georgia Institute of Technology, 770 State Street, Atlanta, GA 30332*
[5] *Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139*
[6] *University of Chicago, 5747 S. Ellis Ave., Jones 311, Chicago, IL 60637*
bates26@llnl.gov

INTRODUCTION

PyNE is a suite of free and open source (BSD licensed) tools to aid in computational nuclear science and engineering. PyNE seeks to provide native implementations of common nuclear algorithms, as well as an interface for the scripting language Python and I/O support for industry standard nuclear codes and data formats. In the past year PyNE has added many features including a Rigorous 2-step Activation workflow (R2S) [1], Direct Accelerated Geometry Monte Carlo (DAGMC) ray tracing [2], Consistent Adjoint-Weighted Importance Sampling (CADIS) variance reduction [3], and expanded ENSDF parsing support. As a part of our ongoing efforts to implement a verification and validation framework we also added continuous integration using the Build and Test Lab [4] at the University of Wisconsin. The PyNE development team has also improved PyNE's ease of use by making binaries available for Windows, Mac, and Linux through the conda package manager as well as adding Python 3 support.

FEATURE ENHANCEMENTS

Mesh

As of v0.4, PyNE includes a mesh representation interface that is used to build up geometries, store materials, and solve spatial differential equations. This is implemented as a layer on top of MOAB meshes [5]. In addition to the PyTAPS interface [6], a Python interface to interact with MOAB mesh objects, it also adds PyNE `Material` objects, which allow the user to define a mix of multiple isotopes, to volume elements as well as a generic tagging interface. These features together form a generic, easy-to-use mesh library that is capable of handling a plethora of nuclear engineering problems.

The `Mesh` class lives in the `pyne.mesh` module. This class houses an iMesh instance called `mesh` which comes from PyTAPS and contains methods for native mesh operations. The `mats` attribute is an instance of a PyNE `MaterialLibrary`. This is a mapping of volume element handles to `Material` objects. Tags—sometimes known as fields—are accessible as attributes on the mesh object itself.

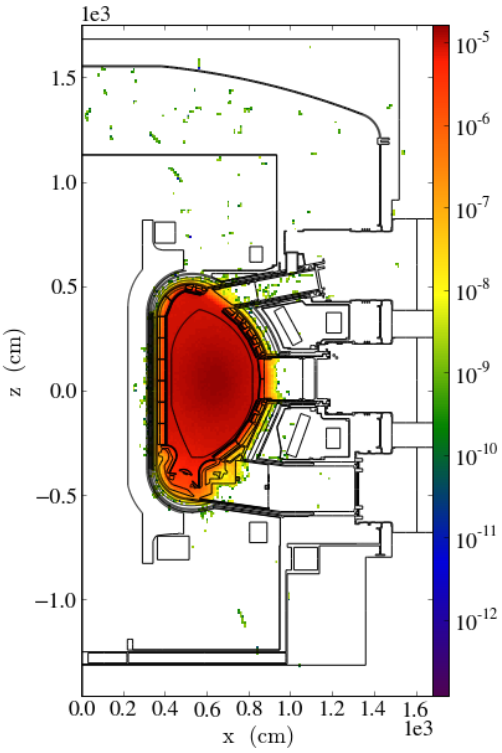


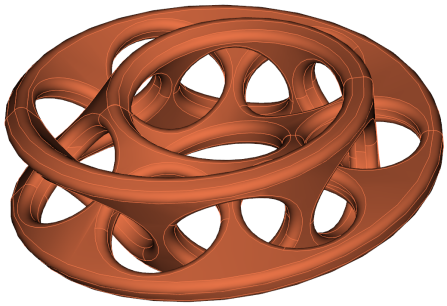
Fig. 1. A 2-D slice of a 3-D PyNE flux mesh of ITER plotted in yt. *This model is for demonstration purposes only.*

There are several different types of tags (`IMesh`, `Material`, `Metadata`, `Computed`) depending on where the data should be stored. All tag types expose the same interface.

