Automating Nuclear Reactor Modeling for Simulation Using OpenMC and Python

Collin Bradford

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Matthew Memmott, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Automating Nuclear Reactor Modeling for Simulation Using OpenMC and Python

Collin Bradford
Department of Physics and Astronomy, BYU
Bachelor of Science

The Brigham Young University (BYU) Molten Salt Micro Reactor (MSMR), is a concept design for a small, modular, molten salt fueled nuclear reactor that has no active components making it class A passively safe. [1] Previously, the reactor models that have been created for neutronics simulations have been coded and changed by hand. A new coding library is presented that automates most of the modeling process, allowing engineers to change the design rapidly. The modeling code was verified through comparison with previous modeling work. The new software will allow rapid design iterations and the generation of a training data set for future machine learning work.

Keywords: nuclear reactor, small modular reactor, SMR, MSMR, molten salt, micro reactor, OpenMC, neutronics simulation code, Monte Carlo Simulation

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Matthew Memmott from the Brigham Young University chemical engineering department for his mentorship and direction both in this capstone project and in completing my undergraduate degree.

I would like to acknowledge Alphatech Research Corporation for funding this research.

This code could not have been written without valuable contributions of the members of my research group: Braden Clayton, Edward Mercado, LaGrande Gunnell, Jeff Adams, and Trevor Talbot who assisted in developing and testing this code.

Finally, I would like to acknowledge the contribution of my wife, Amelia, and other family members who have supported me during my research and undergraduate career.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nuclear energy is a key resource in a transition to clean energy. Nuclear power accounts for about 10% of global energy production and around 40% of clean energy. However, we need an estimated 80% increase in nuclear energy by 2040 to reach international climate goals. [2] A key advantage of nuclear energy is the ability to create carbon-free base load energy that is always available. This would remove the need for costly batteries and fossil-fuel based backup systems used in conjunction with intermittent renewable sources such as wind and solar energy. Nuclear energy also has a very small land use footprint in comparison with other renewable energy sources. The International Environmental Agency (IEA) estimates that cost savings associated with using nuclear energy in a clean energy transition to be around 1.6 trillion dollars. [2] The IEA recommends maintaining current reactors, supporting new reactor construction, and accelerating innovation in the nuclear energy field. [2] To answer the demand for safe, cost effective nuclear energy, a variety of advanced reactor designs are underdevelopment that present unique advantages over traditional pressurized water reactors (PWRs).

Small modular reactors (SMRs) sacrafice the large power output of most active reactors for low cost and versatility. They are designed to be manufactured in an assembly line, shipped to a service location, and brought online in a consistent, repeatable process. The repeatability of the process would bring down the cost of individual reactors, and the versatility of a small reactor would broaden the use cases to include industrial heat, electricity generation, and military applications. [3,4]

Molten salt reactors (MSRs) are designed for cost, safety, and reduction of nuclear waste. MSRs substitute traditional pressurized water coolant for liquid fluoride salts. This removes the dangers and cost associated with maintaining a pressurized reactor coolant system. Because molten salt has a high coefficient of thermal expansion in comparison with the ceramic fuel pellets in use today, some MSR designs dissolve the fissile material directly into the salt. [5–9] The high thermal expansion leads to a large negative coefficient of reactivity, adding a significant passive safety feature to the reactor. With some developments in chemical processes, the fuel salt could be cleaned and repurposed, reducing the waste output from the reactor. [6]

The Brigham Young University (BYU) molten salt micro reactor (MSMR) combines the cost and versatility advantages of an SMR with the safety and waste reduction advantages of an MSR. Although several similar designs have already been proposed [5–7], the BYU MSMR is the only MSMR design that has passive cooling and no active fuel salt circulation.

The BYU MSMR design consists of a cylindrical reactor core with radial interior heat transfer fins and exterior lateral heat transfer fins. The interior radial fins are filled with diamond which acts as both a path for heat transfer and a neutron moderator. The external fins are made of copper. The interior and exterior fin configuration allows the reactor to passively remove all decay heat through convection to the surrounding air in an emergency. Between the interior fins, a fluoride fuel salt rests without any active circulation. Heat exchange tubes run laterally through the reactor core and are used for heat extraction to later processes and are not needed for emergency cooling. Control rods run through the center of each heat exchange tube. The reactor is meant to be produced in a factory and fulfill a 7 year service life before the salt must be reprocessed. A rendered image of the reactor is shown in figure 1.1. Details about the heat transfer characteristics can be found in Larsen et. al. [1]

