Nuclear Science and Engineering

ISSN: 0029-5639 (Print) 1943-748X (Online) Journal homepage: https://www.tandfonline.com/loi/unse20

serpentTools: A Python Package for Expediting Analysis with Serpent

Andrew E. Johnson, Dan Kotlyar, Stefano Terlizzi & Gavin Ridley

To cite this article: Andrew E. Johnson, Dan Kotlyar, Stefano Terlizzi & Gavin Ridley (2020): serpentTools: A Python Package for Expediting Analysis with Serpent, Nuclear Science and Engineering, DOI: 10.1080/00295639.2020.1723992

To link to this article: https://doi.org/10.1080/00295639.2020.1723992

NUCLEAR SCIENCE AND ENGINEERING

DOI: https://doi.org/10.1080/00295639.2020.1723992

serpentTools: A Python Package for Expediting Analysis with Serpent

Andrew E. Johnson,[a]* Dan Kotlyar,[a] Stefano Terlizzi,[a] and Gavin Ridley[b]

[a]*Georgia Institute of Technology, Atlanta, Georgia 30316*
[b]*Massachusetts Institute of Technology, Cambridge, Massachusetts 02139*

Received November 15, 2019
Accepted for Publication January 28, 2020

Abstract — *The* `serpentTools` *Python package is presented as a useful and efficient alternative for processing Serpent results. One positive attribute of Serpent is that many output files are exported directly in a MATLAB format, allowing for results to be loaded with minimal to no effort. However, some files for larger analyses may require immense amounts of memory to load and store all the data, leading to long wait times. To expedite the process of data handling and ease common analyses, the Computational Reactor Engineering lab at the Georgia Institute of Technology has released and is maintaining the* `serpentTools` *Python package: a set of data parsers and containers intended to streamline analysis with Serpent outputs. The parsers are capable of processing large outputs with ease, and yield all data to the user in a simple object-oriented framework. Data can be read into Python in comparable or better times than MATLAB, with the option to store only data needed for a specific purpose. Furthermore, common analyses are implemented directly into the package to expedite frequent analysis, including plotting meshed data and flux specta.* `serpentTools` *is designed to be a useful and practical manner by which the Serpent community can load and analyze data inside a Python environment. This paper presents the Python package, highlighting some basic features, and compares capabilities to similar platforms.*

Keywords — *Python, Serpent, Open source, Visualization.*

Note — *Some figures may be in color only in the electronic version.*

I. INTRODUCTION

The Serpent Monte Carlo code[1] has grown in popularity over recent years, both as an academic and research tool and one capable of production applications. The main Serpent website indicates that over 900 users across 200 research organization and universities utilize Serpent.[a] The strong emphasis on reactor physics, specifically spatial homogenization,[2] depletion, and multiphysics[3] round out a wide set of capabilities contained in a single code. More recent features have enabled analysis into adjoint-weighted sensitivities[4] and stability of depletion schemes.

An important benefit of Serpent is its convenient and organized output file structure. For each of the physics enabled (e.g., depletion, detectors, or cyclewise values like Shannon entropy), a dedicated MATLAB (Ref. 5) output file is generated. This eliminates the requirement for additional parsing utilities and allows researchers and engineers to easily process the data with MATLAB, where the user has access to a wide range of linear algebra routines, numeric modeling, and plotting capabilities.

For simple problems, the computational resources required to load output files may be negligible. As the simulation becomes more complicated, through increased homogenization sets, depletion zones, and/or burnup steps, the size of the output files can reach hundreds of megabytes of MATLAB scripts. Such files can be

*E-mail: ajohnson400@gatech.edu
[a] http://montecarlo.vtt.fi/users.htm.

prohibitively large for sharing and downloading, and require long loading times. Depending on file size and computing environment, there may be increased likelihood of crashes due to insufficient memory. Furthermore, MATLAB licenses can be costly for large organizations, and the free and open source alternative Octave[6] suffers from similar loading issues.

To improve the scalability issues and provide an open source method for loading and examining Serpent data, the Computational Reactor Engineering lab at the Georgia Institute of Technology introduces the `serpentTools` Python package. Designed for ease of use, flexibility, and speed, serpentTools offers a suite of custom parsers and objects for storing each of the unique outputs generated by Serpent. `serpentTools` requires the `NumPy`, `matplotlib`, and optionally, `SciPy` Python packages as dependencies.[7–9] Between these packages, nearly all of the linear algebra and plotting utilities found in MATLAB and Octave are provided to the user. The `serpentTools` package provides powerful, publication-quality plots with a few commands, often with little to no Python experience required. Additionally, the user is able to filter information during the reading process, leading to smaller memory footprints as only the desired information is retained.

This paper is laid out as follows. Section II provides an overview of the project, including links to relevant documentation and the data model used throughout the project. Examples that include plotting multiplication factors and meshed tally data using the Application Programming Interface (API) are demonstrated in Sec. III. Section IV compares `serpentTools` to alternative utilities, namely MATLAB and `PyNE` (Ref. 10).