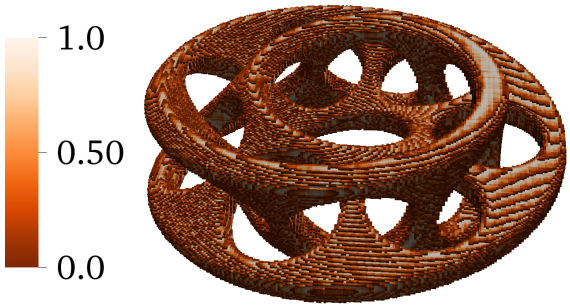
To do volumetric analysis and visualization, the `Mesh` class is natively supported by the yt project [7]. An example of the use of this mesh to analyze neutron flux in ITER is shown in Fig. 1.

DAGMC Module

Direct Accelerated Geometry Monte Carlo is a component of MOAB that facilitates Monte Carlo ray tracing on



(a) CAD model of a complex geometry [8].



(b) Volume fractions of the geometry within the mesh volume elements of an overlaid 200 x 100 x 200 Cartesian mesh.

Fig. 2. A CAD geometry and a discretized representation created using PyNE `discretize_geom()`. *This model is for demonstration purposes only.*

CAD geometries [2]. A `dagmc` module has been added to PyNE, providing a Python interface to these ray tracing capabilities. The `dagmc.discretize_geom()` function accomplishes the common task of mapping geometry cells onto a Cartesian or tetrahedral mesh. The `cell_fracs_to_mats` method of the `Mesh` class can be used to seamlessly create PyNE `Mesh` objects tagged with materials, where the materials are mixtures of the contributions from the various geometry cells found in each mesh volume element. This especially useful for discretizing CAD geometries onto grids for deterministic methods. An example of this is shown in Fig. 2.

ALARA Module

ALARA is a nuclear inventory analysis code developed at University of Wisconsin - Madison [9]. A module has been added to PyNE to facilitate the generation of ALARA input and the parsing of ALARA output. PyNE `Mesh` objects tagged with flux and material data can be used to generate input. The compositions of activated materials as calculated by ALARA can also be read back into a PyNE `Mesh` object. These components can be used in mesh-based activation and burn-up workflows.

R2S Activation Workflow

The Rigorous Two-Step (R2S) method is used to estimate the shutdown dose rate (SDDR) in fusion systems from photons born from neutron activation products [10]. This method involves separate neutron and photon transport simulations, coupled to a dedicated nuclear inventory analysis code. The PyNE `R2S` module implements a mesh-based R2S method and accomplishes this coupling in-memory by leveraging the PyNE `mesh`, `material`, `dagmc`, `mcnp`, and `alara` modules. The `R2S` module currently only supports transport with MCNP and nuclear inventory analysis with ALARA, but support for additional physics codes is planned. Mesh-based photon source sampling is accomplished within MCNP by compiling MCNP against a custom source sampling library within PyNE.

CADIS Variance Reduction

The Consistent Adjoint-Weighted Importance Sampling and the Forward-Weighted CADIS (FW-CADIS) method are hybrid Monte Carlo variance reduction techniques that use deterministic estimates of the forward and adjoint flux to generate Monte Carlo weight windows and source biasing parameters [3]. A mesh-based implementation of this method has been added to the PyNE `variancereduction` module. Work is currently underway to interface with the Denovo [11] deterministic transport code to in order to acquire these deterministic fluxes.

Tally Class

One common requirement in processing the output of MCNP and other nuclear engineering codes is to keep track of multiple tallies. PyNE has added a C++ class with a Python interface that assists users in keeping track of tally data. The PyNE development team is working to add saving, loading, and manipulation of these tally objects in the future.

Fluka Module

The `fluka` module is designed to parse the output files from FLUKA, a fully integrated particle physics Monte Carlo simulation package[12]. Currently, this module only supports the parsing of USRBIN output files, which is a file similar to a meshtal file in MCNP in that it tracks a certain quantity over an evenly spaced volume mesh. This module parses the USRBIN files and saves the tracked data and percent error data as attributes of a `Mesh` object.

Amalgamation

While PyNE is ostensibly a Python-oriented toolkit over two-thirds of the code base is written in C++. This makes it possible to use many of the features of PyNE without needing Python. In order to simplify the use of PyNE's C++ API we have added the ability to amalgamate all of the C++ code into a single source and header file. This makes it possible to add these two files to any project in order to use much of the functionality in PyNE without having to worry about linking multiple libraries in a separate location. This is used in Cyclus [13] to use PyNE features without adding Python as a dependency.

