ANL/NEAMS-19/1

Status of the NEAMS and ARC neutronic fast reactor tools integration to the NEAMS Workbench

Updated Status Report

Nuclear Science and Engineering Division

ANL/NEAMS-19/1

Status of the NEAMS and ARC neutronic fast reactor tools integration to the NEAMS Workbench

Updated Status Report

prepared by
N. Stauff, P. Lartaud, Y. S. Jung, P. Seurin, C. H. Lee
Nuclear Science and Engineering Division, Argonne National Laboratory
K. Zeng, J. Hou
Department of Nuclear Engineering, North Carolina State University

September 30, 2019

EXECUTIVE ABSTRACT

The Workbench initiative was launched in FY-2017 within the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Integration Product Line to facilitate the transition from conventional tools to high-fidelity tools. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench was initiated in FY-2017. The ARC codes contain both legacy codes like DIF3D and REBUS-3 that were developed with over 30 years of experience, and newer NEAMS additions like MC_2-3, PERSENT and PROTEUS. The ARC integration into the NEAMS Workbench interface relies on the PyARC module which handles the pre- and post-processing of the native ARC codes input, and the runtime environment. The PyARC module together with the NEAMS Workbench interface are both released under Open Source Software licenses.

Integrating the ARC codes into the Workbench benefits directly the Advanced Reactor Technology (ART) Campaign by:

- Providing a set of controlled, maintained, documented and validated scripts to generate ARC inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration.
- Improving the user experience with the ARC codes: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, document navigation, and geometry and results visualization.
- Enabling new modeling capabilities for advanced reactor design and analyses. The PyARC module facilitates and automatizes complex calculations and workflows for reactor analysis enabling geometrical perturbations, cross-section update through depletion, etc. The Dakota/PyARC coupling in the Workbench was also demonstrated to enable mathematical optimization and sensitivity analysis/uncertainty quantification (SA/UQ) techniques with ARC neutronic simulations.
- helping users transition to high-fidelity NEAMS codes. This was demonstrated with PROTEUS integration in FY-2019 within the same input logic.

In FY-2019, the effort focused on integrating the explicit assembly management logic of REBUS, the GAMSOR code for gamma heating transport, and of the NODAL solver of PROTEUS. Additional capabilities were also integrated in response to user requests, such as the assembly power peaking calculation and a 2D result plotting capability. For verification and demonstration purposes, the ARC codes were used through the Workbench for solving the Sodium-cooled Fast Reactor Uncertainty Analysis in Modelling (SFR-UAM) benchmark problems, including the newly proposed unit cells and ASTRID problem. Significant effort in FY-2019 focused on training new users from ANL, INL, NCSU, and Westinghouse. In particular, extensive training material was developed in the form of tutorials, including documentation, sample inputs, and associated presentations.

The ARC codes are now actively used through the NEAMS Workbench by nuclear engineers at ANL, INL, NCSU, and Westinghouse, for LFR, MSR, micro-reactor, and SFR core design analyses. Future efforts will focus on adding new and existing modeling capabilities available with the ARC and NEAMS codes, training new users and supporting them to continue building user experience.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 Introduction

One of the objectives of the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Integration Product Line (IPL) is to facilitate the deployment of the high-fidelity codes developed within the NEAMS program. The Workbench [1] initiative was launched in FY-2017 by the IPL to facilitate the transition from conventional tools to high fidelity tools [2]. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. The integration of the Argonne Reactor Computation (ARC) suite of codes into the NEAMS Workbench was initiated in FY-2017 [3, 4].

The ARC suite of codes [5] gathers neutronics, thermal hydraulics, safety, and fuel behavior analysis codes. The current focus of the Workbench integration is on the deterministic neutronic codes. It includes MC_2-3 [6] for multi-group cross-section processing, DIF3D [7] for flux calculation, REBUS-3 [8] for depletion and equilibrium calculations, PERSENT [9] for perturbation theory calculations (perturbation, sensitivity and uncertainty quantification), and GAMSOR [10] for gamma heating calculations. These ARC codes are used at national laboratories, universities, and companies for advanced reactor analyses. They gather more than 30 years of development, went through extensive validation and verification, and can solve complex physics phenomena in a very efficient way. However, these codes require knowledge on reactor physics and experience on fast reactor design in order to be familiar with the extent of their capabilities, and users mostly rely on scripts, developed based on their experiences, to generate inputs. Integrating the ARC codes into the NEAMS Workbench was initiated to address these challenges and to improve user experience with these codes by taking advantage of the various benefits brought by the Workbench interface. In FY-2019, the focus has also been on starting integration of the PROTEUS code [11, 12] developed under the NEAMS program.

The status of the PROTEUS and ARC integration [13] is described in this report, focusing on FY-2019 developments and on description of the new released version 1.0.0 of PyARC. Both the Workbench and PyARC are now being distributed under Open Source Software licenses. The ARC and PROTEUS integration framework in the PyARC bundle of the Workbench is reminded in Chapter 2. The status of the capabilities integrated in PyARC are discussed in Chapter 3. The developed capabilities were verified through application to the Sodium-cooled Fast Reactor Uncertainty Analysis in Modelling (SFR-UAM) benchmark exercises as described in Chapter 4. Finally, Chapter 5 draws the conclusions and discusses future developments.

2 Framework for ARC and PROTEUS Integration

Figure 2-1 illustrates how the Workbench interface connects with the ARC codes. This is a "black box" type of integration where the Workbench must rely on an opaque runtime module (called PyARC) that conducts the native input formatting. One of the benefits of the "black box" type of integration is that the user is shielded from the original input of the legacy codes. There are several components to the integration that are described in this section:

- Workbench interface: It is developed at ORNL and several components of this interface are required for a code's integration:
    - Common input
    - Templates
    - Visualization
- PyARC module: this is a python module required for "black box" integration that contains the logic for processing the code's inputs, generating the legacy ARC code input, running them, and post-processing the outputs. This module is the glue between the Workbench interface and the ARC codes.

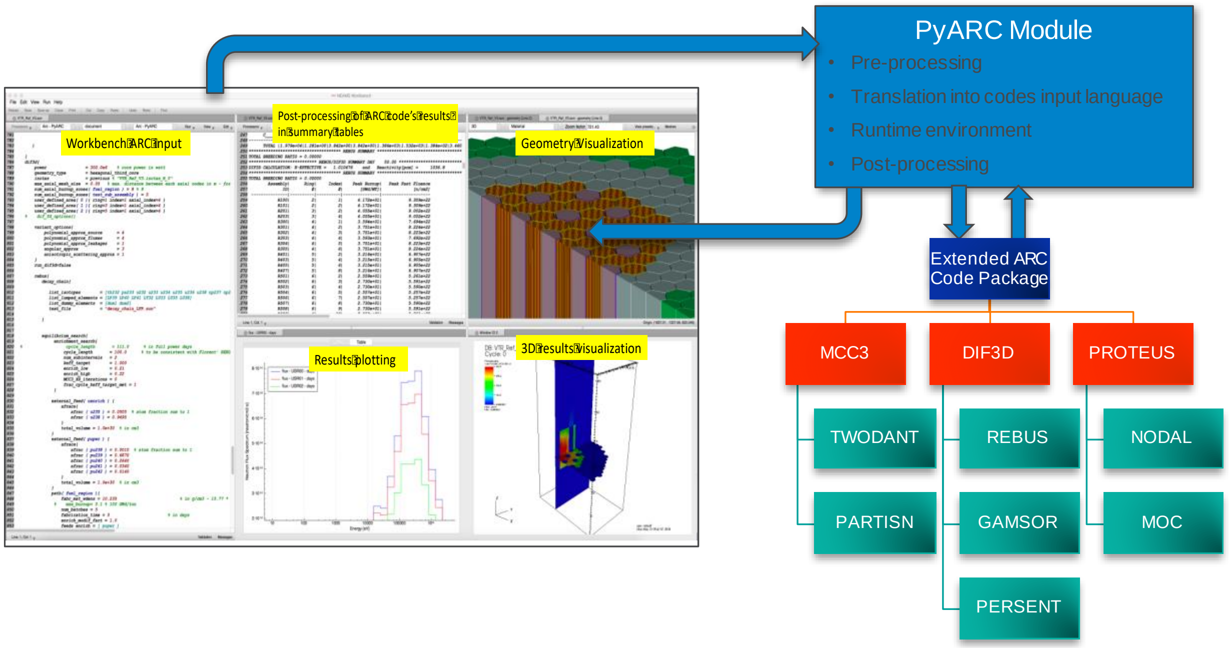


Figure 2-1. Structure of the ARC integration in the Workbench.

2.1 The Workbench Interface

The Workbench [1, 14] interface is developed at ORNL and designed to assist new users, while not obstructing experienced ones. The Workbench provides a common user interface for model creation, real-time validation, execution, output processing, and visualization for integrated codes. For instance, the user is guided by the auto-completion capability in the Workbench to build its core model in the "common input" structure.

2.1.1 Common input

The Workbench input format adopted is described as the "common input" since it is used to generate inputs for MC_2-3, DIF3D, REBUS-3, and PERSENT for integrated problem-dependent cross-section preparation, core analysis, depletion, and sensitivity/uncertainty quantification. The main benefits of the "common input" strategy is to insure every native input uses consistent information and to facilitate project collaboration (since one PyARC input contains all the problem definition, while the MC_2-3, REBUS-3, PERSENT, and PROTEUS inputs only contain part of the information).

This "common input" allows modeling a reactor geometry in an intuitive and flexible way and was developed with continuous involvement of ARC users. It uses the open source Workbench Analysis Sequence Processor's (WASP) Standard Object Notation (SON) [15] format that enables the auto-completion, real-time input validation, and access to templates, through the Workbench interface.

The structure of the input is shown in Figure 2-1 and a tutorial was developed (detailed in Section 2.4) to explain in detail the input logic to new users. The common input is defined in the "*arc.sch*" file that takes ~5000 lines of code. Detailed documentation of all the input options is provided (in the "*PyARC_README.html*" file of the PyARC package). The user has direct access to the input keyword definitions through the Workbench user-interface upon auto-completion. The input is automatically validated for correctness by the WASP Hierarchical Input Validation Engine (HIVE) upon edit by the user from within the NEAMS Workbench and upon by PyARC.

2.1.2 Templates

The Workbench, through its WASP subcomponent, contains the HierarchicAL Input Template Engine (HALITE) developed to expand hierarchical input data into code-specific input. Templates are used to assist users in generating the common input within the Workbench. The common input templates were developed in parallel to the schema and the common input. Those are blocks of input with default values accessible for convenience to the user. A total of **112 templates** were generated for the ARC codes. Templates are also relied upon by the Dakota/PyARC coupling, as explained in Section 2.3.

2.1.3 Visualization

The Workbench provides different built-in types of visualization capabilities that the ARC integration benefits from. In particular, it provides visualization of user input problems through built-in input visualization capabilities (see Figure 2-1). This visualization capability supports 2D/3D hexagonal and Cartesian geometries.

The VisIT tool [16] is integrated into the Workbench and allows direct visualization of the ARC post-processed outputs, as illustrated in Figure 2-1. The Workbench interface supports plotting capabilities using line plots, histograms, bar charts, etc. Two types of line plots were implemented to display the multi-group cross sections processed by MC_2-3 (as illustrated in Figure 2-1), and the region-wise flux spectrum printed by DIF3D and REBUS-3 (also shown in Figure 2-1).

New in V1.0.0: a script was developed to provide the user 2D plot generation of the core geometry, as shown in Figure 2-2, and of the assembly peak or integrated results obtained with DIF3D in standalone-DIF3D simulations (as illustrated in Figure 3-3) or together with GAMSOR, PERSENT, REBUS, or PROTEUS. In addition, the "*.user_info.out*" file is generated to provide the user some important information about its model: errors, warnings, description of the automatically computed 1D and RZ geometries (see Section 3.1) and of the volume fractions. In particular, the list of isotopes and region IDs are detailed to facilitate advanced ARC users understanding the PyARC-generated native code inputs.

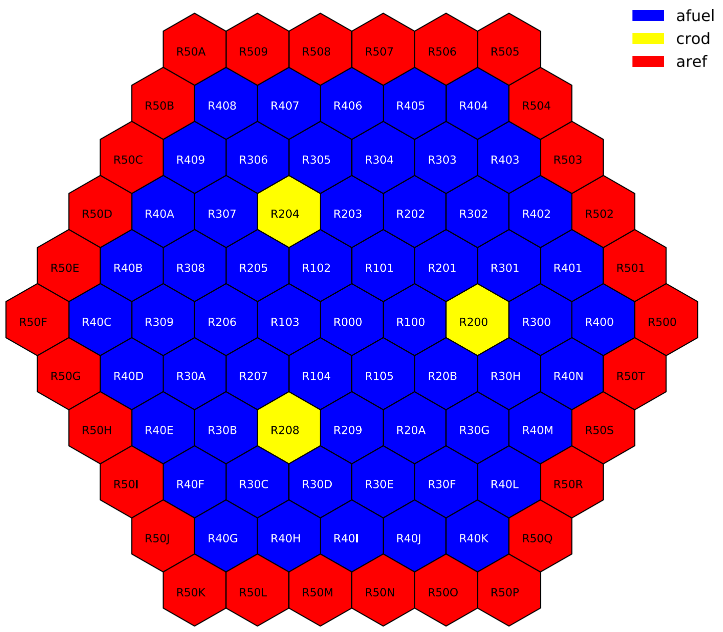


Figure 2-2. Automatic generation of core layout.

2.2 PyARC Module

The PyARC module is the glue between the Workbench interface and the ARC codes. It is being released by ANL under an open-source software license and is bundled with the Workbench install for user convenience. For a "black box" integration, this wrapper is essential as it contains the logic to:

- process information from the common input
- perform additional verifications on the core model that the validation engine of the Workbench cannot perform
- pre-process the information, calculating for instance homogenized atom densities in different regions
- generate the ARC codes' native inputs
- handle the runtime environment, which can be very complex. For instance, running MC_2-3 elementary cell calculations in parallel to calculate fine-mesh cross-sections, followed by TWODANT to calculate region-wise flux spectrum, then by MC_2-3 for broad-mesh condensation of the cross-sections, then by REBUS to calculate depleted compositions, then by PERSENT to calculate neutronic feedback coefficients on depleted core compositions, etc.

- post-process the outputs, gathering the main results of the different codes' and creating a single summary file.

The PyARC module gathers ~13,500 lines of Python code developed through a collaborative environment on GitLab so that new additions are tracked and reviewed. The PyARC module relies on the following sub-modules:

- PyARCModel: loads the input, performs list of additional verification, performs pre-processing on the input
- PyARCUtils: contains utilities procedures
- PyARCUserObject: defines variables and procedures that are used throughout the code
- PyMCC3, PyREBUS, PyPERSENT, PyGAMSOR, PyPROTEUS (includes PyPROTEUSMOCObject, PyPROTEUSNodalObject, PyPROTEUSMSRObject): contain the logic for input writing, execution, and post-processing for each code
- PyRzmflxCode: contain the logic for input writing, execution, and post-processing for TWODANT and PARTISN
- The PyARC module also relies on the PySCL module that is developed by ORNL to provide the standard composition library (SCL)

Tests are developed for regression testing after each code modification and prior to committing and pushing modifications to the protected master branch. Currently, **190 unit tests** are implemented to check the common input processing, interpretation of the standard composition library, input generation (of MC_2-3, DIF3D, REBUS-3, TWODANT, PARTISN, PERSENT, GAMSOR, PROTEUS), execution of the codes, and post-processing of the outputs. Consequently, the unit tests check the pre-processing, input writing, execution, and post-processing logic of PyARC.

ORNL also implemented the continuous integration (CI) testing and deployment (CD) bundle infrastructure for the PyARC software. This checks all tests at each 'push' to the code repository and ensures all features are functional on Linux and Mac operating systems, and subsequently bundles the new PyARC version into an easily deployable file.

2.3 PyARC Coupling with Dakota

The Workbench provides a common user interface for model creation allowing for its integrated codes to communicate and work together with limited coupling development. The feasibility and benefits of using the Workbench as a coupling mechanism between the Dakota [17] code and PyARC was demonstrated in FY-2018 [4, 18]. The Dakota software maintained by Sandia National Laboratory is a sensitivity analysis/uncertainty quantification (SA/UQ) and optimization toolkit with over 20 years of supporting development. Dakota provides advanced mathematical methods to vary one code's input parameters and analyze the output results for optimization and uncertainty quantification analyses.

2.3.1 Workflow Implemented

The workflow in Figure 2-3 was developed to allow Dakota to drive the PyARC calculations within the Workbench interface. For SA/UQ or optimization analyses, the PyARC input is perturbed by a sequence of random values from Dakota. After the ARC runs are performed with different sampled input values, Dakota evaluates the user-specified responses of interest. For SA/UQ types of analyses, Dakota performs statistical analysis on response functions. For optimization problems, it selects best performing solutions and generate new samples.

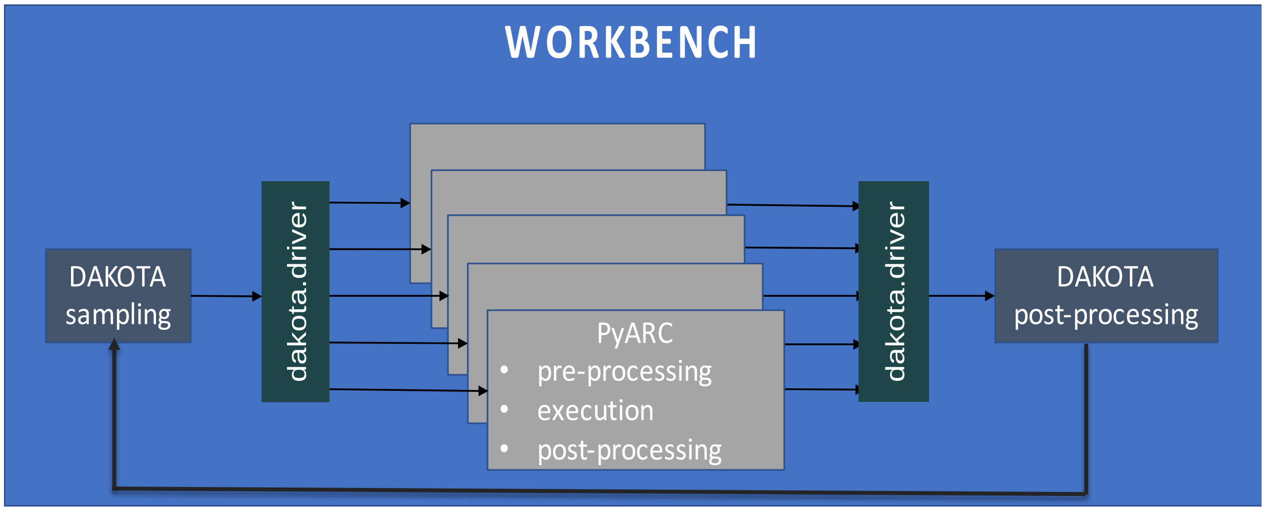


Figure 2-3. Schematic of the Dakota/PyARC coupling.