II. OVERVIEW

`serpentTools` is an open-source, Massachusetts Institute of Technology (MIT)–licensed Python package containing parsing utilities and objects intended to expedite analysis with Serpent. Links to source code and installation guides are provided in Table I. The `serpentTools` data model is presented in Sec. II.A, making the case for an object-oriented approach. An overview of the control afforded to the user is included in Sec. II.B.

It is likely that many users and organizations have developed their own tools for pulling Serpent data into Python or another programming language. These tools may be "purpose-built" and difficult to apply to repeated problems or analyses. This paper does not claim to be the first of its kind, but such tools have not been widely published nor disseminated throughout the scientific body. `serpentTools` does seek to be a reliable and reproducible tool, providing researchers and organizations a transparent API for interacting with their data.

TABLE I

Links to Key `serpentTools` Resources

Subject	Link
Source code	https://github.com/CORE-GATECH-GROUP/serpent-tools
Full documentation	https://serpent-tools.readthedocs.io/en/master/

It is worth noting that `serpentTools` does not provide detailed analysis tools, such as interfaces for nodal diffusion codes or systems analysis. Instead, users will find a consistent API that can readily be integrated into existing workflows. Users can also take advantage of performance improvements (Sec. IV.A), filtering capabilities (Sec. II.B), and visualization methods (Sec. III.B) integrated directly into `serpentTools`.

II.A. Data Model

First and foremost, `serpentTools` provides an easy way to view and manipulate Serpent data with Python. Much like MATLAB users can load up the environment and data using the MATLAB run command, `serpentTools` users can simply use the `serpentTools.read` function. Both functions require the name of the output file and provide the user with all of the data. Inside MATLAB, users can manipulate the data using native matrix multiplication and linear algebra operations. These operations are instead left to the `NumPy` and `SciPy` packages, a pair of highly stable, robust, and fully featured Python packages for science and engineering.

`serpentTools` provides an object-oriented interface to the data, rather than simply a collection of arrays. The object returned from `serpentTools.read` is specific to the type of file being read. A result file yields a `ResultsReader` instance, a detector (tally) file produces a `DetectorReader` instance, and so on. Each of the primary plain-text output files have a corresponding `Reader` object tasked with parsing their file and storing data in appropriate objects.

By the design of the primary output file, values are appended as rows in matrices like `ABS_KEFF` and `INF_TOT`. If a user wanted to compare the contents of

two result files and loaded them into the same MATLAB workspace sequentially, the data from the second run would be appended into the data from the first, requiring some investigation into where the first ends and the second begins. For other MATLAB outputs, the data from the first file would be overwritten and removed upon reading the second file. With `serpentTools`, the data are segregated into two unique `Reader` instances. This eliminates name collision, where loading an output file may overwrite or modify data from a similar yet unique output file in the same workspace.

Nearly every reader object stores the data in custom objects designed for that specific purpose and use case. The `HomogUniv` object is responsible for storing all homogenized group constants for a single spatial region at a specific point in calendar time. This provides a convenient way to quickly examine homogenized data for a single universe and to pass a single collection of data to follow-on analysis, such as nodal diffusion simulations.

The `ResultsReader` creates these universe representations automatically, and provides a simple `getUniv` method to retrieve a homogenized universe. Without this approach, users would have to index into both the group constant index vectors and a temporal vector to identify the start of a universes' data block. This is not an impossible task, but is hindered by the repeated data present in the primary result file, as discussed in Sec. II.B.

A benefit of the object-oriented approach is that classes can be re-used across file types. For the branching coefficient file, wherein group constants are presented across various perturbation states, `serpentTools` presents similar `HomogUniv` instances to the user. As a further improvement, this specific file is not produced as a MATLAB script, meaning that users must write their own parser to handle what can be a very useful output.

Data from detectors with or without Cartesian, hexagonal, and curvilinear meshes are stored on dedicated and unique `Detector` objects. Depending on the underlying spatial grid, a `CartesianDetector` or `HexagonalDetector` may be provided, binned across more quantities of interest like energy, material, reaction, etc. This differentiation provides unique methods for representing the data, such as a hexagonal plot method that builds to-scale surface plots matching the defined hexagonal meshing. As will be demonstrated in Sec. III.B, the `serpentTools` API also provides visualization methods "out-of-the-box" that can produce publication-quality plots and perform detailed analysis into complicated tally data.

II.B. Data Filtering and Control

Without any additional action, `serpentTools` will store all of the data present in the output file. However, due to a growing set of capabilities, the Serpent output files contain a plethora of information, all of which may not be relevant for every analysis type. Due to the structure of the main result file, information like the Serpent `VERSION` and parallel performance statistics are printed for every burnup set but also for every homogenized universe the user has requested.