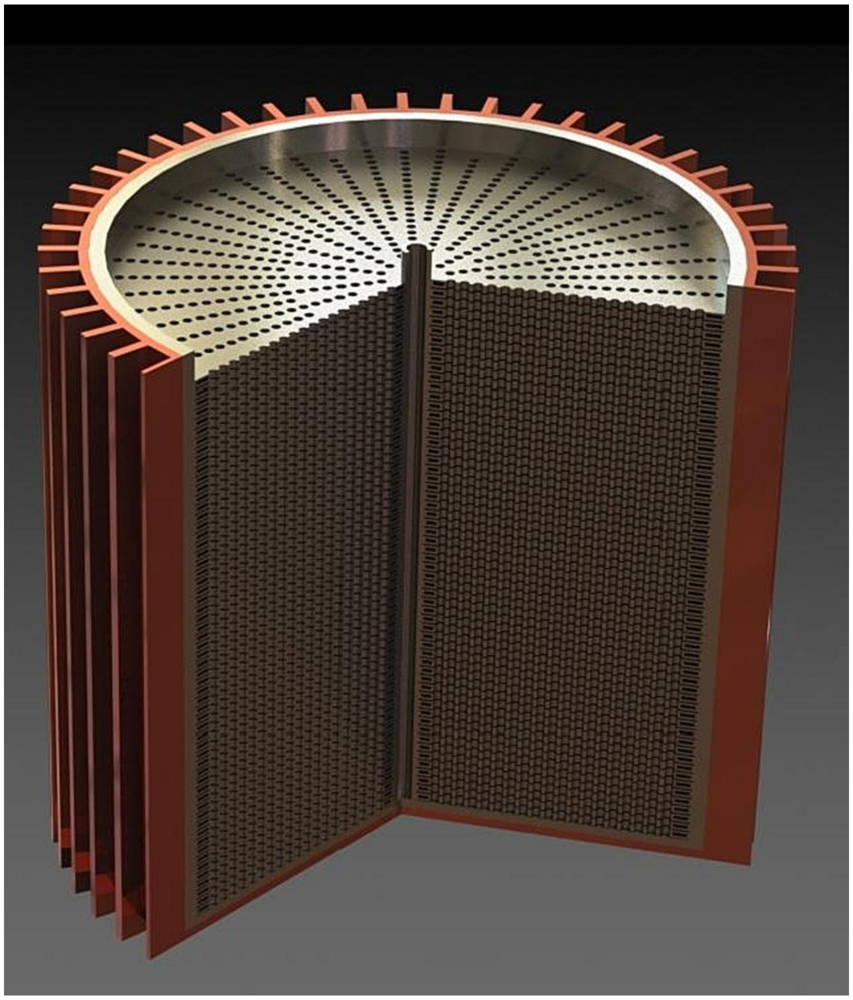

Figure 1.1 Rendered 3d image of the BYU MSMR. The heat sync fins are shown laterally on the outside of the reactor and horizontally inside the reactor. Heat exchange tubes run through the holes in the heat sync fins. Fuel salt rests between the fins and a diamond moderator (not shown) is inside of each fin. This image was originally published in Larsen et. al. [1]

In the absence of any active components, the reactor operates similar to a large thermal battery. It could be manufactured in a factory, loaded onto a truck, and placed in a pre-prepared site for the duration of its service life at which time a replacement would be supplied and the old reactor would be transported to a separate reprocessing facility. Very little maintenance would be required during its service life removing supply chain restrictions and manpower requirements. This makes the reactor ideal for use in a variety of industrial, military, and energy production applications.

To predict the neutronics properties of the BYU MSMR, we model the reactor using constructive solid geometry (CSG) and simulate the neutronics behavior using OpenMC, a Monte-Carlo simulation software. During the design and optimization of the reactor, the process of modeling and simulation must be repeated many times. Every time a change must be made, the CSG modeling code must be changed and debugged and a new simulation must be run. Describing a nuclear reactor in CSG code can get disorganized very easily, slowing down the modeling and debugging process. Depending on the quality of simulation that is being run, the simulation can take several hours for the computer to complete making it imperative that the modeling is done properly each time. Streamlining the process of modeling and simulation is a key optimization that can speed the development and optimization of any nuclear reactor.

In our research group, the modeling has needed an update for a long time. To model the reactor, we currently use a single file that describes all of the materials and geometry of the reactor. The file is about 450 lines long and includes very few comments. The values used to model the reactor such as material properties, material selections, and geometry measurements are often hard-coded at various places throughout the file. To make a change to the reactor design, an engineer copies the design file, changes values and code spread throughout the file, and simulates the model. Due to the disorganized code, it takes a lot of time to make changes to the code and it is difficult to verify that the changes have been made properly. This costs valuable engineering resources, complicates training for new engineers, and puts the validity of our data into question.