Three files are required in this workflow:

- Dakota input: Input built by the analyst with the aid of the Workbench that describes the sensitivity, uncertainty or optimization problem options.
- PyARC "common input": This is the PyARC input built by the analyst with the aid of the Workbench, but saved in a **template** format where some input values are replaced with variables defined in the Dakota input.
- PyARC.driver: Input built by the analyst with the aid of the Workbench that connects the Dakota input to the PyARC application. It contains the logic to extract (with customizable grep commands or links to post-processing scripts) different results from the ARC output summary file.

New in V1.0.0: the main PyARC results of interest (core lifetime, inventory, fissile enrichment, peak power, peak fast flux, etc.) are returned in the summary output file so that the user does not need to develop his own postprocessor logic to extract such results to return to Dakota. An example of Dakota/PyARC coupling is detailed in Sample #11.

2.3.2 *Benefits of the Dakota/PyARC Coupling*

The Dakota/ARC coupling would be a significant effort to set up outside of the Workbench since the individual ARC codes use different input logic. This would require developing a script that propagates the input parameters sampled throughout the different codes, which is effectively done with the PyARC module. This coupling enables new capabilities for ARC users since Dakota can be used for driving sensitivity analysis and uncertainty quantification (SA/UQ) calculations [18, 19], and core design optimization with the ARC codes [20, 21].

2.3.2.1 Optimization problems

Mathematical optimization methods can be used to investigate a space of input options and find the most promising solutions (usually a compromise between targeted performance). This is especially well suited for advanced reactor design work, where many core options need to be evaluated (size and number of fuel pins, different fuel forms, etc.) to assess their impact on core performance (irradiation testing capabilities, inherent safety performance, etc.). Dakota provides a wide range of advanced optimization methods that can be used to effectively investigate the design space and find the best performing core concepts.

2.3.2.2 SA/UQ problems

Dakota extends the SA/UQ capability currently available in the ARC codes (with PERSENT) to propagate the uncertainty on any type of input parameter, and to observe its impact on any output result. In fact, there are different benefits/challenges associated with solving SA/UQ problems through adjoint-based perturbation theory (available with PERSENT [9]) and through stochastic sampling (with Dakota [17]) making both approaches complimentary to each other. The adjoint-based method is usually cheaper in terms of computational resources, is well suited to treat a large number of uncertain parameters such as uncertainties on multi-group cross-sections, and can provide detailed information such as the impact of cross-section values in any energy group, in any core location, on different core parameters. It is usually applied to see the uncertainty impact on the eigenvalue, on reactivity effects and on reaction rates. The stochastic sampling method provides a more general approach that can be applicable to any uncertainty problem considered (including those with changes in core geometry), to analyze the impact uncertain parameters may have on any output of the problem. However, it may require many simulations to reach targeted levels of confidence. An analysis using Dakota to drive PyARC simulations for SA/UQ problems is proposed in Appendix C.

2.4 Training Material

New in V1.0.0: Training material was developed to assist a user getting started. It consists in a list of sample problems that are documented and that demonstrate and explain the most popular ARC capabilities:

- Sample #1 - MC2-3 calculation with homogeneous - 1step calculation, and DIF3D diffusion.
- Sample #2 - DIF3D calculation with VARIANT.
- Sample #3 - MC2-3 calculation with heterogeneous - 2steps (TWODANT) calculation, and with DIF3D finite diffusion.
- Sample #4 - MC2-3 calculation with homogeneous - 1step calculation, and REBUS once-through fuel depletion calculation with DIF3D finite difference option with third core symmetry.
- Sample #5 - REBUS equilibrium calculation with DIF3D finite difference option with third core symmetry (no reprocessing).
- Sample #6 - REBUS equilibrium calculation with DIF3D finite difference option with third core symmetry (with reprocessing and one iteration between MC2-3 and REBUS equilibrium).

- Sample #7 - GAMSOR calculation.
- Sample #8 - REBUS once-through fuel depletion calculation with explicitly defined fuel management strategy and third core symmetry.
- Sample #9 - PERSENT perturbation calculations to process needed reactivity coefficients.
- Sample #10 - PERSENT sensitivity calculations to perform SA/UQ analyses on k-eff and reactivity coefficients.
- Sample #11 - Dakota coupling with PyARC to run SA/UQ and Optimization problems
- Sample #12 – PROTEUS-NODAL calculation
- Sample #13 - PROTEUS-NODAL calculation with molten salt fuel (MSR)

Those Sample problems are available in the "*PyARC/tutorial*" folder that also contains the AdditionalFiles folder where the user has access to different decay chains, lumped fission products, and covariance matrices.

3 Capabilities Integrated in PyARC

One of the benefits of the “black box” type of integration adopted with the PyARC module is that the user is shielded from the original input of the legacy codes. The associated challenge is that some of the options and code’s capabilities may not be made available to the user through the “black box”. The ARC integration focuses on the popular and important capabilities of each code to streamline most user’s workflows. This section summarizes the status of the ARC codes integration within the NEAMS Workbench completed in FY-2019. Continuing efforts are underway to implement additional modeling capabilities based on the ARC and NEAMS codes.

3.1 MC^2-3 [6]

The MC$_2$-3 code is developed within the NEAMS program for multi-group cross-section processing for both fast and thermal spectrum reactors. From the Workbench, one can generate cross-sections for pre-generated or user-defined energy-group structure with different scattering orders. The user can merge cross-sections to define lumped fission products used in REBUS-3. Neutron slowing down equation can be solved over a homogenized cell or over a heterogeneous geometry [22] based on 1D cylindrical or slab geometries.

For region-wise group condensation, two approaches were implemented within the Workbench. The first approach consists in generating neutron leakage files from the fuel regions that can be used as external sources in the non-fuel regions (for instance, reflector), as demonstrated in Sample #1. The second approach consists of using TOWDANT [23] or PARTISN [24], those are S$_n$ neutron transport equation solvers, for fine-group (1000 – 2000 groups) flux calculation using an equivalent 2D (RZ or XY) core model. This approach is demonstrated in Sample #3. In terms of output processing, the multi-group cross-sections generated in the ISOTXS file can be plotted automatically in the Workbench interface, as illustrated in Figure 3-1.

Upon completion, PyARC returns the TWODANT, PARTISN and MC$_2$-3 inputs and outputs (in the “*.inp*” and “*.out*” files) and the multi-group cross-section files:

- The “*.isotxs*” files are binary files that can be re-used by DIF3D/REBUS/PERSENT to avoid re-running the initial MC$_2$-3 calculation
- The “*.isotxs.edit*” files are text files containing all the multi-group XS results that can be opened directly with the Workbench to plot automatically the cross-section using the “ISOTXS – ISOTOPE XS” processor (as shown in Figure 3-1).

Different isotxs files may be generated at different depletion steps, the following nomenclature logic is used:

- “*.isotxs_R_0*”: Reference (un-perturbed), depletion time-step 0 (provided composition or beginning of equilibrium cycle);
- “*.isotxs_D_4*”: “D” perturbation, depletion time-step 4.

New in V1.0.0:

The option to give a lower threshold to the atom density in heterogeneous regions using the option “*min_dens_het_calc*”. This option is especially critical for simulating coolant void coefficient when using the heterogeneous treatment in MC$_2$-3 since its 1D transport solver provides convergence issues with very low-density regions.

MC2-3 is also applied for computing the GAMISO and PMATRX required for GAMSOR calculations, as further discussed in Section 3.5.

Automatic generation of RZ geometry for TWODANT or PARTISN and 1D geometries for MC2-3 were implemented to facilitate the use of these options by reducing pre-processing time to the user and risks of mistakes. These methods are only available for hexagonal-z core geometries. Description of the methods developed and benchmark calculations is provided in Appendix A. Sample #3 provides an example to get started with these options.

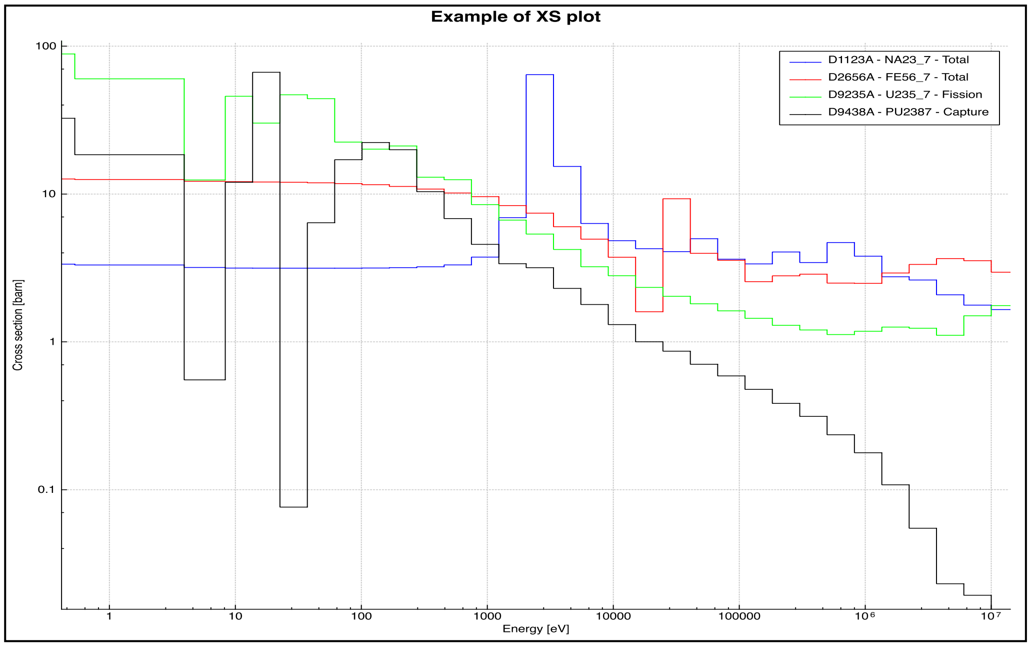


Figure 3-1. Example of multi-group XS plot automatically generated with the Workbench from ISOTXS edit file.

3.2 DIF3D [7]

The DIF3D code is a legacy ARC tool used for neutron and gamma flux calculations on various types of geometries, based on pre-generated multi-group cross-sections. The multi-group cross-sections can be generated using MC2-3 calculations or a compatible set of previously calculated multi-group cross-sections.

The 2D/3D-Hexagonal and 2D/3D-Cartesian types of geometries are supported through the Workbench. The DIF3D code includes 3 neutron solvers (Nodal, Finite Difference, and VARIANT [25]) that were all enabled. This is illustrated with Samples #1 (Finite Difference) and #2 (VARIANT). Both neutron flux and gamma flux (discussed in Section 3.5) calculations are integrated into the Workbench.

PyARC returns the full DIF3D input and output (in the "*.inp*" and "*.out*" files). Post-processing of DIF3D output was implemented by printing the main information of interest to a user (e.g., the neutron flux in different core areas, integrated flux and power per assemblies) in the summary file ("*.summary*"). When opening this "*.summary*" file with the Workbench, the user can use the "*flux spectrum*" processor to automatically plot the neutron flux spectrum. Direct visualization of the power density, neutron flux, atom densities, etc., is enabled by opening the generated ".*vtk*" file with VisIT through the Workbench, as also illustrated in Figure 3-2.

As discussed in Section 2.1.3, new 2D visualization is available, through an external script run on the generated summary file, or direct through the PyARC workflow by using the input line: "*calculations/plot_2d = true*". An example of peak power density radial distribution is provided in Figure 3-3.

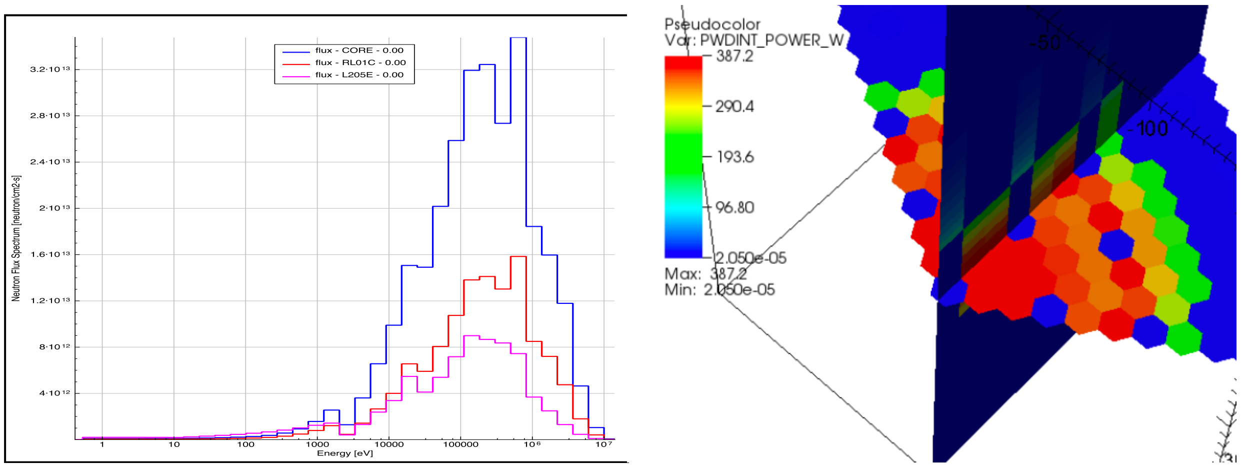
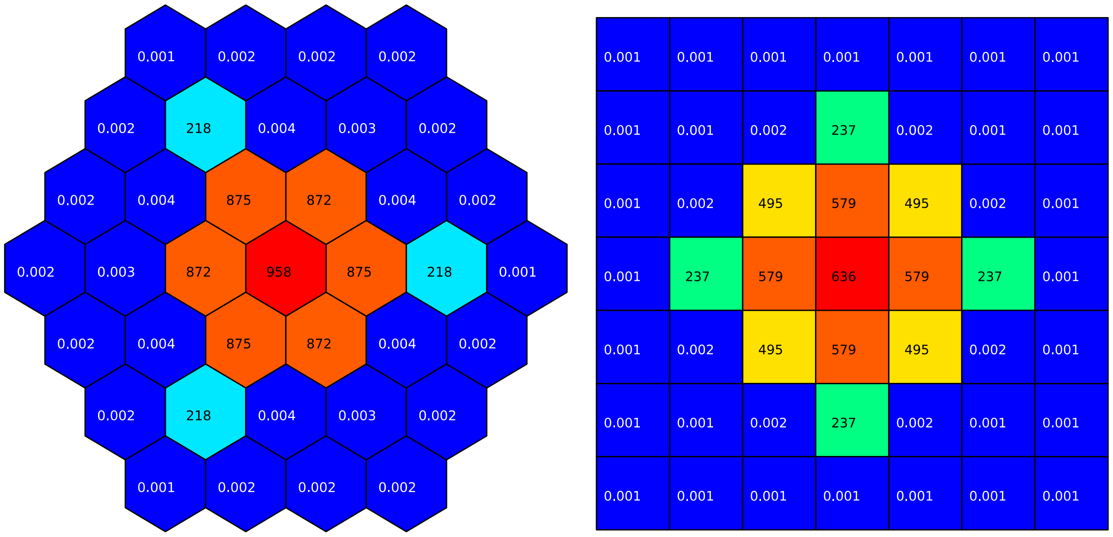


Figure 3-2. Example of visualizations available for DIF3D and REBUS-3: neutron flux spectrum (left) and power map (right) calculated and plotted with the Workbench.



0.001	0.001	0.001	0.001	0.001	0.001	0.001
0.001	0.001	0.002	237	0.002	0.001	0.001
0.001	0.002	495	579	495	0.002	0.001
0.001	237	579	636	579	237	0.001
0.001	0.002	495	579	495	0.002	0.001
0.001	0.001	0.002	237	0.002	0.001	0.001
0.001	0.001	0.001	0.001	0.001	0.001	0.001

Figure 3-3. Example of 2D plot visualization of peak power density per assembly for hexagonal and Cartesian geometry.

3.3 REBUS-3 [8]

REBUS-3 is a legacy ARC code used for fuel cycle analysis using DIF3D solvers. It allows a wide range of fuel cycle modeling options such as assembly shuffling, enrichment or cycle length search at equilibrium state. In terms of post-processing, the same capabilities developed for

DIF3D are made available with REBUS-3 at every time-step of the depletion calculation. In particular, the "*.rebus_X.vtk*" file is generated by REBUS-3 at time *X* days. Some specific information are also printed out in the "*.summary*" file such as the peak burnup and fast fluence, the enrichment modification factor, etc.

The main challenge associated with integrating REBUS-3 is to allow a user-specified decay chain, without making the Workbench input too complicated. This was achieved by directly providing an external text file containing the decay chain input from REBUS-3 (cards 09, 24, 25), which is being parsed in PyARC.

The once-through depletion capability is illustrated in Sample #4. The original option to re-calculate the multi-group cross-sections at different time-step of the depletion was implemented in the case of the once-through depletion, as illustrated in Figure 3-4. This capability can be especially relevant when modeling very high burnup fuel or a reactor with thermalized neutron flux.

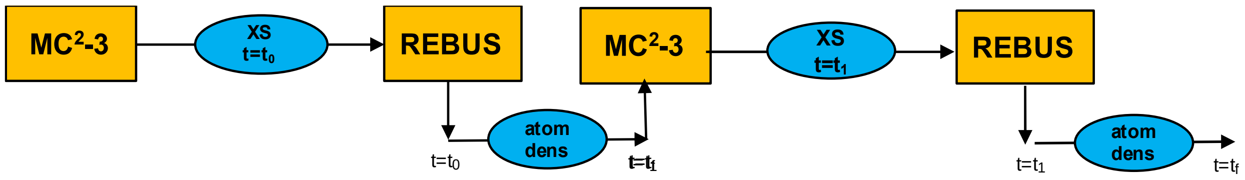


Figure 3-4. Multi-step depletion procedure implemented in PyARC.

The enrichment or cycle length search options at equilibrium state were also integrated, where the user can define reprocessing plants, external feeds, and fuel-cycle strategies, as illustrated in Samples #5 and #6. The original option was implemented to iterate between the multi-group cross-sections computed with MC2-3 at beginning of equilibrium cycle, and the equilibrium search calculation performed with REBUS-3, as illustrated in Figure 3-5.