Consider the case of a two-dimensional lattice supercell of a fuel assembly adjacent to or surrounded by reflectors. Homogenized macroscopic cross sections are requested for both the fuel and reflector assemblies, and five burnup points are given to obtain the cross sections over time. Ten sets of macroscopic multigroup data, such as absorption cross sections, fission spectrum, and n'th moment of the scattering matrix, will be given, one for each homogenized universe at each point in burnup. However, information like the global multiplication factor `ABS_KEFF` will also be given ten times, as dictated by the structure of the result file. Five of these values will be unique, corresponding to each point in burnup, but the other five will be repeated values. Furthermore, information like the Serpent version and run title will also be given ten times, even though these value are invariant with respect to burnup and universe index. Loading this file into MATLAB or Octave would result in loading redundant data, like the repeated multiplication factor or Serpent version. Conversely, `serpentTools` performs passive and active data filtering to (1) remove redundant repeated data and (2) retain information relevant to the user at a given moment. These two features help improve performance and reduce overall memory consumption.

II.B.1. Passive Filtering

Some filtering is performed automatically, especially when reading the primary result file. Using the above example, all ten sets of homogenized group constants would be stored on ten unique `HomogUniv` instances. However, only the five unique values of `ABS_KEFF` and similar burnup-dependent data are stored on the `ResultsReader`. Values that are constant throughout all burnup and homogenized universe iterations, such as the version information and particle counts, are stored only once.

This filtering is performed by understanding the content and structure of each output file and using that information when loading data. For the case of the result file,

a preliminary step is taken to determine the number of homogenized universes. Due to the structure of the result file and how Serpent prints the data, the number of repeated values per burnup step is equal to the number of homogenized universes. When reading the result file, the first value of `ABS_KEFF` is loaded, while the second is skipped, since it is a repeated value. For an arbitrary number of homogenized universes N, a value like `ABS_KEFF` is loaded once every N encounters, as it is only considered unique once per burnup step. Macroscopic cross sections are loaded at each encounter, as these correspond to unique universes (fuel or reflector assemblies in the previous example) at each point in calendar time.

II.B.2. Active Filtering via User Control

The user can directly request variables from the output file, such as the absorption estimator of the multiplication factor or homogenized cross sections. If the user wanted to obtain all group constants and all multiplication factors, there exist groupings of similar quantities that can be used to select multiple values with only a few keywords. Both of these actions can be used in tandem, and are reversible, e.g., the user can easily revert to storing all data from the output file. An example of this control can be found in the example documentation.[b]

For the case of the depletion files, several unique quantities are produced for every burnable material. These quantities are typically $N_z \times N_{bu}$ matrices, where N_Z is the number of isotopes requested by the users and N_{bu} is the number of burnup points. For problems with many burnable regions and burnup steps, the amount of information present in this file can quickly become unmanageable. Serpent does provide ways to reduce the size of this file through automated material subdivision, wherein a single burnable material defined across multiple locations, e.g., a single 3.5 wt% UO_2 fuel across a fuel assembly, is replicated in memory across each location. Each material will then be depleted using corresponding local reaction rates, while only the original material will be contained in the depletion output file.

Similar filtering capabilities are provided for the depletion file as for the result file. Here, the user can request data for specific materials and/or all materials that match a specific pattern. The specific quantities, such as atomic densities or decay heats, can also be selected directly by the user. These settings and more[c] can be used to conserve computational resources and improve processing time with `serpentTools`.

III. USE CASES AND DEMONSTRATIONS

This section is dedicated to highlighting some analyses that Serpent users may perform and how `serpentTools` can expedite and supplement existing workflows. Emphasis is placed on the variety of plots that can be produced using the API, as it is impossible to cover every possible situation. Some light theming has been performed using `matplotlib` settings, but all the plots are generated using less than ten commands.

III.A. Simple Analysis

This section presents a very basic setting using `serpentTools` to view and plot data from the primary result file. Code Block 1 demonstrates some usable code that, when executed inside a Python environment, loads, prints, and plots multiplication factor and homogenized cross sections. The user is referred to the full documentation for detailed examples.[d]

```
>>> import serpentTools
>>> res = serpentTools.read("demo_res.m")
# retrieve multiplication factor
>>> print(res.resdata["absKeff"])
array([[1.21645e+00, 3.8000e-4],
       [1.18028e+00, 4.1000e-4],
. . .
       [1.06846e+00, 4.3000e-4]])
>>> res.plot("burnup", "absKeff",
  ylabel="Multiplication Factor")
>>> u2 = res.getUniv("2", burnup=10)
# retrieve nu-sigma fission from homogenized
  universe
>>> print(u2.infExp["infNsf"])
array([0.0072016, 0.00066214, 0.00864251,
  0.00706604 . . .
0.103409, 0.151597, 0.23421])
>>> u2.plot(["infFiss", "infAbs"], labels=
  ["Fission", "Absorption"])
```

Code Block 1: Example code for reading in and manipulating data from a result file.

Figures 1 and 2 contain the two figures created using the plot commands in Code Block 1. First, the global multiplication factor across burnup is given, with some automatic labeling of the axes. Control for the confidence interval is extended to the user,

[b] https://serpent-tools.readthedocs.io/en/latest/examples/ResultsReader.html.

[c] https://serpent-tools.readthedocs.io/en/master/settings.html.