In addition to difficulties with the modeling process, the organization and standardization of our modeling code is also an issue. When an engineer makes changes to our current modeling files, a new code project is created and the old one is kept. This results in a large amount of data that is difficult to keep organized. This makes it hard to track the latest version of the reactor model within our group and at any given moment, different versions may exist for different engineers in the group. This adds to the resource cost of our group and threatens the validity of the our results.

To fix these inefficiencies, we organized a streamlined modeling system that standardizes the modeling code, modeling parameters, and the simulation results. The modeling code is now contained within a single coding library that draws values from a standardized set of comma separated value (csv) files. The coding library and modeling parameter files are contained in a centralized repository that all engineers in the group can access. When changes need to be made, an engineer creates a new coding file that references the centralized design library and parameters files. The library reads the standard files, changes values as defined by the engineer, and simulates the design. From the engineer's perspective, the coding file is only about 10-15 lines long and all parameters are easily accessible. This streamlines the modeling process, saving engineering resources.

In addition to saving time, the new modeling code allows the engineer to automate the process of modeling many small variations on the same design. For example, an engineer could sweep a design parameter by modeling the reactor using the library inside a loop and changing the parameter every time the loop iterates. This simplifies the process of optimizing design parameters and could later be used to generate simulation data to train a machine learning algorithm to recognize ideal design parameters.

Chapter 2

Methods

The new modeling code developed for the BYU MSMR is designed to allow engineers to make quick changes to the code, keep files organized, and optimize training and collaboration between other group members. The new code will cut out the majority of the time that engineers spend to changing the models, allowing them to spend more time refining the design, planning test runs, and eventually, training a machine learning model to predict key design parameters for the reactor.

The new modeling code follows the following goals:

- Automate repeatable processes
- Standardize the reactor design throughout the group
- Allow for easy customization
- Make code easy to maintain

The code structure, organization methodology, and documentation have all been created with these goals in mind.

2.1 Structure

The BYU MSMR modeling code consists of a Python library and a set of two csv files. The csv files contain all the modeling parameters and material definitions that are used in the design. The Python library contains all the code that is needed to take the design parameters and materials stored in the csv files and combine them into the reactor design. It also has functions for defining simulation parameters and starting a simulation run. Separating the design parameters and code into csv files and a library allows for easy customization as values can be changed without changing (and possibly breaking) the main design code in the Python library.

When the python coding library is initiated, it loads the parameters and materials from the two csv files. The first csv file consists of a list of geometry parameters. Each parameter reflects a value within the reactor such as a thickness of a component or the radius of a tube. The second defines a list of materials for components. Each material consists of a list of elements and isotopes with their accompanying weight or atomic percents. Once loaded, the values contained in the csv files contain all the information needed for the modeling code to form the base reactor design.

Once all the values are loaded into the coding library, the engineer can change any value necessary using the code that calls the library. This makes the design highly customizable and presents two simple ways that an engineer can change the design. First, the engineer can make changes to the base csv code. This is only recommended if the design changes are meant to be permanent and shared throughout the group. Second, the engineer can change the parameters while modeling the reactor just after the csv files have been loaded. These design changes are used in the resulting simulation files, but do not persist after the simulations are complete. This maintains the integrity of the csv files and the base design that is shared among all members of the group, allowing for flexibility when exploring new reactor design changes, but consistency in the base design.

In addition to managing the modeling code parameters, the coding library has a comprehensive set of functions that allow then engineer to model and simulate the design including functions for particle settings, tallies, modeling, and starting a simulation. The particle settings function allows the engineer to specify an initial distribution of particles used in every Monte Carlo simulation. Using the tallies function, an engineer specifies the tallies that will be tabulated by OpenMC during simulation. The modeling function takes the information read from the csv files, the changes made by the engineer, the settings, and tallies, and exports a full set of xml files that can be read by OpenMC during the simulation. If requested by the user, the modeling function can also export a script file along with the xml files that can be used to start a run on the BYU supercomputer. The modeling function also has an option to automatically start the simulation on the BYU supercomputer. Most of these features take only a single line of code to use from the engineer's perspective. This automates most of the common modeling procedures, allowing the engineer to focus on the changes that are being made to the design.

Using the csv files and modeling code, an engineer can create a base reactor design and start a simulation run using a simple 5 line script. This allows the engineer to focus on design changes rather than the modeling code. It also prevents errors and excessive debugging by presenting a standard process for modifying the design.