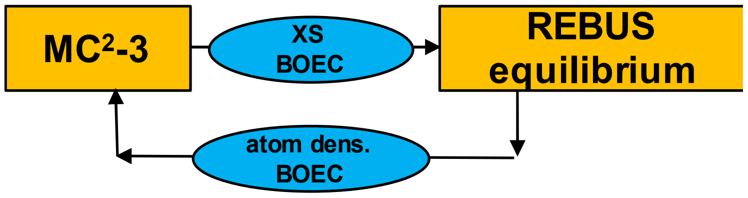


Figure 3-5. Equilibrium cross-section iteration procedure implemented in PyARC.

New in V1.0.0: The explicit fuel management capability of REBUS was enabled using the "*once_through_shuffling*" option. This option allows defining different cycles and the paths for each assembly, allowing to discharge, move or reload assemblies. Sample #8 illustrates this capability and explains the input logic.

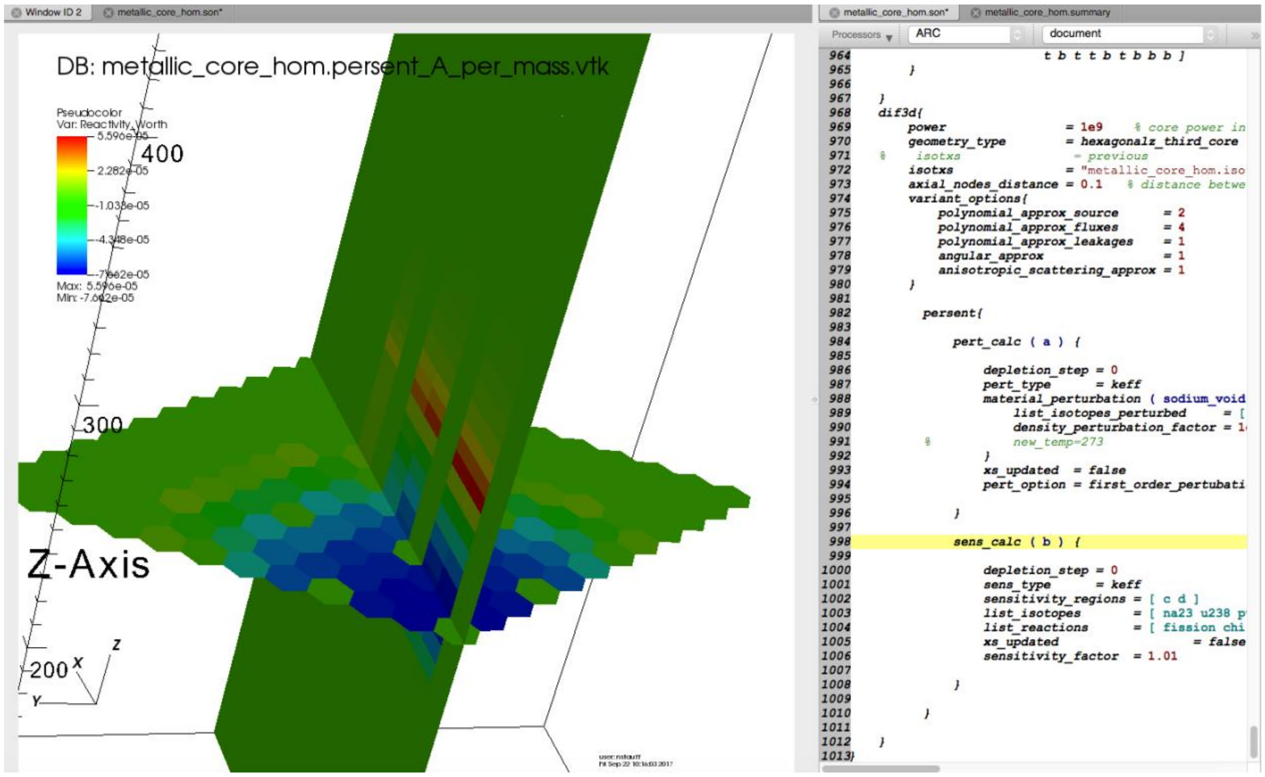
3.4 PERSENT [9]

PERSENT is a perturbation theory code developed within the NEAMS program and based on the neutron transport equation in a 2D or 3D geometry. It allows calculating reactivity feedback coefficients, sensitivity coefficients [26], and nuclear data uncertainties. Perturbation and sensitivity calculations were implemented on eigenvalue, beta and lambda problems and can be automatically run at different depletion steps (computed with REBUS-3). The user can define which materials are perturbed with a change in density or in temperature or which surfaces are perturbed (only for direct DIF3D perturbations, as explained below). The cross-sections of the perturbed composition can be automatically re-calculated both for perturbation and for sensitivity calculations. The nuclear data uncertainties can be estimated on the eigenvalue, beta, lambda, and on the reactivity coefficients automatically by providing a covariance matrix (in a PERSENT-compatible file format).

Direct DIF3D calculations are enabled as an alternative to PERSENT perturbation theory calculations. This can be especially useful for double checking the perturbation theory result and for modeling geometric perturbations such as the axial and radial feedback coefficients or the control rod worth. Second, to optimize the workflow of PERSENT calculations, one added the option to perform preliminary un-perturbed DIF3D calculations to use generated flux files (adjoint and forward) in different PERSENT runs. Finally, every PERSENT or DIF3D calculations can be run in parallel on different CPUs.

Visualization of the perturbation results is enabled within the Workbench using VisIt [13]: for instance, "*.persent_**P**_ref.vtk*" is the vtk file generated by PERSENT for perturbation ***P***. For illustration purposes, Figure 3-6 shows the distribution of the sodium void worth calculated on an SFR design and plotted by VisIt within the Workbench.

New in V1.0.0: Perturbation calculations can now be run based on already perturbed geometry or compositions using the "*ref_is_pert_config*" option. This allows simulation of the voided Doppler coefficient, or of any reactivity coefficients with control rods inserted at critical location through a geometry perturbation. Samples #9 and #10 were developed to train users in correctly computing reactivity coefficients feedback as required for safety analyses (using SAS4A/SASSYS).



```
                        t b t t b t b b b ]
    }

}
dif3d{
    power                = 1e9     % core power in
    geometry_type        = hexagonalz_third_core
%    isotxs               = previous
    isotxs               = "metallic_core_hom.iso
    axial_nodes_distance = 0.1    % distance betwe
    variant_options{
        polynomial_approx_source      = 2
        polynomial_approx_fluxes      = 4
        polynomial_approx_leakages    = 1
        angular_approx                = 1
        anisotropic_scattering_approx = 1
    }

      persent{

          pert_calc ( a ) {

              depletion_step = 0
              pert_type      = keff
              material_perturbation ( sodium_void
                  list_isotopes_perturbed      = [
                  density_perturbation_factor = 1
%                 new_temp=273
              }
              xs_updated  = false
              pert_option = first_order_pertubati

          }

          sens_calc ( b ) {

              depletion_step = 0
              sens_type      = keff
              sensitivity_regions = [ c d ]
              list_isotopes       = [ na23 u238 p
              list_reactions      = [ fission chi
              xs_updated                  = false
              sensitivity_factor  = 1.01

          }

      }

  }
}
```

Figure 3-6. Sodium void worth distribution [kg-1] calculated and plotted within the Workbench.

3.5 GAMSOR [10]

New in V1.0.0: The GAMSOR code is a legacy code developed within the ARC suite to assist analysts in calculating gamma heating. GAMSOR computes the gamma flux through a sequence of MC2-3 for neutron and gamma cross-section preparation, and DIF3D calculations to solve the neutron flux, the gamma flux, and then to combine the results for summary edition. This complex workflow is summarized in Figure 3-7.

The GAMSOR Workflow was fully integrated within the PyARC interface in FY-2019 and Sample #7 was developed as training material. The GAMSOR user input proposed through PyARC only requires the energy-group structure of GAMMA calculation and the list of depletion time-steps on which to perform GAMMA calculations.

In terms of post-processing, similar capabilities as developed for DIF3D are made available, as illustrated in Figure 3-8. The user has access to summary tables with assembly-integrated neutron and gamma powers in the "*.summary*" file. Automatic 2D plotting of the power map is also available. The VisIt visualization tool can be used for 3D visualization of the neutron and gamma power. Finally, the region-wise neutron and gamma power levels are provided back to the user in the "*.zip/gamsor_table_*.out*" files.

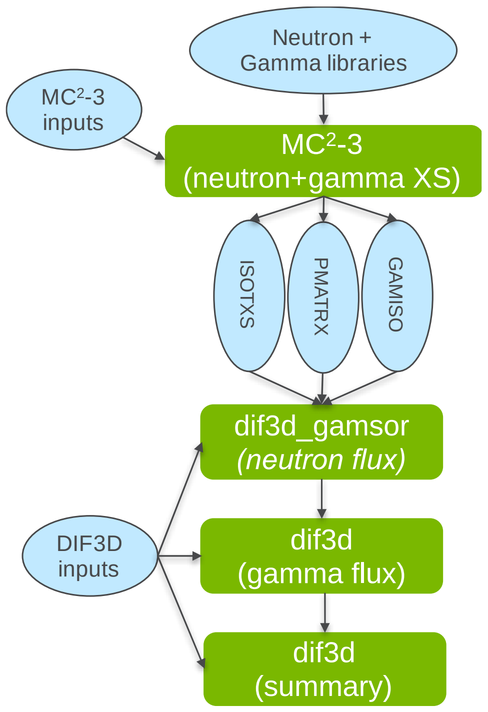
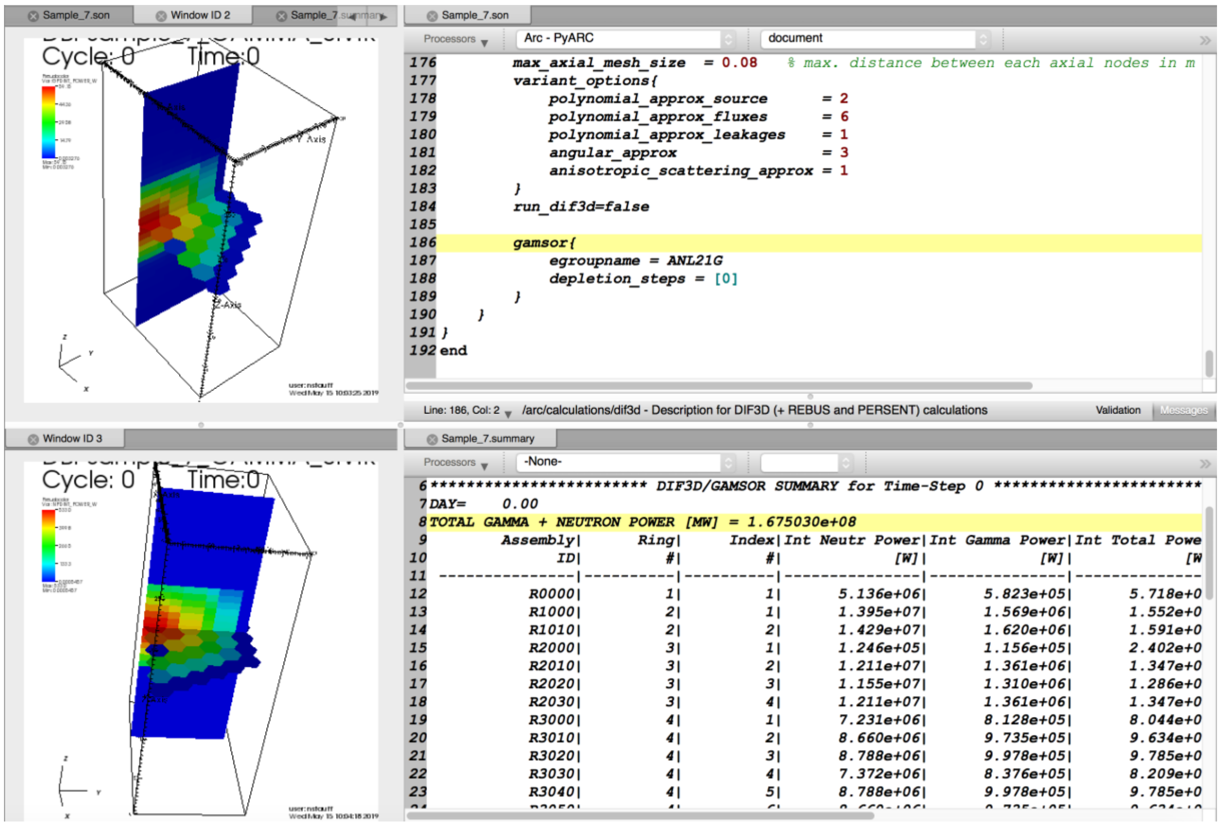


Figure 3-7. GAMSOR Workflow implemented in PyARC.



```
        max_axial_mesh_size  = 0.08   % max. distance between each axial nodes in m
        variant_options{
            polynomial_approx_source      = 2
            polynomial_approx_fluxes      = 6
            polynomial_approx_leakages    = 1
            angular_approx                = 3
            anisotropic_scattering_approx = 1
        }
        run_dif3d=false

        gamsor{
            egroupname = ANL21G
            depletion_steps = [0]
        }
    }
}
end
```

Line: 186, Col: 2 /arc/calculations/dif3d - Description for DIF3D (+ REBUS and PERSENT) calculations

```
************************ DIF3D/GAMSOR SUMMARY for Time-Step 0 ************************
DAY=    0.00
TOTAL GAMMA + NEUTRON POWER [MW] = 1.675030e+08
       Assembly|       Ring|      Index|Int Neutr Power|Int Gamma Power|Int Total Powe
             ID|          #|          #|            [W]|            [W]|            [W
 --------------|-----------|-----------|---------------|---------------|--------------
          R0000|          1|          1|      5.136e+06|      5.823e+05|     5.718e+0
          R1000|          2|          1|      1.395e+07|      1.569e+06|     1.552e+0
          R1010|          2|          2|      1.429e+07|      1.620e+06|     1.591e+0
          R2000|          3|          1|      1.246e+05|      1.156e+05|     2.402e+0
          R2010|          3|          2|      1.211e+07|      1.361e+06|     1.347e+0
          R2020|          3|          3|      1.155e+07|      1.310e+06|     1.286e+0
          R2030|          3|          4|      1.211e+07|      1.361e+06|     1.347e+0
          R3000|          4|          1|      7.231e+06|      8.128e+05|     8.044e+0
          R3010|          4|          2|      8.660e+06|      9.735e+05|     9.634e+0
          R3020|          4|          3|      8.788e+06|      9.978e+05|     9.785e+0
          R3030|          4|          4|      7.372e+06|      8.376e+05|     8.209e+0
          R3040|          4|          5|      8.788e+06|      9.978e+05|     9.785e+0
```

Figure 3-8. Example of enabled GAMSOR input and result visualization.

3.6 PROTEUS

New in V1.0.0:

The PROTEUS code developed under NEAMS project is a high-fidelity deterministic neutron transport code based on unstructured finite element meshes, which solves the steady-state and transient neutron transport problem using the method of characteristics (MOC) or the discrete ordinate (SN) method as high-fidelity neutron transport solvers. Additionally, the nodal transport method (NODAL) option for structured geometries is available to provide a fast running solution option within the same framework so that a user can choose a level of solution fidelity and computational resource requirements depending on its need. In FY-2019, an integration of PROTEUS codes into the NEAMS Workbench interface was initiated to improve the usability by taking advantage of the PyARC framework. For the PROTEUS integrations, the extension of the PyARC module referred to its PyPROTEUS sub-module was developed for connecting the Workbench interfaces using the "black box" approach of code integration. Figure 3-9 illustrates how the Workbench interface connects with the PROTEUS codes through the PyPROTEUS/PyARC wrappers. The followings are the integration status of PROTEUS codes in FY-2019:

- *NODAL*: Fully integrated for steady-state calculations. The integration supports all the features of the Workbench/PyARC framework (input generation, workflow management, post-processing).

- *MOC*: Partially integrated for steady-state and transient calculations. Requires off-line mesh and cross section generation since this is currently not supported under the PyARC common user interface model creation.
- *SN*: Not integrated yet.

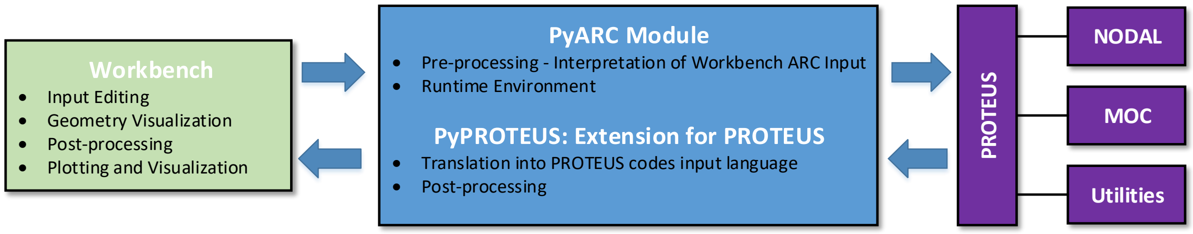

Workbench
Input Editing
Geometry Visualization
Post-processing
Plotting and Visualization
PyARC Module
Pre-processing - Interpretation of Workbench ARC Input
Runtime Environment
PyPROTEUS: Extension for PROTEUS
Translation into PROTEUS codes input language
Post-processing
PROTEUS
NODAL
MOC
Utilities

Figure 3-9. Structure of the PROTEUS integration in the PyARC and the Workbench.

3.6.1 *PROTEUS-NODAL [11]*

The PROTEUS-NODAL code is a nodal transport solver based on homogenized assemblies that provides a conventional fidelity level in a consistent framework of PROTEUS. Two solver methodologies were implemented on that framework that constitute the nodal solver capabilities: P_N and Simplified P_N (SP_N). The P_N approach is the identical methodology used in VARIANT although the release version only handles diffusion theory on Cartesian and hexagonal grids. For the SP_N approach, a transverse integrated nodal methodology was built on the hexagonal grid model utilizing up to a SP_3 approximation. The PROTEUS-NODAL code has capabilities to solve steady-state and transient problems. Additionally, the flowing fuel modeling capability enables to model the impact on the neutron precursor distribution for a flowing fuel in molten salt reactor (MSR) analyses. Currently, only the steady-state and MSR analysis capabilities are fully integrated into the Workbench, while transient analysis capability should be implemented in the future.

The workflow for PROTEUS-NODAL calculations was built upon the existing sub-modules for DIF3D calculations and the implemented workflow is illustrated in Figure 3-10. The PyPROTEUS modules return the following input files for PROTEUS-NODAL execution:

- *Mesh:* defines geometrical dimensions, region configurations, and boundary conditions.
- *Assignment*: defines compositions and assigns them to the geometrical regions.
- *Driver*: defines the simulation parameters such as power level, convergence criteria, and iteration limits.