ENSDF Improvements

Previous versions of PyNE have included some ENSDF parsing capabilities. These have been focused on extracting half-lives and branching ratios of metastable and ground states. This has been vastly expanded to support the parsing of most ENSDF record types and to make level structure and decay data available in PyNE's C++/Python nuclear data interface. This makes it possible to use PyNE to look up most structure and decay data similar to online tools such as NuDat `http://www.nndc.bnl.gov/nudat2/`.

We have broken down the data from ENSDF into six distinct subsets. These include: excited level data, decay normalization data, gamma-ray data, alpha decay data, β^- decay, and electron capture/β^+ decay. In addition to information about the radiations for all decay transitions listed in ENSDF we have also included atomic data from the National Nuclear Data Center to calculate X-ray emissions from conversion electrons in gamma-ray emission and electron capture decay. Work is underway to add decay data access to the PyNE `Material` object to facilitate generation of complex sources for use in Monte Carlo transport codes.

Fission Yield Data

The latest release of PyNE includes two different sets of fission yield data. The first is the IAEA WIMSD library which provides fission product yields based on ENDF/B-VI. The second is from the IAEA Safeguards data library and includes independent fission yields with thermal, fast, and 14-MeV neutrons for ^{232}Th, ^{233}U, ^{235}U, ^{239}Pu, and ^{241}Pu.

VERIFICATION AND VALIDATION

The PyNE development team is working to implement documented verification and validation as a part of our basic development process. This has included: ensuring all code changes to PyNE have at least one reviewer who was not an author, requiring unit tests for all code additions, a coding style guide, and requiring all tests to pass on continuous integration builds before merging code changes. This issue will be addressed in more detail in other concurrent publications.

USABILITY ENHANCEMENTS

Installation Improvements and Binary Distributions

At our first PyNE workshop a significant amount of the instructional time was devoted to PyNE installation and configuration. This reduced the amount of material we were able to cover significantly. From this experience the PyNE development team came to the conclusion that a major focus of our version 0.4 development efforts should be on making the installation process simpler for non-developers. The core of this effort is based around the conda package manager. Conda is an open source package manager that is capable of managing packages on Windows, Mac and Linux. This makes it possible to use a single package manager across all platforms. We developed a standard conda package script which can be found at `https://github.com/conda/conda-recipes`. This automates the installation of dependencies for PyNE, significantly reducing the difficulty of building and installing the software. In addition, we used this package script to automate the production of binary packages for Linux and Mac. Finally, we developed a custom Windows build environment to build a distributable Windows binary.

Python 3 Support

PyNE was originally developed for the Python 2 interpreter as it was and still is the most common version of Python used in scientific computing. Python 3 is slowly starting to replace it, however, as the default Python version. With this changeover on the horizon the PyNE development team made an effort in preparation for the v0.4 release to make PyNE compatible with both versions. PyNE is now built and tested on Python 3 on a regular basis.

CULTIVATION OF USERS AND DEVELOPERS

Computational toolkits in the sciences grow more robust by leveraging a broad user base who test core capabilities with each use. Similarly, such toolkits grow more powerful by a broad developer base that serves the community by contributing new, research-relevant features. Development of a sustainable user and developer community is therefore integral to the success of the PyNE toolkit. To this end, the development team has organized tutorials to reach out to new users and has sought out support development by graduate students.

A tutorial was organized in November at the University of California, Berkeley to both reach out to new users and to gather feedback on the user experience of PyNE. Over a dozen researchers attended. The audience included undergraduate and graduate students in nuclear engineering as well as post-docs and faculty. In a six hour workshop, the attendees installed PyNE and ran prepared examples with the help of members of the development team. In addition to demonstrating the core data manipulation capabilities of the PyNE toolkit, the workshop included a reflective period in which attendees had the opportunity to brainstorm and suggest extensions, features, and improvements for the toolkit that were of interest in the context of their particular research.