2.2 Validation

The BYU MSMR modeling code must undergo a comprehensive set of tests to verify that the design that is produced by the software reflects the intents of the engineers using the software. The validation tests cover everything up until simulation which consists of reading csv files, handling runtime modifications, setting the initial particle distribution, configuring tallies, and exporting all the information properly. This testing is necessary if an engineer is to trust the output from the simulations produced by OpenMC.

We are currently in the final stages of integration testing. During development, each unit was been tested by changing values and observing relevant output. As each feature was added to the main coding library, the coding library as a whole was tested as much as was reasonable considering the progress in development. Now that all main features have been drafted into the design, a comprehensive set of tests are being performed on the entire model to ensure that it represents the model we intend to use and that changes take effect properly.

The testing consists of visual geometry analysis and simulation runs. To test the geometry, we model variations on the base reactor design, chaining several parameters. After each model, we observe a rendered image using the OpenMC Plotter package [10]. This allows us to see that changes are being applied properly and we can take limited measurements using the scale on the output image to check that the features we are changing represent the proper dimensions. Although the visual checks are fast and easy compared with a full simulation, they are imprecise and more rigor is needed to verify the design completely.

To test the overall model rigorously, we simulate the design produced through the modeling code and find key values that can be compared with our current modeling code. A good example of this is the neutron multiplication factor, k_{eff}. k_{eff} refers to the number of new neutrons produced for every neutron in the reactor. The k_{eff} value is tied very closely to the geometry and materials of the reactor. By comparing the k_{eff} values from the new code with the old code, we can verify that the new code is doing what we expect.

Initial results of these tests are given in chapter 3. As explained later, we performed two separate tests. In the first test, we modeled the base reactor using various simplifying assumptions that reduce the computational resources required to simulate the model. These are reducing height (number of plates) of the reactor and adding periodic boundaries to the top and bottom, and reducing the model to a quarter section and adding periodic boundaries to the sides. When both of these assumptions are used, the reactor approximates a full-circle reactor that is infinite in the vertical direction. For the second set of tests, we performed the Doppler coefficient of reactivity analysis on the reactor in the same way that it was performed in Larsen et. al. All results are given below and they suggest that the new modeling software works as expected.

Chapter 3

Results

To verify that the model produced by the new modeling code matches the current model we are using, we ran two sets of simulations. The first set consists of several simulations that match the reactor design found in Larsen et. al. [1] but are simplified to shorten the modeling time required. The second set recreates the temperature offset sweep found in Larsen et. al. [1] that was used to calculate the Doppler coefficient of reactivity.

In addition to verifying the proper functionality of the new modeling code, estimates of the time savings gained by using the new modeling software are given in section 3.3.

3.1 Base Model Simulations

The base model simulations consist of several designs that match the base design found in Larsen et. al. [1] Each design has been simplified to reduce the amount of computational resources required to run the design. To increase computational efficiency, we assume that the design is an infinite cylinder and only model two or three plates within the design and add periodic boundary conditions to the top and bottom of the design. We also select a quarter section of the reactor sliced along the length of the reactor and place periodic boundaries on both sides of the section. This approximates

Plates	Section	Particles	k_{eff}	Error
3	Full	26,667	1.073278	$\pm$0.000575
2	Full	40,000	1.072458	$\pm$0.000474
2	Quarter	10,000	1.073406	$\pm$0.000955
?	Andrew's Run	?	1.070910	$\pm$0.000870

Table 3.1 Results from single runs using new modeling software and comparison with Larsen et. al. [1]

a full-cylinder reactor without simulating the entire design.

For the base model simulations, we simulated several versions of the same design with variations on the design simplifications. The results are given and compared with Larsen et. al. in table 3.1. For all designs performed with the new modeling software, we used a constant particle to volume ratio of 10,000 particles per quarter two plate section except for the largest run. This presents a model using manageable computational time and is consistent with most test runs performed in the group.

The standard deviation for the runs performed using the new software is 0.00042. The average k_{eff} is 1.073047 which represents a 0.2% difference from the result reported in Andrew et. al. of 1.07091.

3.2 Doppler coefficient Simulation

To test the ability of the new software to modify the temperature and densities of materials within the reactor, we recreated the Doppler coefficient of reactivity calculations found in Larsen et. al. [1]. For these runs, we modeled a quarter section with two plates and used periodic boundary conditions as defined in section 3.1. To find the dopler coefficient, we recreated 10 runs from Larsen et. al.