Along with these PROTEUS-specific input files, the PyARC module generates the cross sections and the optional delayed neutron parameters in the ISOTXS and DLAYXS file formats respectively. The PROTEUS-NODAL calculation is executed via the runtime environment of PyARC. Once the calculation is completed, the PROTEUS-NODAL code produces three basic types of outputs as:

- *Main Text-based Screen Output:* contains confirmation that the input was imported successfully, computing timing summaries, and eigenvalue iteration history results.
- *Detailed Summary Output*: contains full solution in the entire domain which is exported to an organized ASCII file for detailed analysis.

- *Visualization Output*: contains the solutions of primary variables such as flux and power in the VTK file format which is readable by VisIt (within the Workbench) or any visualization software.

Similar to the DIF3D calculation capability, post-processing of PROTEUS-NODAL output was implemented by printing the main information of interest to a user in the summary file ("*.summary*"). When opening this "*.summary*" file with the Workbench, the user can use the "*flux spectrum*" processing to automatically plot the neutron flux spectrum. The direct visualization of the primary variables is enabled by opening the generated "*.vtk*" file with VisIT through the Workbench. The implemented post-processing capabilities are illustrated in Figure 3-11. A sample input #12 demonstrating the Nodal workflow was developed within the released tutorial.

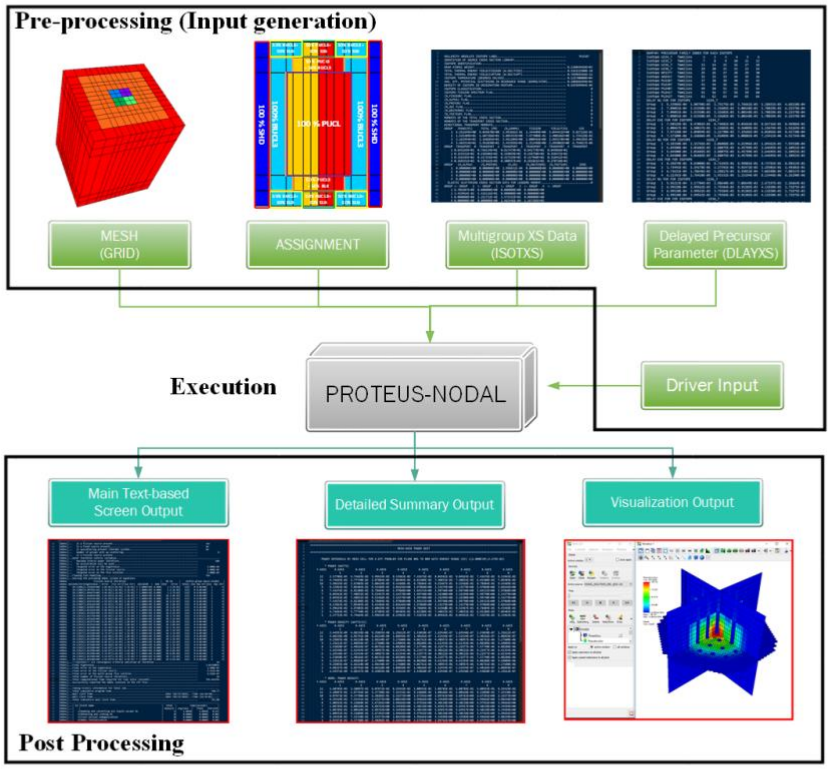
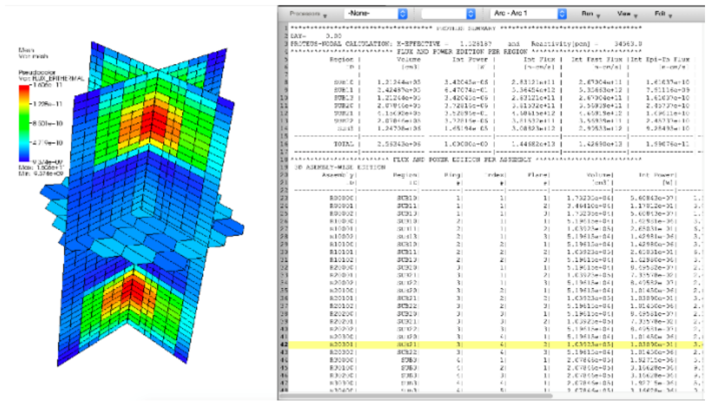

Pre-processing (Input generation)
MESH (GRID)
ASSIGNMENT
Multigroup XS Data (ISOTXS)
Delayed Precursor Parameter (DLAYXS)
Execution
PROTEUS-NODAL
Driver Input
Main Text-based Screen Output
Detailed Summary Output
Visualization Output
Post Processing

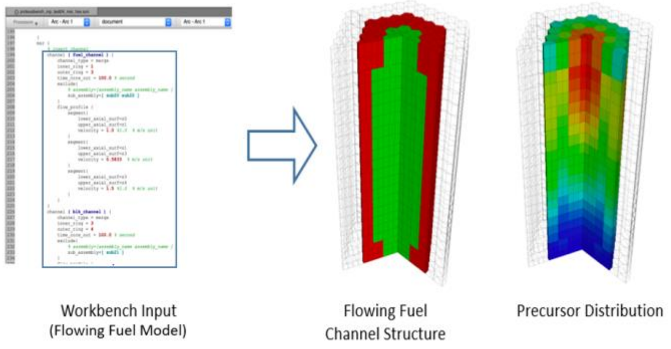
Figure 3-10. PROTEUS-NODAL Workflow implemented in PyARC.



```
************************************ PROTEUS SUMMARY ************************************
DAY=      0.00
PROTEUS-NODAL CALCULATION: K-EFFECTIVE =   1.528187    and   Reactivity[pcm] =    34563.0
************************* FLUX AND POWER EDITION PER REGION *************************
          Region |        Volume       Int Power |        Int Flux |   Int Fast Flux |Int Epi-Th Flux
              ID |         (cm3)             (W) |       (n-cm/s) |        (n-cm/s) |        (n-cm/s)
                 |                               |                |                 |
           SUB10 |   1.21244e+05    3.42045e-06 |    2.83121e+11 |    2.67004e+11 |    1.61637e-10
           SUB11 |   2.42487e+05    6.47074e-01 |    5.36454e+12 |    5.35663e+12 |    7.51116e-09
           SUB12 |   1.21244e+05    3.42045e-06 |    2.83121e+11 |    2.67004e+11 |    1.61637e-10
           SUB20 |   2.07846e+05    3.72815e-06 |    3.81532e+11 |    3.56939e+11 |    2.45737e-10
           SUB21 |   4.15692e+05    3.52895e-01 |    4.40615e+12 |    4.65819e+12 |    1.09611e-10
           SUB22 |   2.07846e+05    3.72815e-06 |    3.81532e+11 |    3.56939e+11 |    2.45737e-10
            SUB3 |   1.24708e+06    1.65194e-05 |    3.08823e+12 |    2.95533e+12 |    6.28495e-10
-----------------|-----------------------------------|-----------------|-----------------|-----------------
           TOTAL |   2.56343e+06    1.00000e+00 |    1.44682e+13 |    1.42690e+13 |    1.59676e-11
-----------------|-----------------------------------|-----------------|-----------------|-----------------
************************* FLUX AND POWER EDITION PER ASSEMBLY *************************
 3D ASSEMBLY-WISE EDITION
      Assembly|        Region|      Ring|     Index|     Plane|       Volume|    Int Power|
            ID|            ID|         #|         #|         #|        (cm3)|          (W)|
--------------|--------------|----------|----------|----------|-------------|-------------|
       R00000|        SUB10|         1|         1|         1|  1.73205e+04|  5.60843e-37|
       R00001|        SUB11|         1|         1|         2|  3.46410e+04|  1.17812e-01|
       R00002|        SUB12|         1|         1|         3|  1.73205e+04|  5.60843e-37|
       R10000|        SUB10|         2|         1|         1|  5.19615e+04|  1.42981e-36|
       R10001|        SUB11|         2|         1|         2|  1.03923e+05|  2.65831e-01|
       R10002|        SUB12|         2|         1|         3|  5.19615e+04|  1.42981e-36|
       R10100|        SUB10|         2|         2|         1|  5.19615e+04|  1.42980e-36|
       R10101|        SUB11|         2|         2|         2|  1.03923e+05|  2.65831e-01|
       R10102|        SUB12|         2|         2|         3|  5.19615e+04|  1.42980e-36|
       R20000|        SUB20|         3|         1|         1|  5.19615e+04|  8.49582e-37|
       R20001|        SUB21|         3|         1|         2|  1.03923e+05|  7.33570e-02|
       R20002|        SUB22|         3|         1|         3|  5.19615e+04|  8.49582e-37|
       R20100|        SUB20|         3|         2|         1|  5.19615e+04|  1.01456e-36|
       R20101|        SUB21|         3|         2|         2|  1.03923e+05|  1.03890e-01|
       R20102|        SUB22|         3|         2|         3|  5.19615e+04|  1.01456e-36|
       R20200|        SUB20|         3|         3|         1|  5.19615e+04|  8.49581e-37|
       R20201|        SUB21|         3|         3|         2|  1.03923e+05|  7.33570e-02|
       R20202|        SUB22|         3|         3|         3|  5.19615e+04|  8.49581e-37|
       R20300|        SUB20|         3|         4|         1|  5.19615e+04|  1.01456e-36|
       R20301|        SUB21|         3|         4|         2|  1.03923e+05|  1.03890e-01|
       R20302|        SUB22|         3|         4|         3|  5.19615e+04|  1.01456e-36|
       R30000|         SUB3|         4|         1|         1|  2.07846e+05|  1.92715e-36|
       R30100|         SUB3|         4|         2|         1|  2.07846e+05|  3.15628e-36|
       R30200|         SUB3|         4|         3|         1|  2.07846e+05|  3.15628e-36|
       R30300|         SUB3|         4|         4|         1|  2.07846e+05|  1.92715e-36|
       R30400|         SUB3|         4|         5|         1|  2.07846e+05|  3.15628e-36|
```

Figure 3-11. Example of Post-processing for PROTEUS-NODAL: Visualization of flux map (left) and assembly-wise summary table (right).

For enabling the MSR analysis capability within the Workbench interfaces, as illustrated in Figure 3-12, the additional pre-process logic was implemented to translate the workbench input format of flowing fuel model description into the associated PROTEUS-NODAL input format. The DLAYXS file generation process was added to the execution logic to provide delayed neutron precursor parameters to the PROTEUS-NODAL calculation. A sample input #13 demonstrating the MSR modeling was developed within the released tutorial.



```
    }
    msr {
      % insert channel
      channel ( fuel_channel ) {
        channel_type = merge
        inner_ring = 1
        outer_ring = 3
        time_core_out = 100.0 % second
        exclude{
          % assembly=(assembly_name assembly_name )
          sub_assembly=[ sub20 sub21 ]
        }
        flow_profile {
          segment{
            lower_axial_surf=s0
            upper_axial_surf=s1
            velocity = 1.0 %1.0  % m/s unit
          }
          segment{
            lower_axial_surf=s1
            upper_axial_surf=s3
            velocity = 0.5833  % m/s unit
          }
          segment{
            lower_axial_surf=s3
            upper_axial_surf=s4
            velocity = 1.0 %1.0  % m/s unit
          }
        }
      }
      channel ( blk_channel ) {
        channel_type = merge
        inner_ring = 3
        outer_ring = 4
        time_core_out = 100.0 % second
        exclude{
          % assembly=(assembly_name assembly_name )
          sub_assembly=[ sub21 ]
        }
```

Figure 3-12. Example of MSR Calculation within the Workbench.

3.6.2 PROTEUS-MOC [12]

The PROTEUS-MOC code is a neutron transport solver based on the 3D method of characteristics (MOC) for 2D unstructured finite element meshes with axial extrusion. It allows modeling most of complex or unconventional geometry reactor problems. To provide high-fidelity level in an efficient manner, the PROTEUS-MOC code employs a unique 3D formulation which combines the two-dimensional (2D) MOC radially and the discontinuous Galerkin finite element method axially. The PROTEUS-MOC code has capabilities to solve steady-state and transient problems. In FY-2019, the steady-state and transient capabilities were connected to the PyARC module to improve the usability of PROTEUS-MOC by leveraging the user-friendly interface provided by Workbench. This extension can be used for PROTEUS-SN with minor updates as well.

The PyARC common input logic for geometry creation does not support an unstructured finite element mesh generation in a format that is compatible with PROTEUS-MOC. Consequently, the PROTEUS-MOC integration does not currently use the PyARC geometry description logic and instead relies on pre-generated off-line mesh and associated cross section generations.

The workflow for PROTEUS-MOC calculation was built upon the existing PyARC module and its sub-module for PROTEUS-NODAL. The implemented workflow is illustrated in Figure 3-13. The 2D mesh file can be generated by making use of mesh generation tools such as CUBIT [27]. Alternatively, the ANL mesh toolkit [28] can be used for generating typical reactor lattices geometries. The associated multi-group cross section data can be prepared by using the cross-section generation codes such as MC_2-3 or Monte Carlo codes (SERPENT and OpenMC). After checking consistency of these externally pre-generated input files, the PyPROTEUS modules can return the following input files for PROTEUS-MOC execution. A user should complete the assignment input file and update the driver input file for the problem of interest.

- *Assignment Input File*: defines compositions and assigns them to the 3D geometrical regions by extruding the regions defined in the 2D mesh file.
- *Driver Input File*: defines the simulation parameters such as power level, angular discretization, convergence criteria, parallelization, iteration limits, etc.

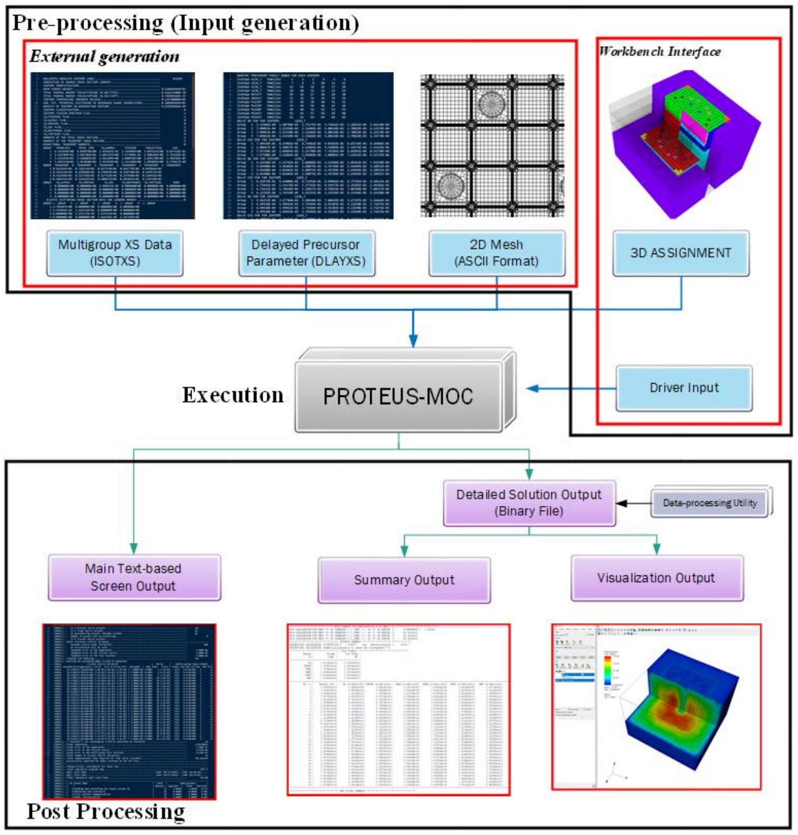


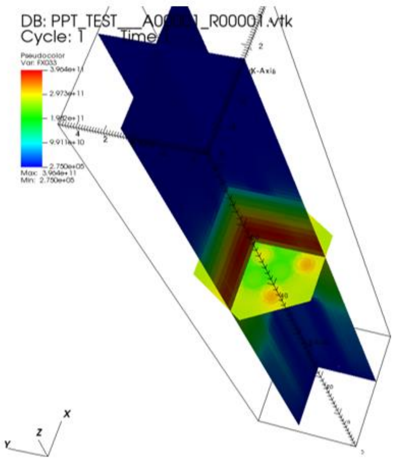
Figure 3-13. PROTEUS-MOC Workflow implemented in PyARC.

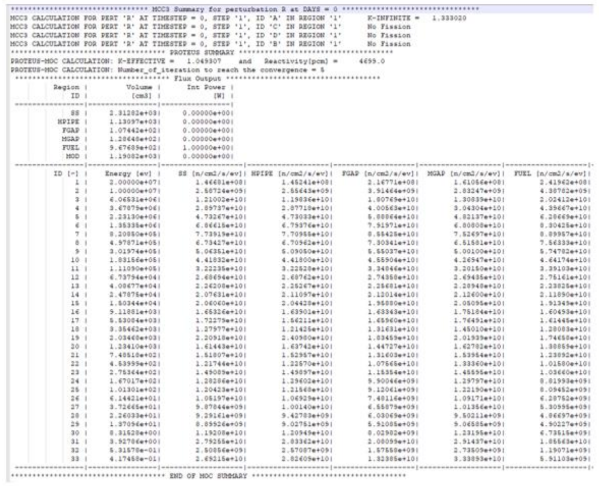
For the transient calculation mode, the additional input keywords are available in the Workbench interface to define perturbations of materials and temperatures for 3D geometrical regions of interest as a function of time. Based upon these user-defined transient descriptions, the pre-process logic can prepare the time-dependent assignment files required for the execution of PROTEUS-MOC.

PROTEUS-MOC is executed via the runtime environment of PyARC. Once the calculation is completed, the PROTEUS-MOC code produces two basic types of outputs as:

- *Main Text-based Screen Output:* contains confirmation that the inputs are imported successfully, computing timing summaries, and eigenvalue iteration history results.
- *Detailed Solution Output:* contains mesh-wise solution in the entire 3D domain which is exported to an organized binary file format for detailed analysis.