Based on the success of this event and the organic growth of our user base, a second user workshop was conducted at the 18th Topical Meeting of the Radiation Protection and Shielding Division of ANS in September 2014. We plan to hold more of these in the future.

The development team has also conducted development sprints at both the University of California and the University of Wisconsin to cultivate the developer communities that have arisen in those institutions. These sprints allow the diverse and geographically dispersed development team to gather and collaborate on code contributions in a coherent manner.

In order to encourage young researchers to become involved in scientific computing, a number of desired PyNE

extensions have been defined online. These short descriptions of desired extensions can be found on the PyNE website and are intended to guide the contributions of young researchers. By defining relevant independent contributions with realistic scope, these descriptions provide an opportunity for a beginner developer to contribute code in a guided manner and will assist their transition from user to developer.

CONCLUSIONS

In the past year the PyNE development team has worked to improve PyNE's usability in addition to adding new features. The availability of binaries for stable releases has made PyNE more accessible to those who are users but not developers. The PyNE project will continue to create free and open source tools that easily interface with the plethora of choices available in nuclear engineering and scientific computing.

LLNL-ABS-656040

REFERENCES

1. E. BIONDO, A. DAVIS, A. SCOPATZ, and P. P. H. WILSON, "Rigorous Two-Step Activation for Fusion Systems with PyNE," in "Proc. of the 18th Topical Meeting of the Radiation Protection & Shielding Division of ANS," Knoxville, TN (2014).
2. T. J. TAUTGES, P. P. H. WILSON, J. KRAFTCHECK, B. F. SMITH, and D. L. HENDERSON, "Acceleration Techniques for Direct Use of CAD-Based Geometries in Monte Carlo Radiation Transport," in "International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)," American Nuclear Society, Saratoga Springs, NY (2009).
3. A. HAGHIGHAT and J. C. WAGNER, "Monte Carlo Variance Reduction with Deterministic Importance Functions," *Progress in Nuclear Energy*, **42**, *1*, 25–53 (2003).
4. UW BATLAB TEAM, "BaTLab," `https://batlab.org` (2014).
5. T. J. TAUTGES, R. MEYERS, K. MERKLEY, C. STIMPSON, and C. ERNST, "MOAB: A Mesh-Oriented Database," SAND2004-1592, Sandia National Laboratories (Apr. 2004), report.
6. J. PORTER, "PyTAPS v1.4 documentation," `http://pythonhosted.org/PyTAPS/` (2011).
7. M. J. TURK, B. D. SMITH, J. S. OISHI, S. SKORY, S. W. SKILLMAN, T. ABEL, and M. L. NORMAN, "yt: A Multi-code Analysis Toolkit for Astrophysical Simulation Data," *Astrophysical Journal, Supplement*, **192**, 9 (Jan. 2011).
8. S. NYBERG, "Self Intersecting Möbius Chain," `http://http://grabcad.com/library/self-intersecting-mobius-chain-1`, accessed: 2014-06-22.
9. P. P. H. WILSON, H. TSIGE-TAMIRAT, H. Y. KHATER, and D. L. HENDERSON, "Validation of the ALARA activation code," *Fusion Technology*, **34**, 784–788 (1998).
10. Y. CHEN and U. FISCHER, "Rigorous MCNP Based Shutdown Dose Rate Calculations: Computational Scheme, Verification Calculations and Application to ITER," *Fusion Engineering and Design*, **63-64**, 107–114 (2002).
11. T. M. EVANS, A. S. STAFFORD, R. N. SLAYBAUGH, and K. T. CLARNO, "Denovo: A New Three-Dimensional Parallel Discrete Ordinates Code in SCALE," *Nuclear Technology*, **171**, 171–200 (2010).
12. G. BATTISTONI, S. MURARO, P. SALA, F. CERUTTI, S. FERRARI, A. ROESLER, A. FASSO', and J. RANFT, "The FLUKA code: Description and benchmarking," *AIP Conference Proceeding*, **896**, 31–49 (2007).
13. R. W. CARLSEN, M. GIDDEN, K. HUFF, A. C. OPOTOWSKY, O. RAKHIMOV, A. M. SCOPATZ, Z. WELCH, and P. WILSON, "Cyclus v1.0.0," (Jun. 2014).