Temperature Offset (K)	k_{eff} from [1]	k_{eff} from new code	Difference	% Difference
-200	1.099070	1.095714	0.003357	0.305896
-156	1.093082	1.089083	0.004000	0.366574
-111	1.086602	1.084624	0.001978	0.182171
-67	1.079768	1.079924	0.000156	0.014443
-22	1.072443	1.074222	0.001779	0.165776
22	1.067040	1.071161	0.004121	0.385510
67	1.060186	1.064886	0.004700	0.442379
111	1.054869	1.059929	0.005060	0.478557
156	1.048637	1.054776	0.006139	0.583730
200	1.040793	1.049794	0.009002	0.861148

Table 3.2 Simulation results for Doppler coefficient of reactivity runs. Results from new modeling software are compared against results from Larsen et. al. [1]

with the same temperature offsets. The results are given in table 3.2. A graph of the k_{eff} values from the new software along with the values from Larsen et. al. [1] is given in figure 3.1.

To find the Doppler coefficient of reactivity, we used the same calculations given in Larsen et. al. [1]. We found the line of best fit through the k_{eff} values and multiplied it by $1/k_{eff}^2$ to get the coefficient of reactivity as defined in formula 3.1. The k_{eff} value that we used in equation 3.1 was the average k_{eff} value found in section 3.1. The final value obtained was -9.7713 pcm which varies from the value given in Larsen et. al. [1] of -14.4331 pcm by 38%.

$$\alpha_T = \frac{1}{k_{eff}^2}\frac{\delta k_{eff}}{\delta T} \tag{3.1}$$

Based on the fact that each individual run from the new software matches the runs given in Larsen et. al. with an error of less than 1%, we believe that the large discrepancy between the

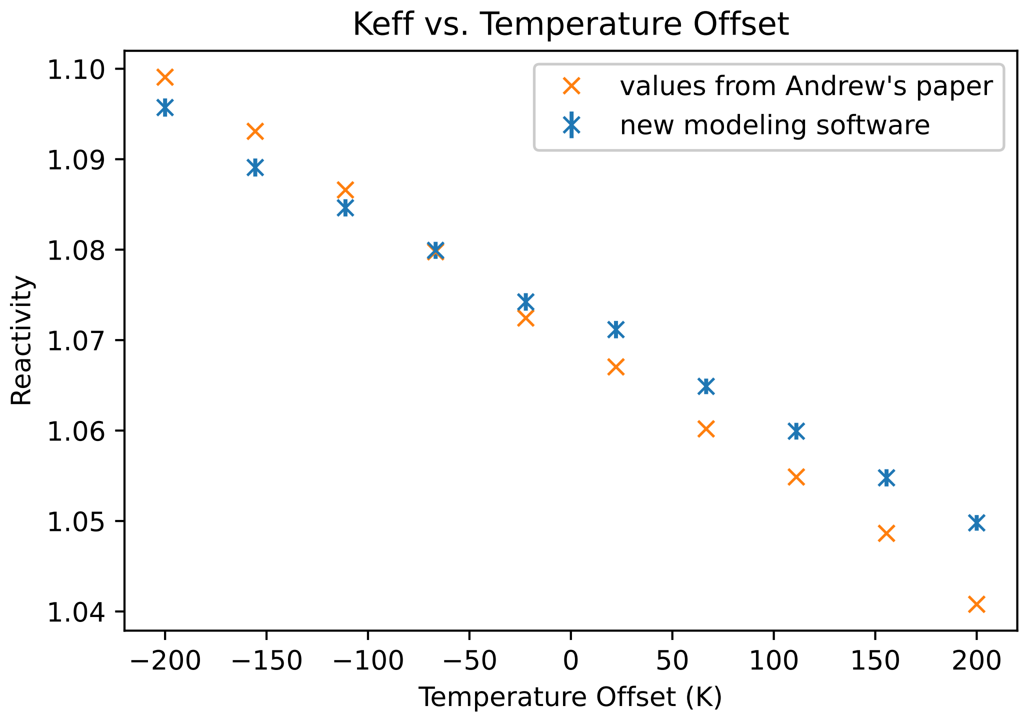

Figure 3.1 A comparison of the reactivity coefficients from the new software against the same values from Larsen et. al. [1]. These values are used to find the Doppler coefficient of reactivity.

Doppler coefficient reported in Larsen et. al. and the value reported from the new software is due to using different formulas for one or more of the material densities based on temperature. This would explain the trends in figure 3.1 that show both sets of runs being roughly linear but with different slopes. This would indicate that the new software itself performs well, but the formulas defined in the materials.csv file that is used to model the reactor are wrong or, at least, different from those used in Larsen et. al.

3.3 Time Requirements

To evaluate the effect of the new modeling software on engineering time required to model the reactor, we estimated time requirements based on the old method of direct code manipulation and the new method using the modeling library and compared the results.