Upon execution of PROTEUS-MOC, it is recommended to use HPC clusters via the remote execution feature of Workbench due to the computational resource demands of detailed 3D transport calculations. The PROTEUS-MOC code provides the external data-processing utility to extract data of interest and to visualize detailed 3D solutions by processing the detailed solution output file. In the workflow, the data-processing utility is additionally executed for the subsequent post-processing. The post-processing of the PROTEUS-MOC outputs was implemented by printing the main information of interest to a user in the summary file ("*.summary*"). When opening this "*.summary*" file with the Workbench, the user can use the "*flux spectrum*" processing to automatically plot the neutron flux spectrum for each region. The direct visualization of the primary variables is enabled by opening the generated "*.vtk*" file with VisIt through the Workbench. The implemented post-processing capabilities are illustrated in Figure 3-14.





```
********************************* MCC3 Summary for perturbation R at DAYS = 0 *********************************
MCC3 CALCULATION FOR PERT 'R' AT TIMESTEP = 0, STEP '1', ID 'A' IN REGION '1'      K-INFINITE =   1.333020
MCC3 CALCULATION FOR PERT 'R' AT TIMESTEP = 0, STEP '1', ID 'C' IN REGION '1'      No Fission
MCC3 CALCULATION FOR PERT 'R' AT TIMESTEP = 0, STEP '1', ID 'D' IN REGION '1'      No Fission
MCC3 CALCULATION FOR PERT 'R' AT TIMESTEP = 0, STEP '1', ID 'B' IN REGION '1'      No Fission
*********************************** PROTEUS SUMMARY ***********************************
PROTEUS-MOC CALCULATION: K-EFFECTIVE =   1.049307    and   Reactivity(pcm) =    4699.0
PROTEUS-MOC CALCULATION: Number_of_iteration to reach the convergence = 5
 *********************************** Flux Output ***********************************
          Region |          Volume |       Int Power |
              ID |           [cm3] |             [W] |
 ----------------|-----------------|-----------------|
              SS |     2.31282e+03|     0.00000e+00|
           HPIPE |     1.13097e+03|     0.00000e+00|
            FGAP |     1.07442e+02|     0.00000e+00|
            MGAP |     1.28648e+02|     0.00000e+00|
            FUEL |     9.67689e+02|     1.00000e+00|
             MOD |     1.19082e+03|     0.00000e+00|
 ----------------|-----------------|-------------------|-------------------|-------------------|-------------------|-------------------|-
          ID [-] |     Energy [ev] |     SS [n/cm2/s/ev]| HPIPE [n/cm2/s/ev]|  FGAP [n/cm2/s/ev]|  MGAP [n/cm2/s/ev]|  FUEL [n/cm2/s/ev]|
               1 |     2.00000e+07|        1.46681e+08|        1.45241e+08|        2.16771e+08|        1.61056e+08|        2.41962e+08|
               2 |     1.00000e+07|        2.58724e+09|        2.55643e+09|        3.91464e+09|        2.83247e+09|        4.38782e+09|
               3 |     6.06531e+06|        1.21002e+10|        1.19836e+10|        1.80769e+10|        1.30839e+10|        2.02412e+10|
               4 |     3.67879e+06|        2.89737e+10|        2.87718e+10|        4.00563e+10|        3.04304e+10|        4.39667e+10|
               5 |     2.23130e+06|        4.73267e+10|        4.73033e+10|        5.88864e+10|        4.82137e+10|        6.28669e+10|
               6 |     1.35335e+06|        6.86615e+10|        6.79376e+10|        7.91971e+10|        6.80808e+10|        8.30425e+10|
               7 |     8.20850e+05|        7.73919e+10|        7.70955e+10|        8.55425e+10|        7.52697e+10|        8.89957e+10|
               8 |     4.97871e+05|        6.73427e+10|        6.70962e+10|        7.30341e+10|        6.51581e+10|        7.56333e+10|
               9 |     3.01974e+05|        5.06351e+10|        5.09050e+10|        5.55037e+10|        5.00100e+10|        5.74782e+10|
              10 |     1.83156e+05|        4.41832e+10|        4.41800e+10|        4.55904e+10|        4.26947e+10|        4.64174e+10|
              11 |     1.11090e+05|        3.22235e+10|        3.22528e+10|        3.34846e+10|        3.20150e+10|        3.39103e+10|
              12 |     6.73794e+04|        2.68694e+10|        2.68762e+10|        2.74358e+10|        2.69435e+10|        2.75161e+10|
              13 |     4.08677e+04|        2.26208e+10|        2.26267e+10|        2.25681e+10|        2.20948e+10|        2.23828e+10|
              14 |     2.47875e+04|        2.07631e+10|        2.11097e+10|        2.12014e+10|        2.12600e+10|        2.11890e+10|
              15 |     1.50344e+04|        2.06060e+10|        2.04428e+10|        1.95880e+10|        2.05095e+10|        1.91349e+10|
              16 |     9.11881e+03|        1.65326e+10|        1.63901e+10|        1.63343e+10|        1.75184e+10|        1.60493e+10|
              17 |     5.53084e+03|        1.72275e+10|        1.56211e+10|        1.65960e+10|        1.76491e+10|        1.61445e+10|
              18 |     3.35462e+03|        1.27977e+10|        1.21425e+10|        1.31631e+10|        1.45010e+10|        1.28083e+10|
              19 |     2.03468e+03|        2.20918e+10|        2.40980e+10|        1.83459e+10|        2.01939e+10|        1.74658e+10|
              20 |     1.23410e+03|        1.61443e+10|        1.63742e+10|        1.44727e+10|        1.62782e+10|        1.38869e+10|
              21 |     7.48518e+02|        1.51807e+10|        1.52957e+10|        1.31603e+10|        1.53954e+10|        1.23092e+10|
              22 |     4.53999e+02|        1.21744e+10|        1.22670e+10|        1.07565e+10|        1.33360e+10|        1.01580e+10|
              23 |     2.75364e+02|        1.49085e+10|        1.49897e+10|        1.15354e+10|        1.45595e+10|        1.03660e+10|
              24 |     1.67017e+02|        1.28286e+10|        1.29602e+10|        9.90046e+09|        1.29797e+10|        8.81993e+09|
              25 |     1.01301e+02|        1.20423e+10|        1.21568e+10|        9.12061e+09|        1.22190e+10|        8.09452e+09|
              26 |     6.14421e+01|        1.05197e+10|        1.06929e+10|        7.48116e+09|        1.09171e+10|        6.28752e+09|
              27 |     3.72665e+01|        9.87844e+09|        1.00140e+10|        6.55879e+09|        1.01356e+10|        5.30995e+09|
              28 |     2.26033e+01|        9.29161e+09|        9.42783e+09|        6.03069e+09|        9.50211e+09|        4.86697e+09|
              29 |     1.37096e+01|        8.89926e+09|        9.02751e+09|        5.91085e+09|        9.06585e+09|        4.90227e+09|
              30 |     8.31528e+00|        1.19208e+10|        1.20949e+10|        8.02982e+09|        1.23195e+10|        6.73615e+09|
              31 |     3.92786e+00|        2.79255e+10|        2.83362e+10|        2.08099e+10|        2.91437e+10|        1.85563e+10|
              32 |     5.31578e-01|        2.50856e+09|        2.57087e+09|        1.57558e+09|        2.73509e+09|        1.19071e+09|
              33 |     4.17458e-01|        2.69215e+10|        2.82609e+10|        1.32385e+10|        3.33893e+10|        5.91103e+09|
 ----------------|-----------------|-------------------|-------------------|-------------------|-------------------|-------------------|-
*********************************** END OF MOC SUMMARY ***********************************
```

Figure 3-14. Example of Post-processing for PROTEUS-MOC: Visualization of flux map (left) and region-wise summary table (right).

4 Verification & Application

In FY-2019, verification of the ARC integration in the NEAMS Workbench was performed by ANL and North Carolina State University (NCSU) through its application to solve international OECD/NEA benchmarks. The OECD/NEA sub-group on Uncertainty Analysis in Modelling (UAM) for Design, Operation and Safety Analysis of Sodium-cooled Fast Reactors (SFR-UAM) has been formed under the NSC/WPRS/EGUAM and specified a series of benchmarks [29]. For demonstration purposes of the implemented capabilities available in the Workbench, the fuel assembly depletion benchmark and the full SFR core benchmarks proposed within the SFR UAM [30, 31] were modeled using the ARC codes through the Workbench in FY-2018 [4]. In FY19, participation focused on full core benchmark modeling, unit cell benchmark analyses, and on an original analysis of the manufacturing uncertainties summarized in Appendix C [19]. Results obtained with the Workbench/PyARC were presented at the benchmark annual review in ORNL on May 13-17, 2019.

4.1 Full Core Benchmarks

4.1.1 ABR-1000

The ABR-1000 full core benchmark problem was modelled using ARC codes within the NEAMS Workbench, as shown in Figure 4-1. The MC$_2$-3 code is used to generate region-homogenized 33-group cross sections using the ENDF/B-VII.0 library [32], with 1-D heterogeneous treatment and using TWODANT to solve for region-wise flux. The core physics calculation is performed using DIF3D using the transport option VARIANT with 3rd order angular flux and 3rd order scattering approximation.

Table 4-1 presents the core simulation results of the ABR-1000 core at reference state. Quantities of interest includes core k_{eff}, effective delayed neutron fraction β_{eff} and various reactivity feedback coefficients calculated with PERSENT. Density feedback coefficients of sodium ($\Delta\rho_{\text{cool}}$), structural material ($\Delta\rho_{\text{str}}$), and fuel ($\Delta\rho_{\text{fuel}}$) are computed by perturbing the density of the corresponding material in the fuel assemblies by 1%. The Doppler feedback effect is represented with Doppler constant (K_{Dop}), which is calculated by doubling the nominal average fuel temperature. The sodium void worth is computed by decreasing the density of sodium in fuel region into 1%.

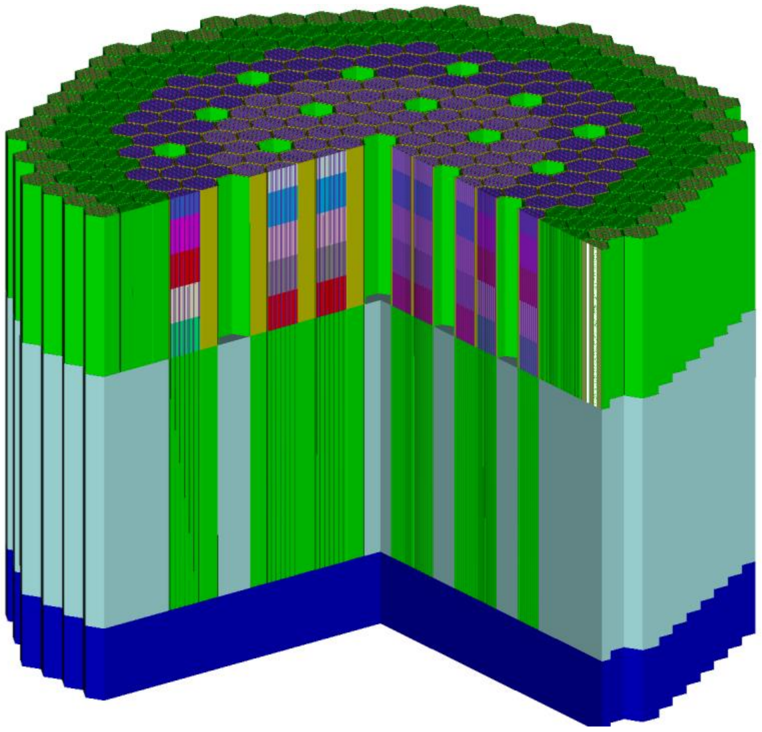


Figure 4-1. Radial layout of the ABR-1000 core modeled with Workbench/PyARC.

Table 4-1. Various output responses in ABR-1000 core simulations at reference state.

Output response	ABR-1000 Core
k_{eff}	1.01659
β_{eff}	329.7 pcm
K_D	-390.3 pcm
Na Void Worth	1566.3 pcm
1% Sodium	15.8 pcm
1% Duct	17.0 pcm
1% Cladding	30.1 pcm
1% Fuel	-401.9 pcm

The impact of nuclear data uncertainty on k_{eff} and various reactivity coefficients are analyzed with the adjoint-based perturbation approach. The sensitivity coefficient of output responses are calculated with Eq. (1) using PERSENT. As shown in Figure 4-2, for the ABR-1000 reactor core, the top 10 nuclide-reaction pairs those are most sensitive to k_{eff} are calculated with respect to different energy group, while the k_{eff} is found to be most sensitive to Pu239 fission spectrum. The uncertainties are computed based on an updated COMMARA-2.0 covariance matrix [33]. It is also found that uncertainty of k_{eff} is 1.29%, and a relatively large uncertainty of 11.40% is observed for sodium void worth. Similar analysis can be applied to different output responses, as summarized in Appendix B. The U238 inelastic, Na23 inelastic, and Fe56 elastic cross sections are found to be the most influential nuclide-reaction pairs to output responses.

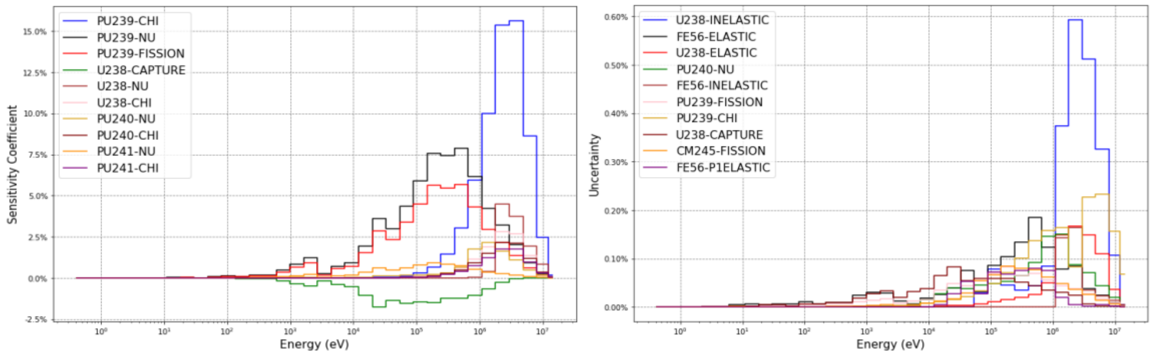


Figure 4-2. Top 10 sensitive nuclide-reaction pairs to ABR-1000 core k_{eff}, top 10 contributors to ABR-1000 core k_{eff} uncertainty.

4.1.2 MOX-3600

The MOX-3600 full core benchmark problem was modelled using ARC codes within the NEAMS Workbench, using same approach as above described, as shown in Figure 4-3. Table 4-2 presents the core simulation results of the MOX-3600 core at reference state. The top 5 nuclide-reaction pairs that contribute to uncertainties of MOX3600 core k_{eff} and various feedback coefficients are listed in Appendix B. U238 inelastic cross section is the top contributor to uncertainty of most of output responses, except for the sodium void worth and sodium density feedback coefficients, where Na23 becomes dominant.

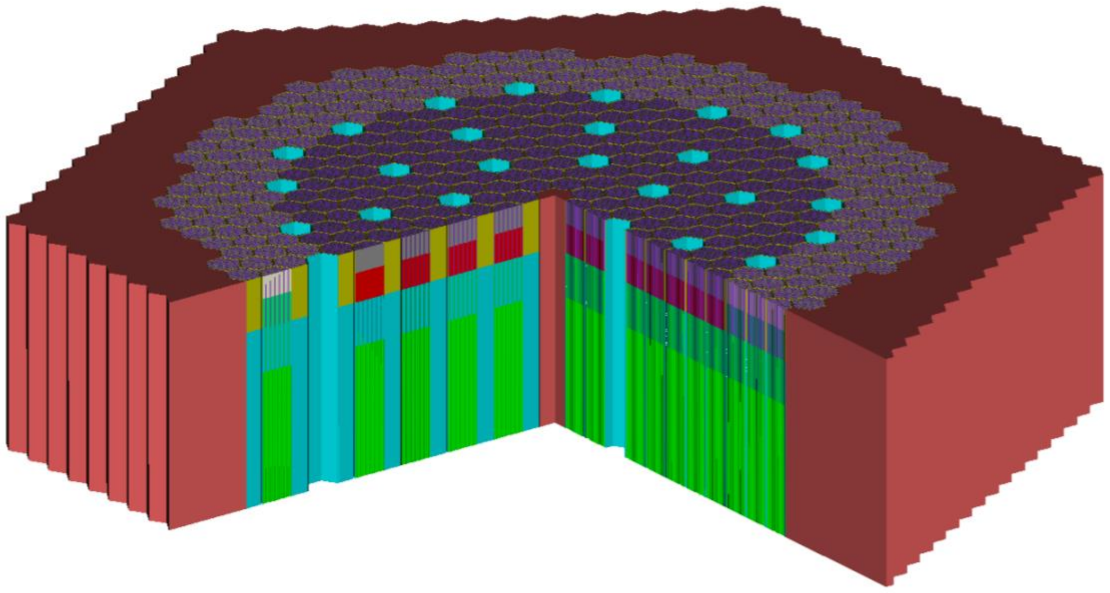


Figure 4-3. Radial layout of the MOX3600 core modeled with Workbench/PyARC.

Table 4-2. Various output responses in MOX3600 core simulations at reference state.

Output response	MOX3600 Core
k_{eff}	1.014288
β_{eff}	349.9 pcm
K_D	-847.4 pcm
Na Void Worth	1726.7 pcm
1% Sodium	16.1 pcm
1% Duct	20.6 pcm
1% Cladding	29.5 pcm
1% Fuel	-239.6 pcm

4.1.3 ASTRID

In FY19, the French CEA (Atomic Energy Commission) released specifications for a new proposed full-core benchmark based on the ASTRID core [34]. This is a medium 1500 MWth core that exhibits near zero sodium void worth at the end of equilibrium cycle core state, which bring improved natural core behavior during unprotected loss of flow transients to this oxide-fueled core. Preliminary PyARC model of the ASTRID core was completed in FY19.

As shown in Figure 4-4, the core of ASTRID is composed of two main fuel regions, divided into 177 inner fuel assemblies and 114 outer fuel assemblies, as well as twelve control rods, six safety rods and the radial reflector and shielding regions. The inner fuel assemblies are made of an alternating succession of two fertile sub-assemblies and two fissile sub-assemblies. The outer assemblies are composed of just one fissile region on top of one fertile region. In this PyARC model, the sodium plenum, reflectors and shields are modeled as homogeneous materials.

The cross sections are generated with the reference two-step MC_2-3 procedure, using the automatically-generated TWODANT RZ geometry shown in Figure 4-5, to condense the multi-group cross-sections into 33 groups. Equivalent 1D-geometries of the fuel assemblies and the control rods were automatically generated, as displayed in Figure 4-6, to perform a heterogeneous calculation with MC_2-3 to account for spatial self-shielding. The different reactivity feedback coefficients are calculated with DIF3D and PERSENT using a third core symmetry model and preliminary results are displayed in Table 4-3.