[d] https://serpent-tools.readthedocs.io/en/latest/examples/index.html.

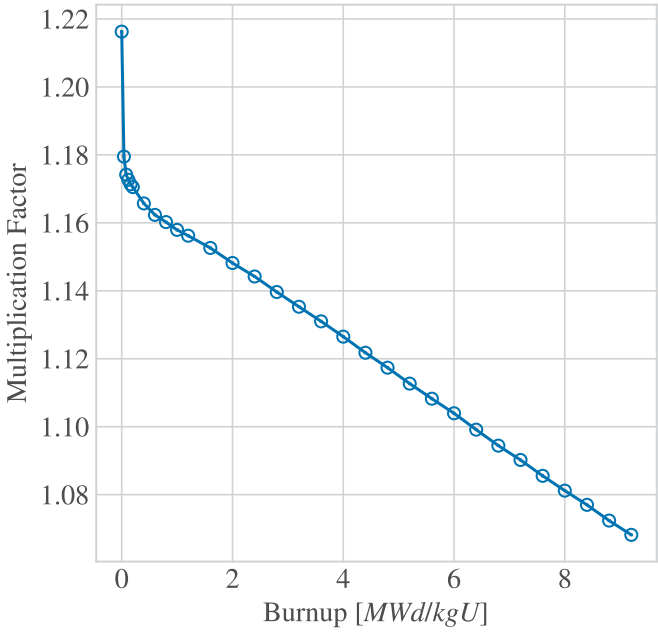

Fig. 1. Global multiplication factor as plotted using `serpentTools`.

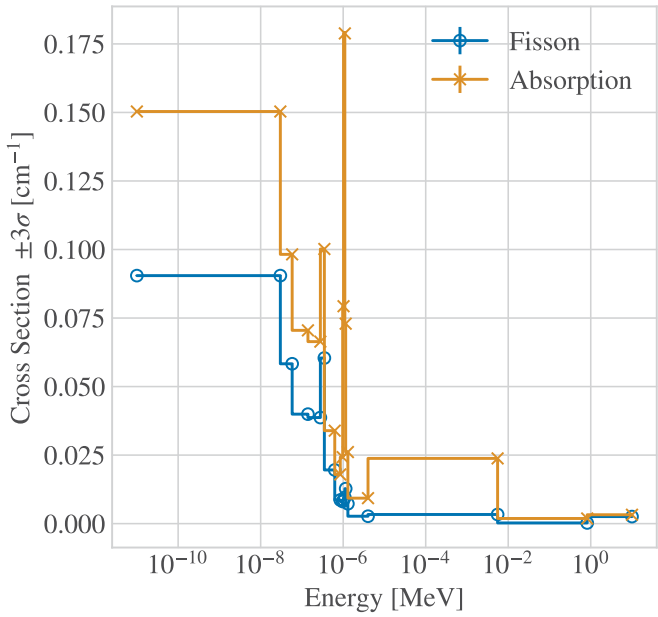

Fig. 2. Homogenized macroscopic cross sections visualized with `serpentTools`.

defaulting to three σ. The absolute magnitude of the uncertainty is plotted as vertical error bars centered at the expected value in all cases. Specifically, for Fig. 1 the uncertainties are on the order of 10 pcm and are difficult to discern. This is a plot that could easily be replicated in MATLAB or Octave given the sequential nature of the `ABS_KEFF` data, but serves as a gentle introduction into the API.

The universe data plot in Fig. 2 would require the user to index into the burnup and group constant universe identifier vectors to find the correct row in `INF_FISS` and `INF_ABS` prior to plotting. With `serpentTools`, this indexing is done for the user, returning the correct `HomogUniv` object that stores the macroscopic cross sections using a `getUniv` method.

III.B. Convenient Visualization Methods

As mentioned above, `serpentTools` supports nearly all plain-text outputs, including depletion, detector, sensitivity, history, and branching coefficient files. Many of the readers have flexible visualization methods, including energy spectra, meshed tally data, and isotopic evolution over time. This section attempts to illustrate some of the more useful plot methods provided to the user.

Serpent has broad support for tallying quantities of interest (e.g., flux, powers, reaction rates) with or without some filtering or binning (e.g., spatial mesh, energy grid, time bins). Users can also stack filters and scores to create multidimensional tally data. For example, one could tally the fast and thermal fission reaction rate across a Cartesian mesh and across multiple materials. With `serpentTools`, tally data are reshaped such that each dimension corresponds to a unique bin type (e.g., reaction, material, x mesh index, etc.). This way, one can easily look at just the fast fission rate from the example without having to define multiple detectors.

Figures 3 and 4 demonstrate two plots that can be built using `serpentTools` in less than ten lines of code. First, Fig. 3 provides the neutron spectrum, normalized per unit lethargy. All the normalization and labeling is done with a single function call, and the user has full control over the axis labels and scales and the confidence interval for uncertainties.