Using the old method of direct code manipulation, it is estimated that a change to the reactor design would take 30 to 120 minutes depending on the amount of debugging that had to be performed and the skill level of the programmer. After the initial design change, any similar change would take 5 minutes to copy the required files, edit the required variables, and start the run. Thus, the contribution of the engineer is roughly 75 minutes plus 5 extra minutes per variation.

Using the new software, it is estimated that an engineer can start a run in 10 to 30 minutes based on the programmer's familiarity with the software. This initial time investment includes programming a loop that models all variations of the reactor. It takes the modeling software around 2 seconds to model each version of the reactor. Thus, the programming contribution of the engineer is roughly 30 minutes to set up the run.

In addition to time savings for each run, the training requirements for new engineers is reduced when using the new modeling software. It is difficult to estimate the time requirement for students using the traditional modeling method as most students had to learn OpenMC CSG modeling from scratch, a process that usually took several months. The new modeling code hides most of the CSG code and allows new engineers to start modeling the reactor using a small set of functions and previous Python knowledge. Most new researchers should be able to start contributing within a couple of weeks. With the new modeling code, the limiting factor on the contributions of a new researcher to the group is now the engineer's knowledge of neutronics and not their programming skills.

Chapter 4

Conclusions

From the first set of simulation runs given in section 3.1, we see that using simplifying assumptions to reduce the computational time of the reactor does not affect the results of the simulation. This is based on the standard deviation for the k_{eff} values found in the set of results of 0.00042. The run also shows that the base design matches the design in Larsen et. al. [1]. This is based on the error between the average of the new runs and the value reported the paper of less than 0.2%.

From the second set of simulation runs given in section 3.2, we see that a variety of simulations match those given in Larsen et. al. [1]. This is based on the fact that none of the k_{eff} values for the simulations were different by more than 1%. We also see that the software properly models variations on the base reactor design. This can be seen through observing that the general trends of the run match those given in Larsen et. al. For example, the k_{eff} values from the simulation runs still follow a linear trend with a negative slope. We also see that something is wrong with the specifics of the simulations that are being performed. This can be seen by observing that the coefficient of reactivity calculated is different from the value reported by Larsen et. al. by 38%. This is likely due to discrepancies in the formulas used to find physical constants as seen by the way that the error changes across the simulations in figure 3.1.

Based on the time analysis given in section 3.3, we see that the average initial time requirement for a set of simulation runs is reduced by just over half of the time it would take without the new software. In addition, there is no continued time requirement for variations on the design when the new software is used. In addition to the reduced time requirement for routine simulation runs, the training time has been reduced such that learning the software is no longer the constraining factor when training new researchers.

Appendix A

Code Documentation

A.1 class modeler

Models the reactor from parameterized values stored in a set of csv files. The values are loaded from the current directory using the load() method. When loaded, the values are stored in the materials and geometry dictionaries. The user can then change the values in the dictionaries. Following the changes, the user can model the reactor using the `model()` command. The location parameter allows the user to specify a folder where the files should be kept. This allows the user to create many runs all stored in separate folders.

A.1.1 Attributes

geometry : dictionary

The dictionary that holds all the geometry values. After the load() function has been called, this is dictionary mirrors the geometry csv file. These values can be changed before the reactor is modeled.

materials : dictionary

The dictionary that holds all the materials values. After the load() function has been called, this is dictionary mirrors the materials csv file. These values can be changed before the reactor is modeled.

useQuarterSection : boolean

Defines whether the modeler should slice the reactor into a quarter section for modeling. This cuts down on computation time. If a quarter section is used, the settings and tallies are updated to reflect the change (namely, the particle starting box and the tally mesh are confined inside the quarter section). The default is false.

materialObjects : dictionary

The dictionary that holds all the OpenMC materials objects. This field is updated using the updateMaterialObjects() command which updates this dictionary using the values in the materials dictionary.

colorScheme : dictionary

This is a default color scheme that matches some colors to materials for ease of plotting. This dictionary can be used directly with the universe.plot() function provided by OpenMC.

includeSH : boolean

Whether or not to include a .sh script that can be submitted to the BYU supercomputer.

shCPUCount : int

Number of CPUs used for the run on the supercomputer. This value is stored in the .sh file if that is used.

shMem : string

String representation of the memory used for the run on the supercomputer. This value is used in creating the .sh file.

shTime : string

The time that the run is allowed on the supercomputer in the format: "hh:mm:ss".

shEmail : string

Email for notifications sent from the supercomputer. This value is used in creating the .sh file.

shNotifyBegin : boolean

Notify when the run begins. This value is used in creating the .sh file.

shNotifyFail : boolean

Notify if the run fails. This value is used in creating the .sh file.

shNotifyFinish = boolean

Notify when the run finishes. This value is used in creating the .sh file.

modelDir : string

The directory in which to place the finished model. This is usually set when model() is called.

modelingFilesDir : string

The directory that holds the modeling files. This is usually set when load() is called.

autoSubmitBatch : boolean

Automiatically submit a batch to the supercomputer when the model is complete.

useDefaultHXTubeConfig : boolean

Uses the default hx tube configuration found in Andrew's paper. The default value is False.