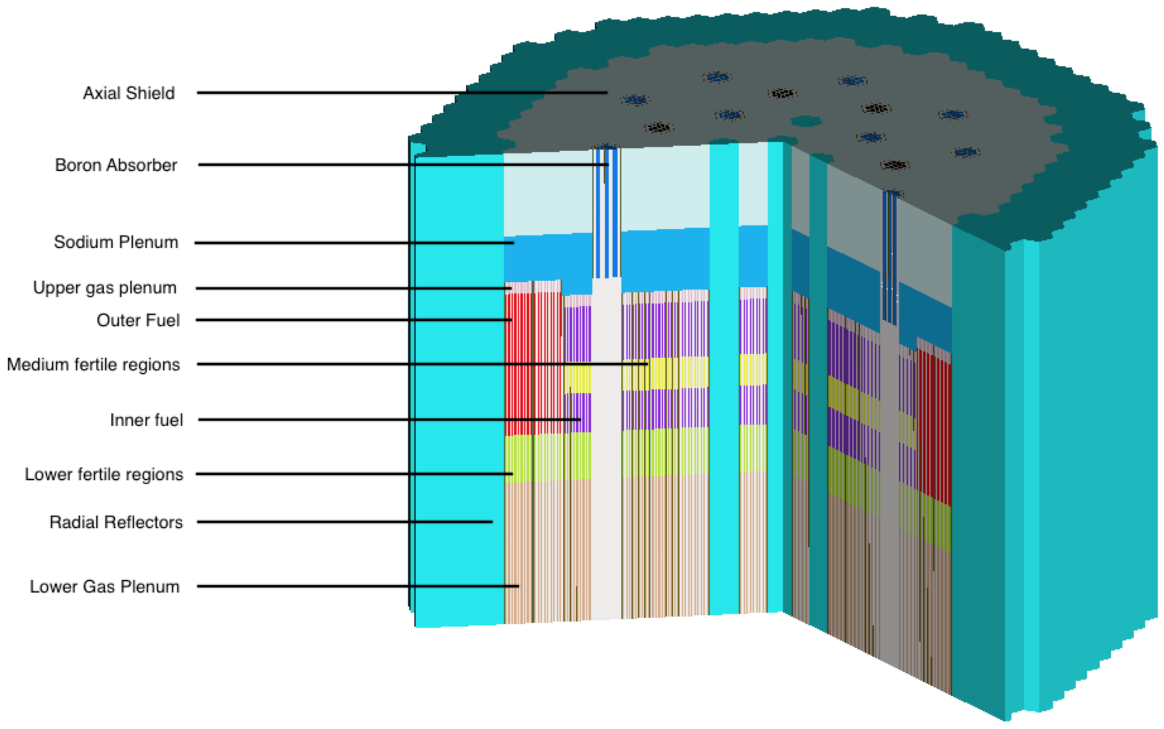


Figure 4-4. ASTRID core 3D layout modeled with Workbench/PyARC.

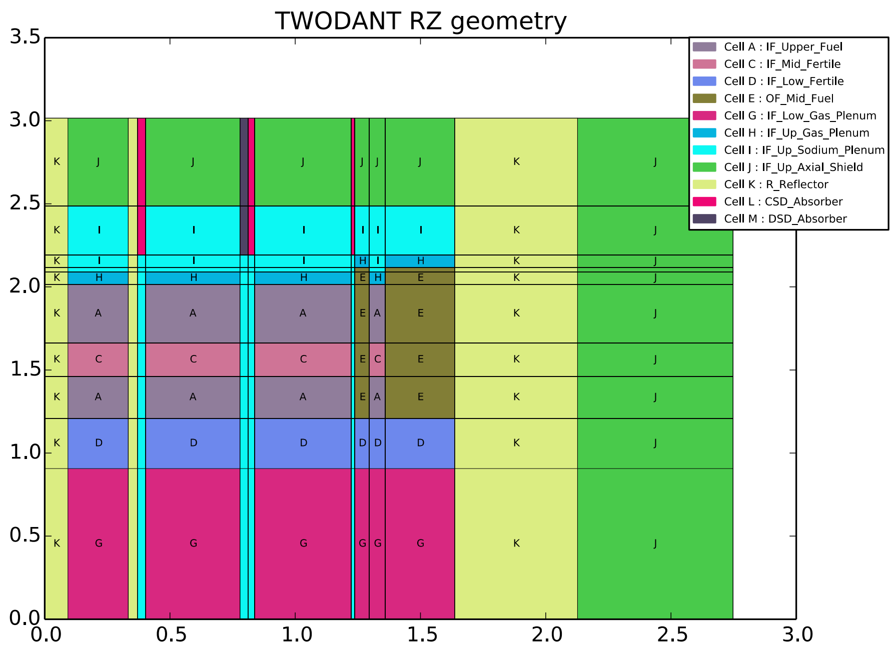


Figure 4-5. ASTRID core RZ layout computed with Workbench/PyARC.

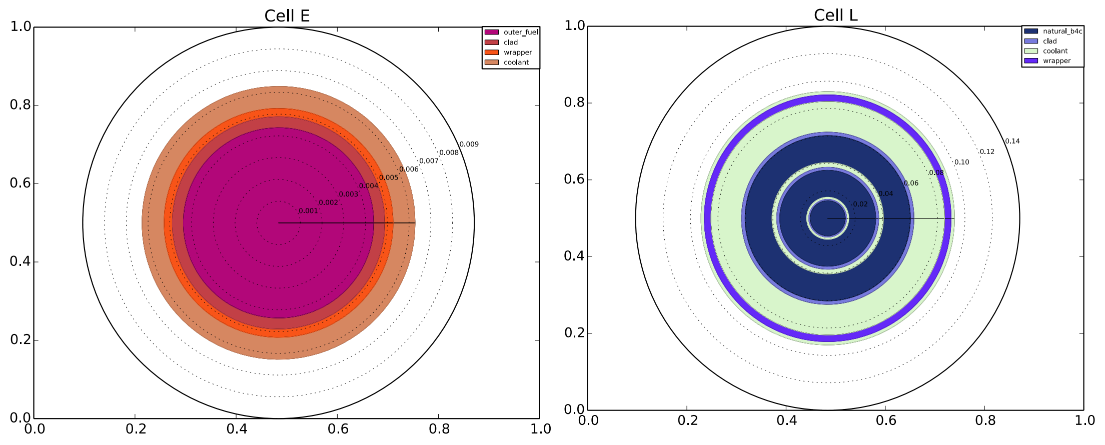


Figure 4-6. ASTRID fuel (left) and control assemblies (right) 1D layout computed with Workbench/PyARC.

Table 4-3. Various output responses in ASTRID core simulations at reference state.

K-effective, nominal	1.00267
1% Sodium, pcm/K	0.043
1% Wrapper, pcm/K	0.017
1% Cladding, pcm/K	0.033
1% Fuel, pcm/K	-0.38
1% Fuel + Axial, pcm/K	-0.14
1% Grid, pcm/K	-0.89
CSD+DSD worth (fully inserted)	-6075
CSD+DSD worth (5cm from top)	-107
Delayed Neutron Fraction	346
Fissile K_D Doppler, pcm	-626
Fertile K_D Doppler, pcm	-314
Na Void Worth, pcm	-368

4.2 Unit Cell Benchmarks

4.2.1 Pin Cell Benchmarks

The pin cell models defined in the UAM-SFR benchmark are represented by a circular fuel rod with cladding centered in a hexagon filled with sodium coolant in the two-dimensional (2D) geometry, as shown in Figure 4-7. The pin cells are modeled using ARC code suite with 3 cylindrical rings using MC_2-3 based on area equivalence. The void regions (i.e., central hole and clad-fuel gap) in the MOX3600 pin cell are homogenized with fuel region based on mass conservation. Condensed region-wise self-shielding cross sections in 1041 energy group is calculated using the 2082 group master library with 1-D heterogeneity treatment, and then further condensed into 33 energy groups using MC_2-3.

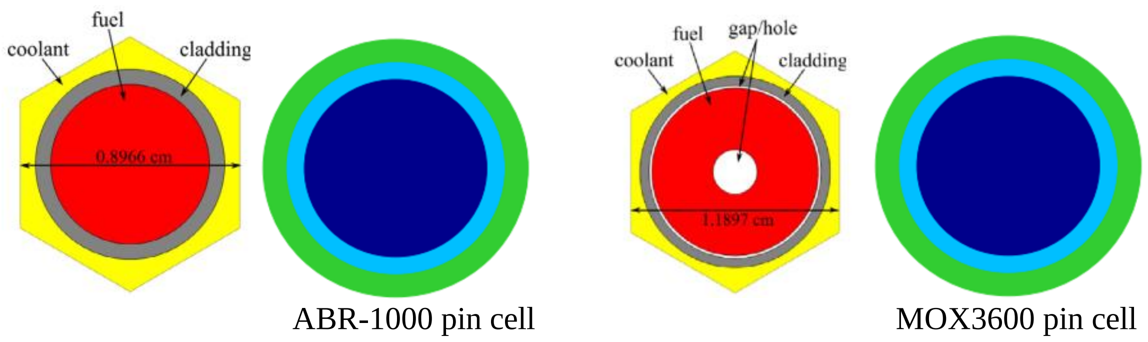
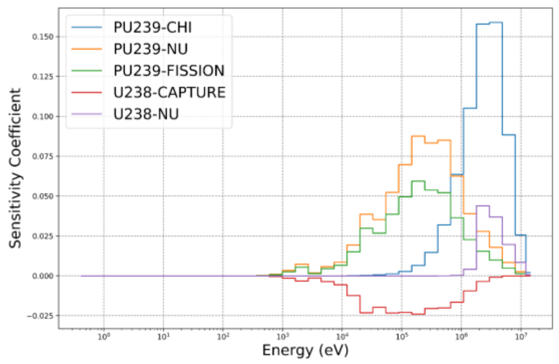


Figure 4-7. Pin cell model and associated equivalent model using MC2-3.

The microscopic cross sections are further condensed into one group cross section using the calculated neutron spectrum. Perturbation theory is employed to calculate the sensitivity coefficients of microscopic cross sections in 33 groups and the uncertainty are computed based on COMMARA-2.0 covariance matrix.

Table 4-4. Uncertainty of pin cell model introduced from nuclear data.

Output	ABR-1000		MOX3600	
	value	rel. std.	value	rel. std.
k_{eff}	1.36096	1.54%	1.19439	1.66%
coolant_mic_el_23Na	4.23958E+00	5.16%	4.83620E+00	4.56%
cladding_mic_el_56Fe	3.65566E+00	8.23%	4.14555E+00	5.21%
fuel_mic_inel_238U	1.15632E+00	5.63%	9.51650E-01	5.03%
fuel_mic_fis_238U	3.35464E-02	7.56%	4.35516E-02	7.33%
fuel_mic_fis_239Pu	1.56852E+00	0.51%	1.76940E+00	0.56%
fuel_mic_fis_240Pu	3.62626E-01	5.17%	3.56222E-01	5.21%
fuel_mic_fis_241Pu	2.11656E+00	1.13%	2.48123E+00	1.57%
fuel_mic_fis_242Pu	2.42651E-01	5.62%	2.52016E-01	5.56%
fuel_mic_n_gam_238U	2.01562E-01	1.83%	2.72662E-01	1.73%
fuel_mic_n_gam_239Pu	2.81618E-01	6.67%	4.72345E-01	5.15%
fuel_mic_n_gam_240Pu	3.72531E-01	4.22%	4.82651E-01	3.66%
fuel_mic_n_gam_241Pu	3.02565E-01	20.26%	4.45446E-01	13.46%
fuel_mic_n_gam_242Pu	2.86651E-01	5.55%	4.31566E-01	4.56%
fuel_mac_fis	5.46560E-03	1.56%	5.35400E-03	1.53%
fuel_mac_abs	1.16256E-02	0.96%	1.25663E-02	0.75%



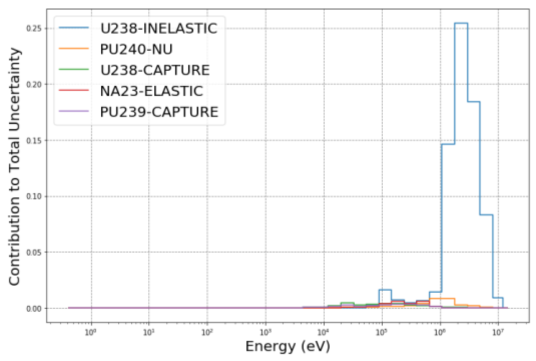


Figure 4-8. Top 5 sensitive nuclide-reaction pairs to ABR-1000 pin cell k_{inf} (left), Top 5 uncertainty contributors to the ABR-1000 pin cell k_{inf} (right).

The infinite multiplication factors of pin cell models are also computed. Figure 4-8 and Table 4-5 show the top 5 largest uncertainty contributors to pin cell k_{inf}. The uncertainties of k_{inf} in both cases are found to be larger than 1.50%, and U238 inelastic cross section is found to be the largest uncertainty contributor. The rest of uncertainties contributors from two cases differ from each other, basically because these two models have different fuel compositions and enrichments. For example, O16 elastic cross section contributes 0.23% to k_{inf} uncertainty in MOX3600 pin cell case, while it is not one of top uncertainty contributors in ABR-1000 pin cell case.

Table 4-5. Top 5 contributors to uncertainty of pin cell k_{inf}.

Output	ABR-1000		MOX3600	
Nominal k_{inf}	1.36096		1.19439	
Total rel. std.	1.53%		1.66%	
Top 5 contributors to k_{inf} uncertainty	U238-inel.	1.30%	U238-inel.	1.52%
	Pu240-ν	0.28%	U238-cap.	0.32%
	U238-cap.	0.26%	Pu239-cap.	0.24%
	Na23-el.	0.23%	O16-el.	0.23%
	Pu239-cap.	0.22%	Pu239-fis.	0.21%

4.2.2 Assembly Cell Benchmarks

The two-dimensional fuel assembly models have been developed using the Workbench/PyARC. As shown in Figure 4-9, the fuel assembly consists in 271 fuel pin cells enclosed by a hexagonal duct. Reflective boundary condition is employed for modeling these two fuel assembly models using DIF3D, the cross sections are generated using MC_2-3 with 1-D heterogeneity treatment (with detailed description of each ring of pins).

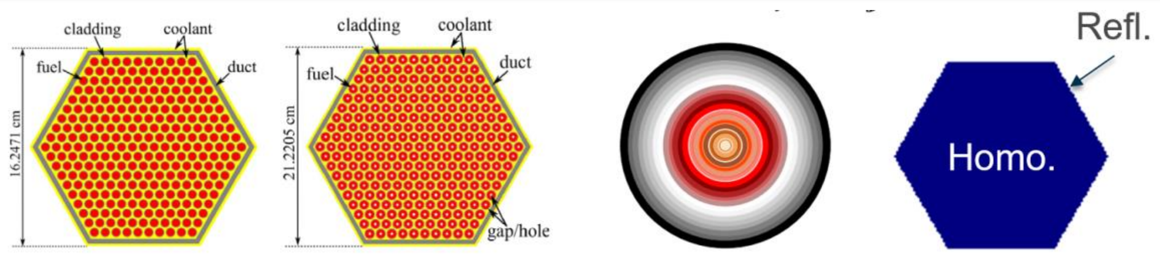


Figure 4-9. Fuel assembly models and equivalent assembly model.

The output responses required for the SFR-UAM sub-exercises specification includes k_{inf}, Doppler constant and sodium void worth, as well as their associated uncertainties. The Doppler constant is computed by doubling the fuel temperature, and sodium void worth is computed by reducing the density of sodium in the fuel region into 1%. The uncertainties of assembly k_{inf} and two reactivity coefficients, i.e., Doppler constant and sodium void worth, are summarized in Table 4-6.

It is found that the uncertainty of k_{inf} of ABR-1000 assembly is smaller compared with MOX3600 fuel assembly, which is consistent to the pin cell case. The top 5 nuclide-reaction contributors to the output uncertainties are presented in Table 4-7. For both fuel assembly models, U238 inelastic cross section is found to be the largest uncertainty contributor to k_{inf} and Doppler constant, while Na23 elastic cross section is the top 1 contributor to uncertainty of sodium void worth. The rest of main contributors differs between the two fuel assembly models. It is found that Fe56-elastic and Na23-elastic cross sections are top contributors in ABR-1000 core uncertainties, while O16 elastic cross section is one of the top 5 contributor to uncertainty of MOX3600 fuel assembly.

Table 4-6. Output responses with associated uncertainties in different assembly cases.

Case	ABR-1000		MOX3600	
Output	value	rel. std.	value	rel. std.
k_{inf}	1.28132	1.403%	1.14869	1.625%
Doppler Constant, pcm	-319	4.356%	-755	6.723%
Sodium Void Worth, pcm	5832	6.081%	2910	5.566%

Table 4-7. Top 5 uncertainty contributors of various output responses in assembly model.

k_{inf}				Doppler Constant				Sodium void worth			
ABR-1000		MOX3600		ABR-1000		MOX3600		ABR-1000		MOX3600	
input	rel. std.	input	rel. std.	input	rel. std.	input	rel. std.	input	rel. std.	input	rel. std.
inel._238U	1.13%	inel._238U	1.42%	inel._238U	4.05%	inel._238U	4.92%	el._23Na	3.58%	el._23Na	3.62%
cap._238U	0.27%	cap._238U	0.33%	el._23Na	2.89%	fis._238U	2.96%	inel._23Na	1.78%	inel._23Na	2.95%
el._23Na	0.27%	cap._239Pu	0.23%	el._56Fe	2.17%	inel._241Pu	1.86%	inel._238U	1.06%	inel._238U	1.46%
cap._239Pu	0.26%	fis._239Pu	0.22%	cap._240Pu	0.75%	inel._239Pu	1.56%	cap._238U	0.85%	cap._238U	1.33%
el._56Fe	0.25%	el._16O	0.21%	inel._240Pu	0.50%	el._16O	1.35%	el._56Fe	0.61%	fis._239Pu	0.79%

5 Conclusions and Future Work

This report details the efforts under way for integrating the Argonne Reactor Computation (ARC) and NEAMS codes into the Workbench. These codes contain both legacy codes like DIF3D and REBUS-3 that were developed with over 30 years of experience, and newer NEAMS additions like MC_2-3, PERSENT, and PROTEUS. These codes are extremely attractive by their wide capabilities and computational efficiency.

Integrating the ARC codes into the Workbench benefits directly the advanced reactor community (within the DOE national laboratories, universities and companies) by:

- Providing a set of controlled, maintained and validated scripts to generate ARC inputs, which promotes best practices, reduces the learning curve, and facilitates project collaboration.
- Improving the user experience with the ARC codes: the Workbench interface provides assistance for building an input through auto-completion, real-time validation, document navigation, and geometry and results visualization.
- Enabling new modeling capabilities for advanced reactor design and analyses. The PyARC module facilitates and automatizes complex calculations and workflows for reactor analysis enabling geometrical perturbations, cross-section update through depletion, etc. The Dakota/PyARC coupling in the Workbench was also demonstrated to enable mathematical optimization and sensitivity analysis/uncertainty quantification (SA/UQ) techniques with ARC neutronic simulations.
- Helping user transition to high-fidelity NEAMS codes. This was demonstrated with PROTEUS integration in FY-2019 within the same input logic.

In FY-2019, effort focused on integrating the GAMSOR code for gamma heating calculations, and the PROTEUS-NODAL code for 3D Transport calculation. Initial work was performed on the integration of the workflow for PROTEUS-MOC calculations. Various more minor improvements were completed to enable additional modeling options and to respond to user requests.