While plotting the flux spectrum can be a straightforward task, meshed data can pose additional challenges. Rectangular meshes can be plotted fairly easily, both with MATLAB and `matplotlib`. When relying on `serpentTools` for Cartesian mesh plots, the true spatial grid of the problem is considered, allowing each dimension to have a potentially different mesh spacing. Furthermore, arguments can be passed to these mesh plot routines to filter data below specific values and color meshes based on a logarithmic scaling. The latter is helpful for problems with greatly varying tally magnitudes. Color bars and axis labels are provided by default and easily manipulated through the API.

Additionally, `serpentTools` provides a hexagonal plot method, demonstrated in Fig. 4. Cells with no fissile

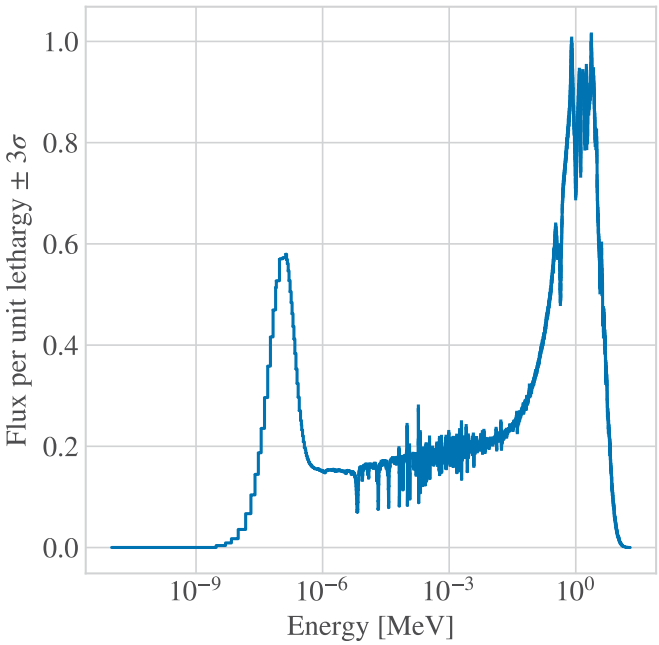

Fig. 3. Flux spectrum, automatically normalized per unit lethargy.

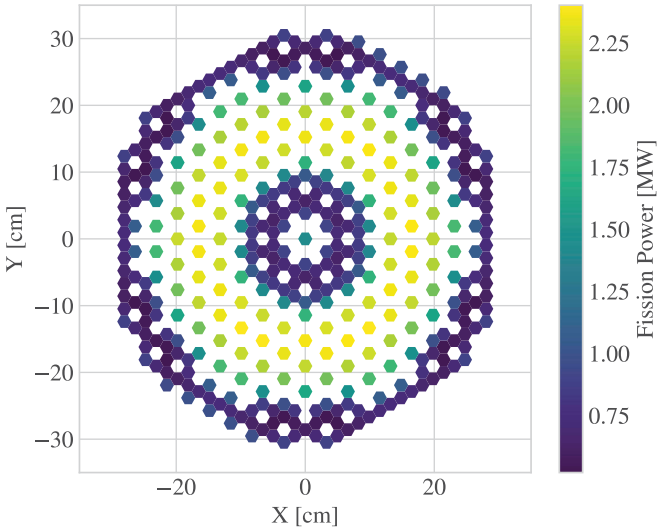

Fig. 4. Mesh plot of power profiles leveraging hexagonal detector support.

materials do not contribute to the local fission energy plotted and have been filtered out through the plot method. Similar formatting options are provided as with the Cartesian mesh plot routines described above.

Through the depletion file, users have access to isotopic atom and mass densities, decay heat contributions, toxicities, and more. The output files contain a large collection of arrays, which are converted to `DepletedMaterial` objects. Each of these objects has methods for quickly plotting isotopic data for one or more nuclides. Figure 5 is one such example, containing decay heat generated by multiple isotopes for a single burnable material.

III.C. Support for Total Monte Carlo Simulations

`serpentTools` also supports reading multiple depletion or detector files at once and automatically computing average quantities for the user. The process of repeating identical runs with unique random seeds to reduce stochastic effects is a common practice in Monte Carlo work and can be aided by using the `serpentTools` `Sampler` classes.[e] Figures 6 and 7 present the spread of axial power tallies and isotopic activity obtained using these classes using a single function call. The mean values are also computed and plotted for convenience.