A.1.2 Methods

load(self,matfile="materials.csv",geomfile="geometry.csv") : None

Loads the the geometry and materials values from the approperate csv files.

model(self,path='.',excRegion=None,cells=None) : None

Models the reactor and places the xml files in the directory specified by the path parameter. Before modeling, the materialObjects dictionary is updated from the parameters stored in the materials dictionary. Also, if excRegion or cells are specified, the geometry is updated. HOWEVER if the excRegion and cells are NOT specified, the method will default to the geometry currently held in the object memory. That is, if the user has already called updateGeometry() and specified regions to exclude and cells to include, those values will be used to model the reactor.

updateMaterialObjects(self) : None

Updates the materials objects dictionary. This function builds a set of OpenMC materials objects from the materials data dictionary values. This should be called any time changes are made to the material data and before modeling. This function also updates the colorScheme that is used in default plotting.

xplot(self,x=0,resFactor=8,figsize=None) : matplotlib.image.AxesImage

Plots the reactor using a slice parallel to the zy plane and through the x coordinate given by the x parameter. The resFactor variable sets the resolution of the image. The figsize sets the display size for the image. If None, then only one image is plotted using the default OpenMC size.

zplot(self,z=0,resFactor=8,figsize=None) : matplotlib.image.AxesImage

Plots the reactor using a slice parallel to the xy plane and through the z coordinate given by the z parameter. The resFactor variable sets the resolution of the image. The figsize sets the display size for the image. If None, then only one image is plotted using the default OpenMC size.

tallies(self,meshDimensions,path,lowerLeft=None,upperRight=None) : None

Defines the tallies for the reactor using the values specified by the user.

settings(self,batches,inactive,particles,distribution=None) : None

Defines the settings for the run using the values specified by the user.

setHXTubeConfig(self,hxTubeLocations) : None

Sets the location of hx tubes within the reactor. hxTubeLocations expects a list of x,y coordinates in format: [[x1,y1],[x2,y2],...[xn,yn]]

setDefaultHXTubeConfig(self) : None

Sets the "default" set of hx tubes. The default configuration is the one that is used in Andrew's paper. This is not set automatically, but must be set by the user.

updateTemp(material,temp) : None

Sets a new temperature for a material. The density is set automatically using the density function defined in the materials.csv file.

tempOffset(offset) : None

Applies a temperature offset to all materials in the reactor. Densities are automatically adjusted using the density function defined in the materials.csv file.

A.2 load(self, modelingFilesDir)

Loads the the geometry and materials values from the approperate csv files.

A.2.1 Parameters

modelingFilesDir : string

The path to the materials and geometry files.

A.3 model(self,path='.',excRegion=None,cells=None,submit=False)

Models the reactor

A.3.1 Parameters

path : string

The path at which all the completed modeling files should be stored. This is the directory from which OpenMC can be run to simulate the design.

excRegion : OpenMC.Region

Any region that should be subtracted from the cells of the base reactor design. This allows extra cells to be inserted in place of these excluded regions. The regions will be sliced out of the reactor just before it is modeled using the appropriate equivalent of the command: `base_reactor_region = base_reactor_region & ∼excRegion`

cells : [OpenMC.cell,...]

A list of cells that should be added to the reactor model. Any area that is taken up by a new cell should be included appropriately using the excRegion parameter to make room for the new cell.

submit : boolean

Overrides self.autoSubmitBatch only when set to `true`. Automatically submits a batch job to the supercomputer.

A.4 settings(self,batches,inactive,particles,distribution=None)

Sets up the run settings for the model.

A.4.1 Parameters

batches : int

Number of batches for the run.

inactive : int

Number of inactive batches to run before the tallies start.

particles : int

Number of particles to be used in the run.

distribution : OpenMC distribution

Default if this is set to None is a box that includes the entire reactor core. If this is specified, the user distribution bound will be used.