For demonstration purposes, the ARC codes were used through the Workbench for solving the SFR-UAM benchmark problems. They are currently used at ANL, Westinghouse, INL, Oklo, and NCSU through the Workbench by nuclear engineers for LFR, MSR, micro-reactor, and SFR core design analyses [21, 35, 36, 37, 38]. The following in-persons training were organized in FY-2019 and several more remote follow-up trainings and assistance sessions were held:

- ANS Winter 2018 panel – November 11, 2018
- Training at Westinghouse – May 29-30, 2019
- Training at ANL (~10 participants) – July 3, 2019

Future efforts will focus on continuously adding new and existing modeling capabilities available with the ARC and NEAMS codes, training new users and supporting them to continue building user experience.

REFERENCES

[1] B. T. Rearden, R. A. Lefebvre, "Objectives of the NEAMS Workbench," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[2] B. T. Rearden, R. A. Lefebvre, A. B. Thompson, B. R. Langley, N.E. Stauff, "Introduction to the Nuclear Energy Advanced Modeling and Simulation Workbench," M&C 2017, Jiju Island, South Korea, April (2017).

[3] N. Stauff, N. Gaughan, and T. Kim, "ARC integration into the NEAMS Workbench," ANL/NE-17/31, September 30, 2017.

[4] N. Stauff, "Updated status of the ART neutronic fast reactor tools integration to the NEAMS Workbench," ANL/NEAMS-18/1, September 30, (2018).

[5] ARC 11.0: Code System for Analysis of Nuclear Reactors, Argonne National Laboratory (2014). Available from Available from Radiation Safety Information Computational Center as CCC-824.

[6] Changho Lee, Yeon Sang Jung, and Won Sik Yang, "MC$_2$-3: Multigroup Cross Section Generation Code for Fast Reactor Analysis," ANL/NE-11-41 Rev.3, August 31 (2018).

[7] K. L. Derstine, "DIF3D: A Code to Solve One-, Two-, and Three-Dimensional Finite Difference Diffusion Theory Problems," ANL-82-64, Argonne National Laboratory (1984).

[8] B. J. Toppel, "A User's Guide to the REBUS-3 Fuel Cycle Analysis Capability," ANL-83-2, Argonne National Laboratory (1983).

[9] M. A. Smith, W. S. Yang, A. Mohamed, E. E. Lewis, "Perturbation and Sensitivity Tool Based on the VARIANT Option of DIF3D," ANS Transactions 107, San Diego, Nov. 11-15 (2012).

[10] M. A. Smith, C. H. Lee, and R. N. Hill, "GAMSOR: Gamma Source Preparation and DIF3D Flux Solution," ANL/NE-16/50 Rev. 1.0, June 28, 2017.

[11] Y. S. Jung, C. H. Lee, M. A. Smith, "PROTEUS-NODAL User Manual (Rev.0)," ANL/NE-18/4, Argonne National Laboratory, September 30 (2018).

[12] Y. S. Jung, C. H. Lee, M. A. Smith, "PROTEUS-MOC User Manual (Rev.0)," ANL/NE-18/10, Argonne National Laboratory, September 30 (2018).

[13] Nicolas E. Stauff, Taek K. Kim, Robert A. Lefebvre, Brandon R. Langley, Bradley T. Rearden, "Integration of the Argonne Reactor Computation codes into the NEAMS Workbench," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[14] R. A. Lefebvre, A. B. Thompson, B. R. Langley, B. T. Rearden, "NEAMS Workbench 1.0 Beta Status," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[15] Robert A. LEFEBVRE, Brandon R. LANGLEY, and Jordan P. LEFEBVRE, "Workbench Analysis Sequence Processor", ORNL/TM-2017/619, UT-Battelle, LLC, Oak Ridge National Laboratory (2017).

[16] LLNL: VisIT Visualization Tool (2002– 2016). https://wci.llnl.gov/codes/visit

[17] Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.7 User's Manual.

[18] Nicolas E. Stauff, Robert A. Lefebvre, Laura Swiler, Bradley T. Rearden, "Coupling of DAKOTA with the ARC suite of codes in the NEAMS Workbench for Uncertainty Quantification," ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[19] Kaiyue Zeng, Nicolas E. Stauff, Jason Hou, "Sensitivity and Uncertainty Analysis of the Advanced Burner Reactor Core Using NEAMS Workbench," Submitted to ANS Winter Meeting (2019).

[20] K. Zeng, Nicolas Stauff, "Multi-criteria optimization of the Advanced Burner Test Reactor," – Submitted to Progress in Nuclear Energy, (2019).

[21] T. K. Kim, N. Stauff, C. Stansbury, A. Levinsky, F. Franceschini, "Long Core Life Options for the Westinghouse LFR," proceedings of Global 2019, Seattle, WA, Sept (2019).

[22] C. H. Lee, N. E. Stauff, "Improved Reactivity Worth Estimation of MC2-3/DIF3D in Fast Reactor Analysis," Proceedings of ANS Sumer Meeting, paper 14201, San Antonio, Texas (2015).

[23] R. E. Alcouffe, F. W. Brinkley, D. R. Marr, and R. D. O'Dell, "User's Guide for TWODANT: A Code Package for Two-Dimensional, Diffusion-Accelerated, Neutral-Particle Transport," LA-10049-M, Los Alamos National Laboratory (1990).

[24] R. E. Alcouffe, R. S. Baker, J. A. Dahl, S.A. Turner, and Robert Ward, "PARTISN: A Time-Dependent, Parallel Neutral Particle Transport Code System," LA-UR-08-07258 (Revised Nov. 2008).

[25] G. Palmiotti et al, "Variational nodal transport methods with anisotropic scattering," Nuclear Science and Engineering, Vol. 115, pp. 233-243 (1993).

[26] G. Aliberti and M. Smith, "PERSENT: need of a deterministic code for sensitivity analysis in 3D geometry and transport theory," Proceedings of PHYSOR2014, Kyoto, Japan (2014).

[27] CUBIT Web page, www.cubit.sandia.gov.

[28] M. A. Smith and E. R. Shemon, "User Manual for the PROTEUS Mesh Tools," ANL/NE-15/17 (Rev.2), Argonne National Laboratory, September 19 (2016).

[29] Gerald Rimpault et al, "Objectives and Status of the OECD/NEA sub-group on Uncertainty Analysis in Best-Estimate Modelling (UAM) for Design, Operation and Safety Analysis of SFRs (SFR-UAM)," FR'17, Yekaterinburg, Russia.

[30] Nicolas E. Stauff et al., "Evaluation of the OECD/NEA/SFR-UAM Neutronics Reactivity Feedback and Uncertainty Benchmarks," FR'17, Yekaterinburg, Russia.

[31] Benchmark For Uncertainty Analysis In Modelling (UAM) For Design, Operation and Safety Analysis of SFRs, Core Definitions (version 1.5-6 – 2016 September 13th)

[32] M. B. Chadwick et al., "ENDF/B-VII.0: Next Generation Evaluated Nuclear Data Library for Nuclear Science and Technology," Nuclear Data Sheets 107, 2931 (2006).

[33] M. Herman et al, “COMMARA-2.0 Neutron Cross Section Covariance Library,” BNL-94830-2011 (2011).

[34] L. Buiron, P. Sciora, G. Rimpault, “Additional ASTRID Benchmark,” Draft Version, July (2019).

[35] Bo Feng and Nicolas Stauff, “High Power Density Annular Fuel in a Fast Test Reactor,” ANS Summer meeting, Philadelphia, PA, USA, June 17-21, (2018).

[36] Nicolas E. Stauff, F. Heidet, “Assessment of Low Enriched Uranium Fueled Core Configurations for the Versatile Test Reactor,” proceedings of ANS Annual 2019, Minneapolis, MN, June 9-13 (2019).

[37] Yinbin Miao, Nicolas Stauff, Aaron Oaks, Abdellatif M. Yacout, Taek K. Kim, “Fuel Performance Evaluation of Annular Metallic Fuels for an Advanced Fast Reactor Concept,” Nuclear Engineering and Design, Vol. 352, (2019). https://doi.org/10.1016/j.nucengdes.2019.110157

[38] I. T. Usman, P. Lartaud, and N. E. Stauff, “Sensitivity Analysis and Uncertainty Quantification of FFTF Cycle 8C using the NEAMS Workbench,” Submitted to ANS Winter Meeting (2019).

[39] Benchmark for Neutronic Analysis of Sodium-cooled Fast Reactor Cores with Various Fuel Types and Core Sizes, OECD Nuclear Energy Agency, February 2016, NEA/NSC/R(2015)9.

[40] A. BANERJEE, et al., "Thermal Property Characterization of a Titanium Modified Austenitic Stainless Steel (Alloy D9)," Journal of Nuclear Materials, pp. 20-30, 2005.

[41] J. H. KIM, et al., "Fabrication of Uranium Alloy Fuel Slug for Sodium-Cooled Fast Reactor by Injection Casting," Radioanal Nucl. Chem., pp. 797-803, 2014.

[42] J. FINK, et al., "Thermodynamic and Transport Properties of Sodium Liquid and Vapor," ANL/RE-95/2, Argonne National Laboratory, Lemont, IL, USA, 1995.

[43] X. LI, et al., "Fabrication and Characterization of B4C-based Ceramic Composites with Different Mass Fractions of Hexagonal Boron Nitride," Ceramics International, vol. 41, no. 1, pp. 27-36, 2014.

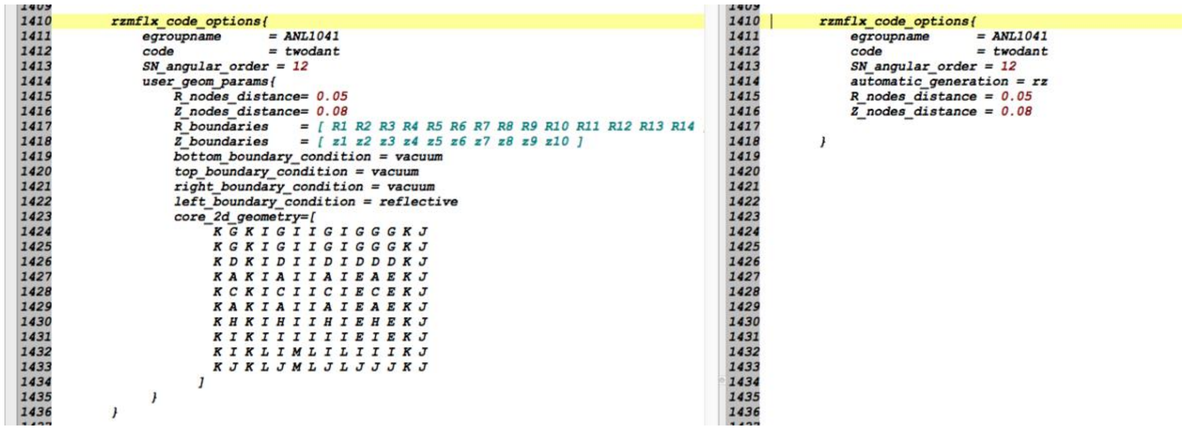
[44] J. KU, et al., "Analysis of IFR Driver Fuel Hot Channel Factors," ANL/RA/CP-80315, Argonne National Lab., Lemont, IL, USA, 1994.

APPENDIX A: AUTOMATIC GEOMETRIES GENERATION FOR MULTI CROSS-SECTIONS GENERATION

This appendix describes the methods developed within PyARC for automatic 1D and 2D/RZ geometries generation for MC2-3 and TWODANT (or PARTISN). Demonstration of the capabilities are provided based on developed test cases to assess the impact on estimated eigenvalue. Future work should include comparison with Monte Carlo solution to confirm resulting value is closer to the correct solution.

A.1 Automatic RZ Geometry Generation for TWODANT or PARTISN

The 2-step cross-section generation procedure with MC2-3 provides significant improvements in multi-group cross-sections accuracy when compared with the 1-step procedure [22], especially in processing the cross-sections of structure regions. However, it requires development of core-equivalent RZ geometry that will be used by TWODANT or PARTISN. This remains a tedious task for the user and a potential source of mistakes. A procedure was developed within PyARC to automatically generate a RZ geometry from the user-specified hexagonal-z geometry (in the *arc/geometry* card) and the improvement in user experience is illustrated with Figure A-1. It allows further reducing the amount of pre-processing needed from the user, and to reduce redundancies in model description provided in the PyARC input. This procedure is not yet available for Cartesian geometries.



```
rzmflx_code_options{
    egroupname       = ANL1041
    code             = twodant
    SN_angular_order = 12
    user_geom_params{
        R_nodes_distance= 0.05
        Z_nodes_distance= 0.08
        R_boundaries     = [ R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14
        Z_boundaries     = [ z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 ]
        bottom_boundary_condition = vacuum
        top_boundary_condition = vacuum
        right_boundary_condition = vacuum
        left_boundary_condition = reflective
        core_2d_geometry=[
            K G K I G I I G I G G G K J
            K G K I G I I G I G G G K J
            K D K I D I I D I D D D K J
            K A K I A I I A I E A E K J
            K C K I C I I C I E C E K J
            K A K I A I I A I E A E K J
            K H K I H I I H I E H E K J
            K I K I I I I I I E I E K J
            K I K L I M L I L I I I K J
            K J K L J M L J L J J J K J
          ]
    }
}
```

```
rzmflx_code_options{
    egroupname       = ANL1041
    code             = twodant
    SN_angular_order = 12
    automatic_generation = rz
    R_nodes_distance = 0.05
    Z_nodes_distance = 0.08

}
```

Figure A-1. Example of PyARC input with manual (left) and automatic (right) RZ geometry generation.

The RZ-geometry generated for TWODANT is typically obtained by calculating the area in the core containing the different cells of the MC2-3 calculation. Then, starting from the most inner rings of the core, the user builds the cylindrical geometry with the corresponding cells. The radiuses giving the boundaries between each radial region are chosen so that the corresponding area matches the actual area of the cells' regions using equation (1), where R represents the equivalent radius, N represents the number of assemblies considered with a pitch of p. The difficulty when trying to automatize this process is that there is no generic rule and different users could choose to build two accurate different geometries for the same model.

$$V = \pi * R^2 = N * \frac{\sqrt{3}}{2} * p^2 \quad (1)$$

The approach developed constructs a complete RZ map of the core with all the sub-assemblies, starting from the inner rings. Then, this map is reduced as much as possible to minimize the number of radial regions and the runtime of TWODANT, while maintaining the model accuracy. To do so, adjacent and non-adjacent (but separated by less than one ring) identical sub-assemblies are merged into one region. The purpose of this approach is to guarantee the viability of the RZ geometry.

To test this new implementation, a test core was created in the Workbench. The core is composed of two types of fuel assemblies, as shown in Figure A-2. It is used to perform TWODANT and DIF3D calculations with the automatic RZ geometry automatically generated by PyARC, and to compare the results with calculations on a RZ geometry created manually. Both RZ geometries are different but are valid approximations of the full 3D geometry. A calculation without TWODANT was also performed. The two different RZ geometries, as well as the results, are presented in Table A-1. The plots of the RZ geometries shown in Figure A-3 are provided through an automatic feature developed with the new implementation to provide visualization tools to the PyARC user (generated after PyARC pre-processing step).

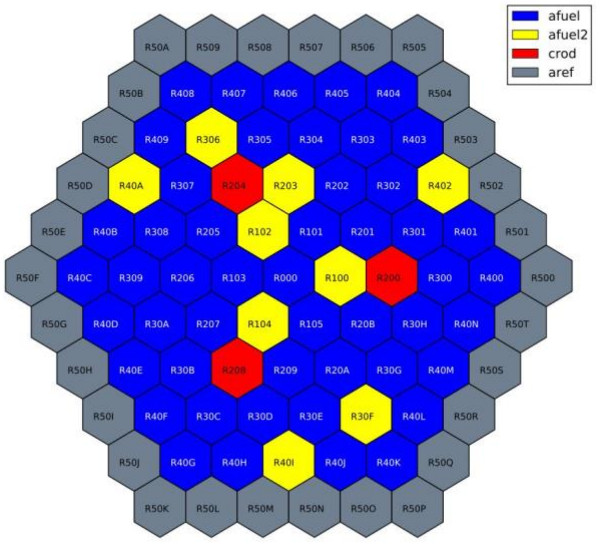

afuel
afuel2
crod
aref
R50A R509 R508 R507 R506 R505
R50B R408 R407 R406 R405 R404 R504
R50C R409 R306 R305 R304 R303 R403 R503
R50D R40A R307 R204 R203 R202 R302 R402 R502
R50E R40B R308 R205 R102 R101 R201 R301 R401 R501
R50F R40C R309 R206 R103 R000 R100 R200 R300 R400 R500
R50G R40D R30A R207 R104 R105 R20B R30H R40N R50T
R50H R40E R30B R208 R209 R20A R30G R40M R50S
R50I R40F R30C R30D R30E R30F R40L R50R
R50J R40G R40H R40I R40J R40K R50Q
R50K R50L R50M R50N R50O R50P

Figure A-2. Geometry of the core used to test the automatic RZ geometry generation.

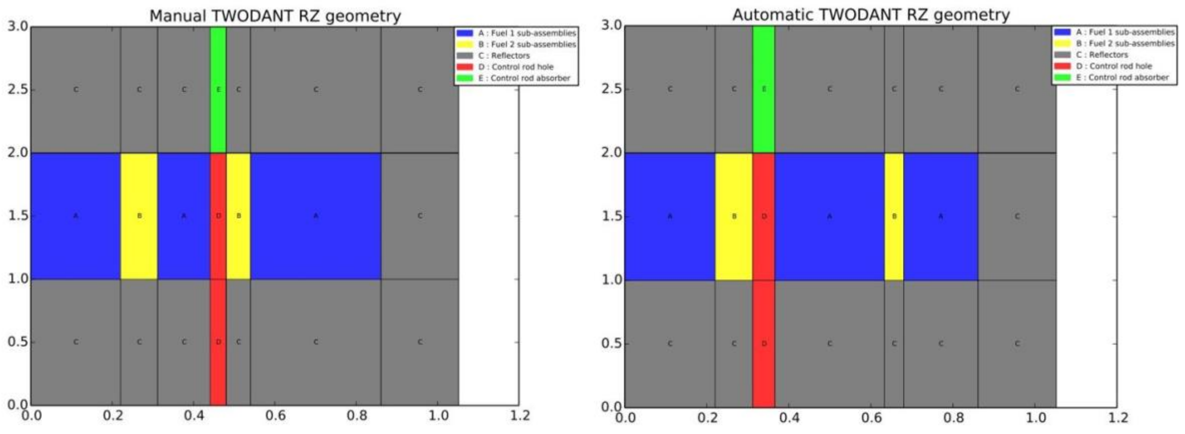


Figure A-3. Two RZ geometries created automatically (right) and manually (left).