III.D. Conversion to Alternative Storage

The authors of `serpentTools` are cognizant that many users and workflows may prefer MATLAB, due to personal preference and/or existing analysis frameworks. These Serpent users may still have to work with prohibitively large Serpent output files with large loading times and potentially large rates of failure. As a remedy, the `serpentTools` package contains a conversion tool that

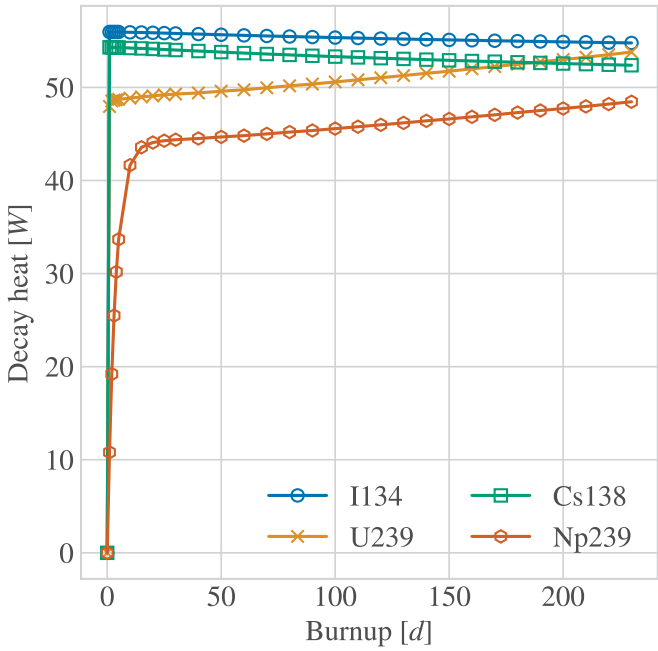

Fig. 5. Top contributors to final decay heat as plotted with `serpentTools`.

[e] https://serpent-tools.readthedocs.io/en/master/samplers/index.html.

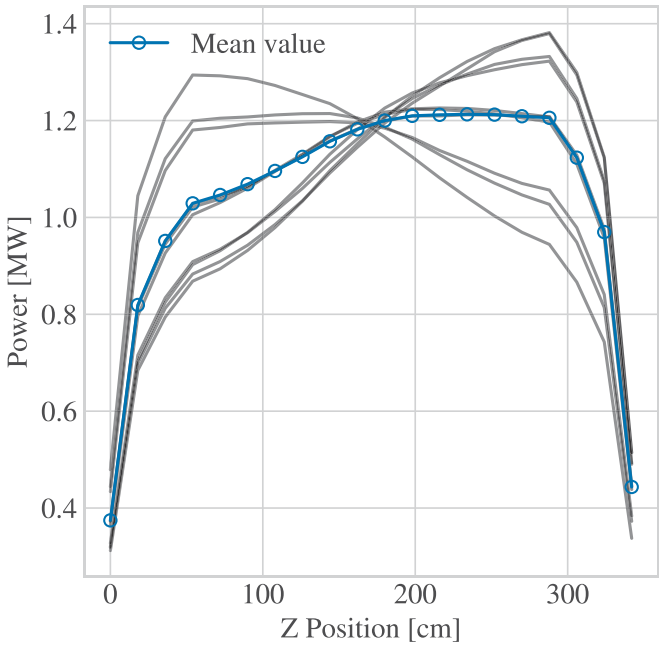

Fig. 6. Spread of axial powers produced using a `DetectorSampler`.

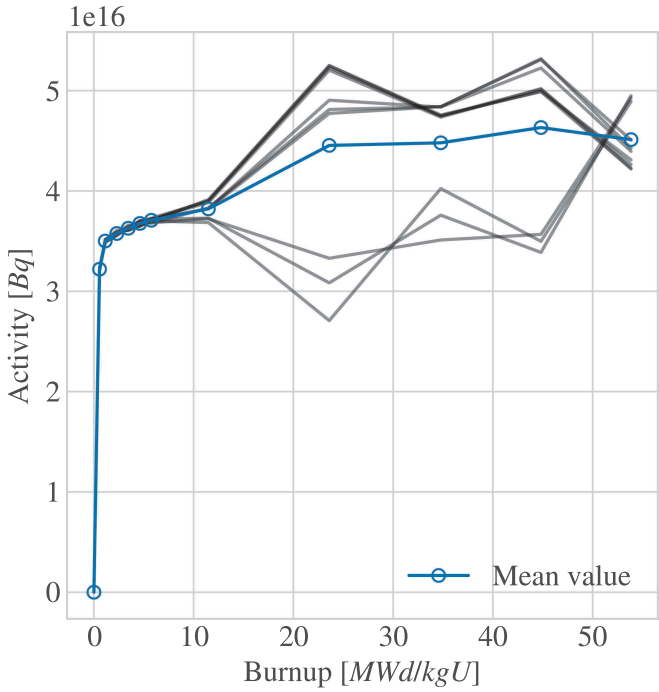

Fig. 7. Spread of ^{239}Np activity through depletion from ten unique runs.

will convert the plain-text .m output files to binary .mat files. It is the experience of the authors that these binary files are more likely to be loaded into MATLAB. From the command line or the Python API, a user can utilize `serpentTools` as a compression tool to create this binary file. Furthermore, using filtering capabilities, the binary file can be further reduced, storing only the data the user requested.

The developers are also interested in exploring additional file types, specifically hierarchical data format (HDF) types.[11] Consider the following workflow, where a Serpent job with multiple physics (spatial homogenization, burnup, tallies, etc.) has been performed on a remote cluster or workstation. Rather than retrieving raw text files onto a user's local machine, `serpentTools` could be used to filter, compress, and produce a single HDF file. By only storing information needed for a specific analysis, less information has to be downloaded from the remote location, freeing up both local disk space and network bandwidth.