A.5 tallies(self,name,scores,meshDimensions,lowerLeft=None,upperRight

Sets up the tally settings

A.6 Parameters

meshDimensions : List

A list of three nubmers defining the grid dimensions.

lowerLeft : List

The lower left corner of the area for the tally. This defaults to the lower left corner of the reactor core if no value is given.

upperRight : List

The upper right corner of the area for the tally. This defaults to the lower left corner of the reactor core if no value is given.

A.7 updateMaterialObjects(self)

Updates the materials objects dictionary. This function builds a set of OpenMC materials objects from the materials data dictionary values. This should be called any time changes are made to the material data and run before modeling.

This function also updates the colorScheme that is used in default plotting.

A.8 xplot(self, x=0, resFactor=8,figsize=None)

Plots the reactor using a slice parallel to the zy plane and through the x coordinate given by the x parameter. The resFactor variable sets the resolution of the image. The figsize sets the display size for the image. If None, then only one image is plotted using the default OpenMC size.

A.8.1 Parameters

x : float

The x location at which the plot should plot through.

resFactor : int

Increase for more reactor resolution

figsize : (int,int)

Sets the size of a secondary, resized image to be plotted. I can't stop OpenMC from plotting an initial image that is small, but if this variable is set, a seoncdary image is displayed at the size defined by the user.

A.9 zplot(self, z=0, resFactor=8,figsize=None)

Plots the reactor using a slice parallel to the zy plane and through the z coordinate given by the z parameter. The resFactor variable sets the resolution of the image. The figsize sets the display size for the image. If None, then only one image is plotted using the default OpenMC size.

A.9.1 Parameters

z : float

The z location at which the plot should plot through.

resFactor : int

Increase for more reactor resolution

figsize : (int,int)

Sets the size of a secondary, resized image to be plotted. I can't stop OpenMC from plotting an initial image that is small, but if this variable is set, a seoncdary image is displayed at the size defined by the user.

A.9.2 Returns

matplotlib.image.AxesImage

The image that was just plotted.

A.10 setHXTubeConfig (self,locations)

Sets the locations of all hx tubes in the design.

A.10.1 Parameters

locations : List : [[x1,y1],[x2,y2],...[xn,yn]]

A list of x,y coordinates that define the center points of all heat exchange (HX) tubes. hxTubeLocations expects a list of x,y coordinates in format: [[x1,y1],[x2,y2],...[xn,yn]]

A.11 setDefaultHXTubeConfig (self)

Sets the hx tubes for the "default" which is the design used in Andrew's paper. [1]

A.12 updateTemp(self,material,temp) : None

Updates a temperature for a material. Automatically changes the density of the material as defined in the density function contained in the materials .csv file.

A.12.1 Parameters

material : string

The key for the material for which the temperature should be changed. For a list of keys, run self.materials.keys() on the object you are using. For example: model.materials.keys()

temp : float

The desired temperature of the material in Kelvin.

A.13 tempOffset(self,offset)

Applies a temperature offset to all materials in the reactor. Densities are adjusted according to the density calculation given in the materials .csv file.

A.13.1 Parameters

offset : float

The temperature offset that is to be applied to all of the materials in the reactor.

Bibliography

[1] Andrew Larsen, Braden Clayton, LaGrande Gunnell, Austin Bryner, Logan Brown, Nick Rollins, and Matthew Memmott. Thermal design and analysis of a passive modular molten salt microreactor concept. *Nuclear Engineering and Design*, 402:112107, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0029549322004587, doi:https://doi.org/10.1016/j.nucengdes.2022.112107.

[2] IEA. Nuclear power in a clean energy system. 2019. URL: https://www.iea.org/reports/nuclear-power-in-a-clean-energy-system.

[3] Department of Defense. Project pele, 2022. URL: https://www.cto.mil/pele_eis/.

[4] International Atomic Energy Agency. Status report - smahtr, 2016. URL: https://aris.iaea.org/PDF/SmAHTR.pdf.

[5] International Atomic Energy Agency. Status report - imsr-400, 2016. URL: https://aris.iaea.org/PDF/IMSR400.pdf.

[6] International Atomic Energy Agency. Status report - lftr, 2016. URL: https://aris.iaea.org/PDF/LFTR.pdf.

[7] International Atomic Energy Agency. Status report - thorcon, 2020. URL: https://aris.iaea.org/PDF/ThorCon_2020.pdf.

[8] International Atomic Energy Agency. Status report - msr-fiji, 2016. URL: https://aris.iaea.org/PDF/MSR-FUJI.pdf.

[9] International Atomic Energy Agency. Msfr (crns, france), 2016. URL: https://aris.iaea.org/PDF/MSFR.pdf.

[10] OpenMC Development Team. Openmc plotter, 2023. URL: https://pypi.org/project/openmc-plotter/.

Index