IV. COMPARISONS TO OTHER PROGRAMS

Two comparisons are made in this section. First, a comparison of wall-clock times required to load data in MATLAB, Octave, and `serpentTools` is presented. Section IV.B details some philosophical and implementation-level differences between `serpentTools` and `PyNE` (Ref. 10), another popular Python package for nuclear engineering.

IV.A. Loading Times

Result files were generated using increasing numbers of burnup points and requested homogenized universes. MATLAB and Octave were tasked with opening without rendering their user interfaces, loading the data with the `run` commands, and exiting. For `serpentTools`, the time was measured as the time required to import `serpentTools` and load the data from within a Python environment using `serpentTools.read`. All comparisons were performed in the same computational environment using a single processing unit.

Loading times for each environment are presented in Fig. 8. The startup time for each environment is deducted from the total time. For simple analyses, including a few homogenized universes and/or a few burnup points, the loading times may be inconsequential. However, as demonstrated in Fig. 8, `serpentTools` scales far better than MATLAB or Octave for the primary result file. A main reason for this scalability is the ability to only store one set of nonhomogenized data per burnup step. Here, `ABS_KEFF` contains N_{bu} rows rather than $N_{bu} \times N_{univ}$ produced using MATLAB or Octave, as discussed in Sec. II.B. Similar improvements can be found for the depletion file as well. It is worth noting that Fig. 8 does

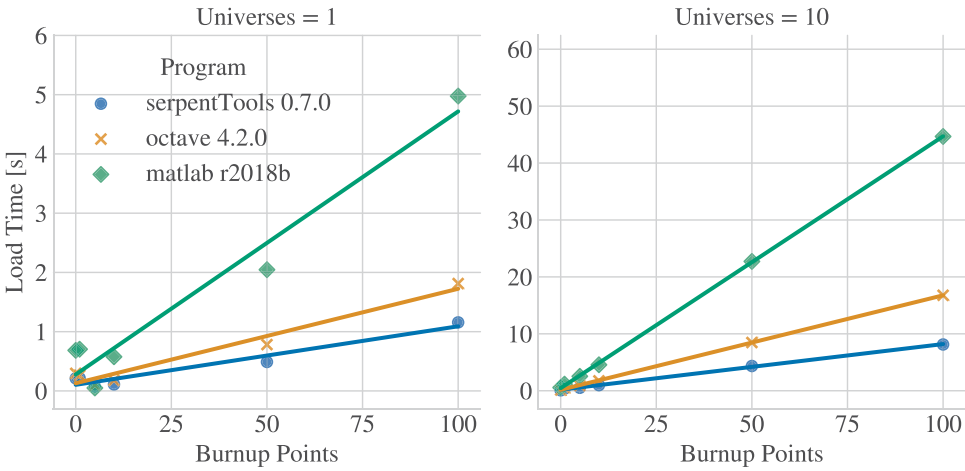

Fig. 8. Time required to load results data with different environments.

not reflect the improvements from actively filtering data per the `rc` object. This will surely improve loading times and memory footprints because array conversions and storage are replaced with a few conditional statements.

IV.B. PyNE Nuclear Engineering Toolkit

`PyNE` (Ref. 10) is an open source, Berkeley Software Distribution (BSD)-licensed[f] Python package that is capable of reading select Serpent outputs. These are limited to the primary result file, depletion, and detector files.[g] Data read in using the `pyne.serpent` module is converted to a dictionary of arrays, with the option to write the data to a separate Python file. Utilizing `PyNE` does afford the user identical data that one would obtain with `serpentTools`, and the authors make no attempt to besmirch the `PyNE` project. Indeed, `PyNE` was a first mover in the realm of open-source nuclear engineering tools and offers more diverse tools than present in `serpentTools`.

The primary differences in using `serpentTools` over `PyNE` is the scope of the projects and the data storage. `serpentTools` has chosen to focus solely on parsing and storing Serpent data, with minor enhancements. Conversely, `PyNE` contains a wide range of features that may or may not be applicable to the average Serpent user, such as enrichment solvers, JSON bindings, and spatial solvers.[h] Second, reading select Serpent files using `PyNE`, users are presented with dictionaries of `NumPy` arrays, largely equivalent to a MATLAB workspace.

As demonstrated in Sec. III.B, `serpentTools` has placed a high value on some convenience methods. `PyNE` leaves such features up to the users and may require a more in-depth knowledge of the Python ecosystem. This is not a substantial ask, and if both packages present the user with the same data, ultimately the same analyses and workflows can be supported.

Furthermore, `serpentTools` provides a rich API, full of application-specific data structures for storing and accessing data. The `ResultsReader`, responsible for handling the primary result file, also produces `HomogUniv` instances, responsible for storing all the group constants for a single universe at a single point in calendar time. This can be advantageous, as the user is shielded from multiple indexing routines and can instead focus on analyzing the data. Similarly, for the `DepletedMaterial` objects produced from the depletion file, all of the isotopic data for a single material are presented in a compact namespace.

V. CONCLUSION

`serpentTools`, an alternative method for processing data from the Serpent Monte Carlo code, has been presented. Compared to MATLAB and Octave, `serpentTools` stores all the same data without user interaction, using community-standard Python array data structures. `serpentTools` utilizes an object-oriented data model, removing the chance of name collision in loading successive files, and provides several simple and powerful visualization methods.

Each of the main plain-text outputs has a dedicated reader that is responsible for understanding the file structure and content. These readers can load either all data or

[f] https://github.com/PyNE/PyNE/blob/0.5.11/license.txt.
[g] https://github.com/PyNE/PyNE/blob/0.5.11/PyNE/serpent.py.
[h] http://pyne.io/usersguide/index.html.

a user-specified subset into a Python environment. By restricting the scope of each parser, data can be loaded two to five times faster than MATLAB or Octave. Contrasting with MATLAB, `serpentTools` is open source and distributed free of charge under an MIT license, removing the need for expensive single-use or institutional licenses. Furthermore, `serpentTools` provides single-line plotting routines and novel data structures that, to the best of the authors' knowledge, do not exist in the published literature.

For the interested, all the documentation for `serpentTools` can be found online at https://serpent-tools.readthedocs.io/en/master/, including installation guides and detailed examples. Those who wish to contribute to the project are welcome to do so and are recommended to follow a developer guide.[i] Last, `serpentTools` is in near constant development, driven by community needs and feedback. Bugs can be reported and features requested through the project's GitHub page. Users who wish to be informed of updated versions and released software are encouraged to contact the developers[j] to be added to a mailing list and to follow the project development.

Acknowledgments

This work was partially funded through the U.S. Regulatory Commission, project number HQ-84-14-G-0058. The authors would also like to thank contributors Paul Romano and Anton Travleev. Serpent is developed and maintained by the VTT Technical Research Centre of Finland, LTD, which does not endorse nor financially support `serpentTools` nor the content of this paper.

ORCID

Andrew E. Johnson http://orcid.org/0000-0003-2125-8775
Dan Kotlyar http://orcid.org/0000-0002-5581-7400
Stefano Terlizzi http://orcid.org/0000-0002-2179-6963
Gavin Ridley http://orcid.org/0000-0003-1635-8042

References

1. J. LEPPÄNEN et al., "The Serpent Monte Carlo Code: Status, Development, and Applications in 2013," *Ann. Nucl. Energy*, **82**, 142 (2015); https://doi.org/10.1016/j.anucene.2014.08.024.
2. J. LEPPÄNEN, M. PUSA, and E. FRIDMAN, "Overview of Methodology for Spatial Homogenization in the Serpent 2 Monte Carlo Code," *Ann. Nucl. Energy*, **96**, 126 (2016); https://doi.org/10.1016/j.anucene.2016.06.007.
3. V. VALTAVIRTA, "Development and Applications of Multiphysics Capabilities in a Continuous Energy Monte Carlo Neutron Transport Code," PhD Thesis, School of Science, Department of Applied Physics, Allto University, Finland (2017).
4. M. AUFIERO et al., "A Collision History-Based Approach to Sensitivity/Perturbation Calculations in the Continuous Energy Monte Carlo Code Serpent," *Ann. Nucl. Energy*, **85**, 245 (2015); https://doi.org/10.1016/j.anucene.2015.05.008.
5. "MATLAB Release r2019b," The MathWorks Inc., Natick, Massachusetts (2019); https://www.mathworks.com/products/matlab.html.
6. J. W. EATON et al., "GNU Octave Version 5.1.0 Manual: A High-Level Interactive Language for Numerical Computations," (2019); https://www.gnu.org/software/octave/doc/v5.1.0/ (current as of Nov. 15, 2019).
7. S. Van Der WALT, S. C. COLBERT, and G. VAROQUAUX, "The NumPy Array: A Structure for Efficient Numerical Computation," *Comput. Sci. Eng.*, **9**, 21 (2007); https://doi.org/10.1109/MCSE.2011.37.
8. J. D. HUNTER, "Matplotlib: A 2D Graphics Environment," *Comput. Sci. Eng.*, **9**, *3*, 90 (2007); https://doi.org/10.1109/MCSE.2007.55.
9. E. JONES et al., "SciPy: Open Source Scientific Tools for Python," (2001); http://www.scipy.org/ (current as of Nov. 15, 2019).
10. C. BATES et al., "PyNE Progress Report," *Proc. American Nuclear Society Winter Mtg.*, Vol. 111, pp. 1165–1168, Anaheim, California, November 9-13, 2014; http://www.ans.org/meetings/file/514.
11. "Hierarchical Data Format, Version 5," The HDF Group (1997-2019); https://www.hdfgroup.org/solutions/hdf5/ (current as of Nov. 15, 2019).

[i] https://serpent-tools.readthedocs.io/en/master/develop/index.html.
[j] http://pwp.gatech.edu/core/contact-us/.