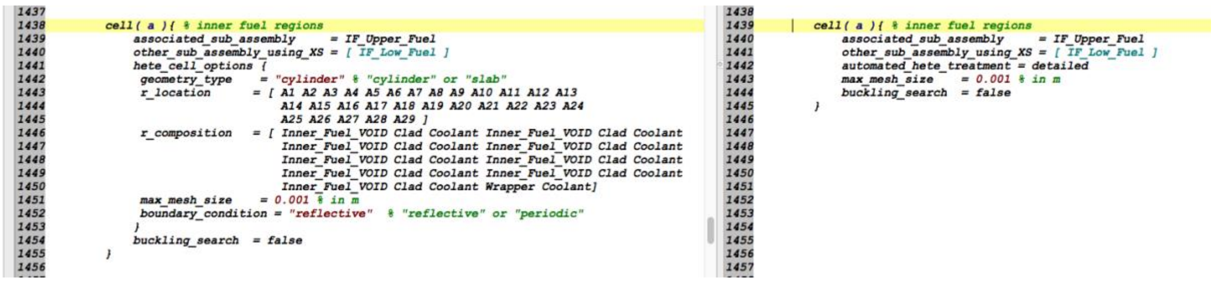
Even though the two geometries present a difference of 300 pcm for the k-effective in TWODANT, the impact on DIF3D is minor: only 13 pcm of difference for the k-effective between the two RZ geometries provided. The geometry automatically created provides reasonable results for this core model. The impact of the 2-step cross-section processing with TWODANT calculation on the K-effective estimate from DIF3D is about 380pcm.

Table A-1. Impact of the RZ core model for the test core.

	Manual RZ geometry	Automatic RZ geometry	Without TWODANT
k-effective (TWODANT)	0.977023	0.980305	NA
k-effective (DIF3D)	0.979149	0.979279	0.975414

A.2 Automatic 1D Geometries Generation for MC^2-3

The MC2-3 1D heterogeneous treatment allows to account for spatial self-shielding when generating the multi-group cross sections for each cell. The resonance spatial self-shielding is characterized by a change in the neutron flux level and spectrum in the cell geometry. Even though this effect is much reduced in fast reactors than in thermal reactors, the effect still is important to account for in fuel lattice region (impact can be up to 1000 pcm) and in control rod regions (impact can be up to 20% on the control rod worth) [22, 39]. Similar approach to the one above described was developed to automatically generate the MC2-3 1D-geometry, which helps reducing the pre-processing time of the user, and potential for user mistakes. An example of PyARC inputs with and without the automatic 1D geometry description is displayed in Figure A-4.



```
        cell( a ){ % inner fuel regions
            associated_sub_assembly      = IF_Upper_Fuel
            other_sub_assembly_using_XS = [ IF_Low_Fuel ]
            hete_cell_options {
             geometry_type     = "cylinder" % "cylinder" or "slab"
             r_location       = [ A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13
                                  A14 A15 A16 A17 A18 A19 A20 A21 A22 A23 A24
                                  A25 A26 A27 A28 A29 ]
             r_composition    = [ Inner_Fuel_VOID Clad Coolant Inner_Fuel_VOID Clad Coolant
                                  Inner_Fuel_VOID Clad Coolant Inner_Fuel_VOID Clad Coolant
                                  Inner_Fuel_VOID Clad Coolant Inner_Fuel_VOID Clad Coolant
                                  Inner_Fuel_VOID Clad Coolant Inner_Fuel_VOID Clad Coolant
                                  Inner_Fuel_VOID Clad Coolant Wrapper Coolant]
             max_mesh_size     = 0.001 % in m
             boundary_condition = "reflective"  % "reflective" or "periodic"
            }
            buckling_search  = false
        }
```

```
        cell( a ){ % inner fuel regions
            associated_sub_assembly      = IF_Upper_Fuel
            other_sub_assembly_using_XS = [ IF_Low_Fuel ]
            automated_hete_treatment = detailed
            max_mesh_size     = 0.001 % in m
            buckling_search  = false
        }
```

Figure A-4. Example of PyARC input with manual (left) and automatic (right) 1D geometry generation.

There are different ways of building a 1D equivalent geometry of an assembly. The most common is to find a ***pin-equivalent***: the assembly is reduced to one fuel region, one cladding region and the outer coolant region, modeling an average pin of the assembly including the duct. However, this method lacks in precision as it does not provide the whole geometry of the assembly. The more ***detailed*** method describes the full 2D geometry, ring by ring, in the same way that was done in the previous section on the whole core scale. The 2D assembly geometry in Figure A-5 is used to illustrate the two methods those lead to the 1D geometries shown in Figure A-6.

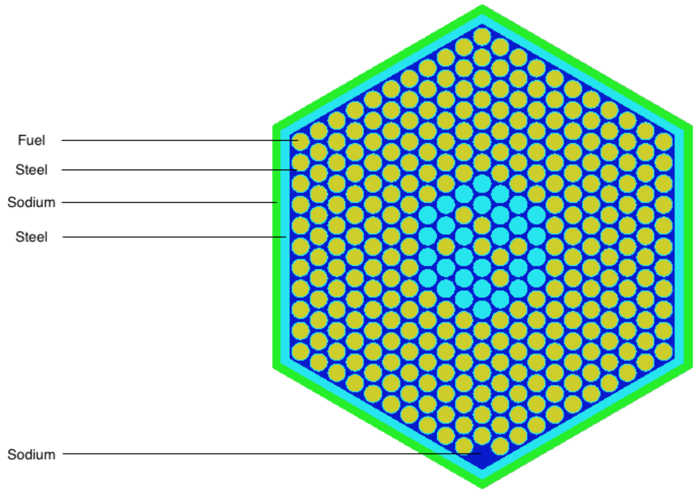


Figure A-5. 2D description of the test geometry.

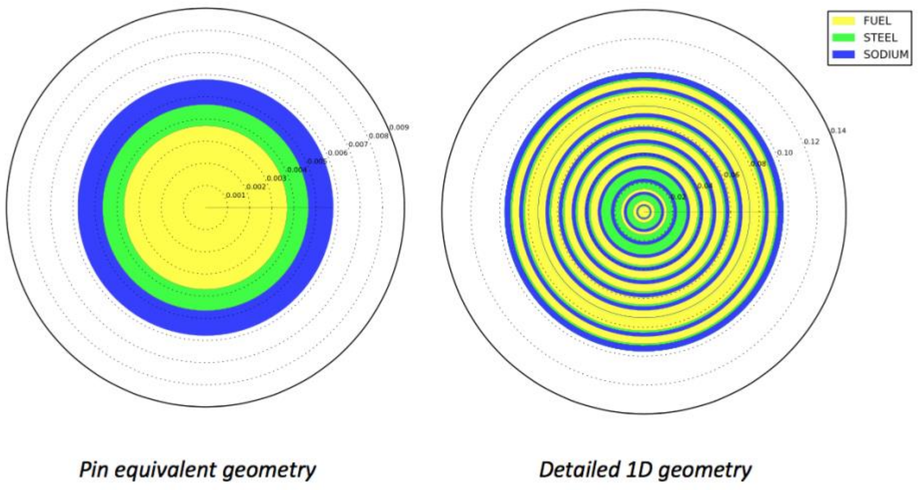


Figure A-6. Automatically-generated 1D geometries for the test-case.

On the left of Figure A-6 is the pin-equivalent geometry. The assembly is reduced to an average pin. The steel cladding of this equivalent-pin includes the duct. This type of geometry is not compatible with the description of non-fissile regions with a given external composition. In control rods, for instance, the neutron flux is high on the periphery of the assembly and drops drastically inside the assembly due to neutron absorption. A pin-equivalent geometry would not accurately model the progressive decrease in the flux.

On the right of Figure A-6 is the detailed geometry of the assembly, modeled ring by ring. This geometry is more precise and will provide a better description of the self-shielding effect on the assembly. However, the larger number of regions does increase the running time of MC$_2$-3.

The comparison of the different 1D models on the estimated assembly k-effective is provided in Table A-2. As expected, the k-infinite with homogeneous treatment is equal for the two different 1D geometries, because both geometries provide the same volume fractions. As observed, the heterogeneous effect tends to increase the k-inf, because the neutron flux in the resonance regions is lowered, which decreases the number of captures. The difference of 130 pcm in the heterogeneous treatment shows that the detailed geometry provides a better description of the assembly and of the self-shielding effect. Finally, the table presents a difference of 140 pcm between the k-effective provided by the two geometries. This difference is reasonable, especially considering an assembly with such level of heterogeneity. Overall, the two different 1D geometries provides reasonable results even though the detailed geometry should be more precise.

Table A-2. Impact of the 1D heterogeneous model for the test assembly.

	Pin equivalent geometry	Detailed geometry	Homogeneous
k-inf (MC_2-3 homogeneous)	1.892600	1.892600	1.891840
k-inf (MC_2-3 heterogeneous)	1.893140	1.894470	NA
k-effective (DIF3D)	1.892571	1.893913	1.892031

APPENDIX B: UNCERTAINTY CONTRIBUTORS IN FULL CORE UAM-SFR BENCHMARKS

	ABR-1000			MOX3600		
	Total Uncertainty	Top 5 uncertainty contributors		Total Uncertainty	Top 5 uncertainty contributors	
k_{eff}	1.29%	U238-inelastic	0.94%	1.37%	U238-inelastic	1.15%
		FE56-elastic	0.35%		U238-elastic	0.25%
		U238-elastic	0.30%		PU240-nu	0.22%
		PU240-nu	0.28%		PU239-chi	0.22%
		FE56- inelastic	0.22%		PU239-fission	0.19%
β_{eff}	1.14%	U238- inelastic	0.94%	1.16%	U238-inelastic	1.03%
		PU240 –nu	0.36%		U238-nu	0.31%
		U238-nu	0.27%		PU240-nu	0.29%
		U238-elastic	0.25%*i		PU239-chi	0.22%
		FE56- inelastic	0.23%		U238-elastic	0.21%*i
Lambda	4.28%	U238- inelastic	2.92%	4.08%	U238- inelastic	3.24%
		FE56-elastic	1.41%		U238-elastic	0.89%
		U238-elastic	0.97%		FE56-elastic	0.77%
		NA23--elastic	0.88%		PU239-chi	0.51%
		FE56- inelastic	0.70%		NA23-elastic	0.46%
K_D	5.86%	U238- inelastic	3.34%	4.83%	U238- inelastic	3.59%
		NA23-elastic	3.14%		O16-elastic	1.50%
		FE56-elastic	2.34%		NA23-elastic	1.49%
		PU239- inelastic	0.93%		FE56-elastic	1.07%
		FE56- inelastic	0.85%		U238-elastic	0.65%
Na Void Worth	19.67%	FE56-elastic	13.14%	7.91%	NA23- inelastic	4.00%
		NA23-P1elastic	6.81%		U238- inelastic	3.27%
		FE56-P1elastic	6.52%		FE56-P1elastic	2.87%
		NA23- inelastic	6.25%		FE56-elastic	2.75%
		U238- inelastic	4.20%		NA23-elastic	2.16%
1% Sodium	12.32%	FE56-elastic	6.87%	7.20%	NA23- inelastic	4.03%
		NA23- inelastic	5.17%		U238- inelastic	3.94%
		U238- inelastic	4.03%		NA23-elastic	2.04%
		NA23-P1elastic	3.70%		U238-elastic	1.45%
		FE56-P1elastic	3.57%		NA23-P1elastic	1.35%
1% Duct	7.87%	U238- inelastic	5.93%	6.48%	U238- inelastic	5.24%
		FE56- inelastic	2.84%		FE56- inelastic	2.35%
		U238-elastic	2.72%		U238-elastic	1.69%
		FE56-elastic	1.04%		FE56-elastic	0.77%
		NA23-P1elastic	0.87%		PU239-fission	0.46%
1% Cladding	7.87%	U238-inelastic	5.93%	6.23%	U238- inelastic	5.26%
		FE56-inelastic	2.84%		U238-elastic	1.70%
		U238-elastic	2.72%		FE56- inelastic	1.63%
		FE56-elastic	1.04%		FE56-elastic	0.68%
		NA23-P1elastic	0.87%		PU239-fission	0.49%
1% Fuel	2.00%	FE56-elastic	1.47%	1.85%	U238- inelastic	1.14%
		FE56-P1elastic	0.61%		FE56-elastic	0.83%
		FE56-inelastic	0.50%		FE56- inelastic	0.51%
		NA23-elastic	0.44%		FE56-P1elastic	0.43%
		NA23-P1elastic	0.41%		U238-elastic	0.37%*i

Note: *i represents negative contribution to uncertainty variance.

APPENDIX C: ASSESSMENT OF MANUFACTURING UNCERTAINTIES ON ABR-1000 BENCHMARK CORE

The adjoint-based method proposed in PERSENT is computationally inexpensive, and can provide detailed sensitivity and uncertainty information contributed from different isotopes, reaction types, and energy range. As a result, this method is widely used in quantifying the uncertainty from nuclear data. Another source of input uncertainty considered in this study are the manufacturing tolerances, such as fuel density and rod radius. DAKOTA has been coupled with ARC to perform SA/UQ using stochastic sampling approach within the NEAMS Workbench [18] with the following procedure:

1. The NEAMS Workbench simplified model was developed, and further parameterized as a template file with a list of input manufacturing parameters with associated probabilistic distributions.
2. DAKOTA was employed for generating *N* random samples of input parameters using Latin Hypercube Sampling (LHS) technique, with which corresponding *N* sampled Workbench input files could be generated from the template developed in step 1.
3. After all the samples were executed, output responses were extracted and returned to DAKOTA for sensitivity and uncertainty analysis. The best-estimated value and the uncertainty of an output response *R* was represented with the sample mean value and associated standard deviation, respectively as shown in Eq. (1) and (2):

$$\overline{R} = \frac{1}{N}\sum_{j=1}^{N} R_j \quad (1)$$

$$\sigma_R = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}\left(R_j - \overline{R}\right)^2} \quad (2)$$

The confidence level of the calculated uncertainty depends on the sample size. In this work, the 95% confidence interval of the calculated uncertainty was reported, which covers the true uncertainty with 95% confidence. In order to calculate the 95%/95% confidence interval, the output response must be justified as normally distributed. The Anderson-Darling normality test was performed to compute the so-called A_2 value, which measures the average deviation of the distribution of output response from the perfect normal distribution. The normality hypothesis is rejected if the calculated A_2 value is larger than threshold value, which is 0.759 and 0.718 under sample size of 100 and 1000, respectively. Once the output response is quantified as normally distributed, the following chi-square distribution could be constructed using the calculated ($\sigma^2_{calc.}$) and true (σ^2_{true}) variance of output response *R*:

$$\chi^2 = \frac{(N-1)\sigma^2_{calc.}}{\sigma^2_{true}} \quad (3)$$

Due to the limitation of computational resources, currently a small sample size *N=100* is used to quantify the uncertainty propagated from manufacturing tolerance. For the 95% confidence level, the $\chi^2_{2.5\%}$ and $\chi^2_{97.5\%}$ can be mathematically evaluated, and the 95%/95% confidence interval is then determined as [88%$\sigma_{calc.}$, 116%$\sigma_{calc.}$].

The input manufacturing parameters are perturbed according to the uncertainties and distributions summarized in Table C-3. The manufacturing uncertainty of the density of D9 steel is estimated in [40], and was used as the uncertainty of HT9 density because HT9 and D9 have very similar compositions. The uncertainty of the height of fuel rod is estimated to be 5% by considering the average swelling ratio. The uncertainties of duct thickness and gap thickness are assumed to be the same as that in cladding thickness.

Table C-3 – Input Manufacturing Uncertainties and References.

	Rel. STD.	References
Fuel density	0.5% (Normal)	[41]
Coolant density	0.3% (Normal)	[42]
HT9 density	1.17% (Normal)	[40]
B4C density	0.65% (Normal)	[43]
Fuel rod radius	2% (Normal)	[44]
Cladding thickness	3% (Normal)	[44]
Height of fuel rod	2% (Normal)	Estimated
Duct thickness	3% (Normal)	Estimated
Gap thickness	3% (Normal)	Estimated

The impact of manufacturing tolerance on core eigenvalue and feedback coefficients are summarized in Table C-4. The uncertainties of core output responses are evaluated with consideration of all input manufacturing uncertainties. The uncertainties contributed from different input manufacturing parameters are also evaluated by separately perturbing corresponding input parameters. It was found that fuel density, rod radius and height of fuel rod are the most influential parameters to core output responses. A large uncertainty of 10.20% is observed in sodium density feedback coefficient, which is contributed mostly from the density of HT9, the height and radius of fuel rods.

Table C-4 – Impact of manufacturing uncertainties on core performance.

	k_{eff}	β_{eff}	$\Delta\rho_{\text{CR}}$	$\Delta\rho_{\text{cool}}$	$\Delta\rho_{\text{str}}$	$\Delta\rho_{\text{fuel}}$	$\Delta\rho_{\text{ax}}$	$\Delta\rho_{\text{rad}}$
All	1.86%	0.18%	4.19%	8.93%	2.19%	2.04%	2.08%	3.81%
Fuel density	0.21%	0.00%	0.58%	0.66%	0.18%	0.45%	0.45%	0.46%
Coolant density	0.00%	0.00%	0.02%	0.35%	0.01%	0.02%	0.02%	0.11%*
HT9 density	0.03%	0.07%	0.10%	1.25%	1.26%	0.06%	0.06%*	0.94%*
B4C density	0.00%	0.00%	0.46%	0.08%	0.02%	0.01%	0.01%*	0.02%*
Fuel rod radius	1.71%	0.01%	3.58%	7.65%	0.74%	1.89%	1.62%	3.72%
Cladding thickness	0.07%	0.00%	0.81%	0.74%	0.34%	0.34%	0.98%	0.34%
Duct thickness	0.10%	0.00%	0.00%	0.13%	0.12%	0.12%	0.18%	0.34%
Gap thickness	0.19%	0.02%	0.01%	0.23%*	0.19%*	0.27%*	0.41%	0.34%

Anderson Darling normality tests were performed for all output responses. It is found that the normality assumption is valid for most of the distributions of output responses. For example, Figure C-7 shows the distribution of k_{eff} extracted from 100 samples when all input manufacturing uncertainties are considered. The corresponding A_2 value is found to be 0.279 (<0.759), which means that the average deviation between k_{eff} distribution and perfect normal distribution is small enough such that k_{eff} can be regarded as normally distributed. With this normality assumption and the calculated sample standard deviation of 1.86%, the 95%/95% confidence interval of the true uncertainty (i.e., 'uncertainty of the true uncertainty') can be determined as [1.64%, 2.16%] according to Eq. (9). Similar procedure could be applied to quantify the 95%/95% confidence intervals for the rest of output responses.

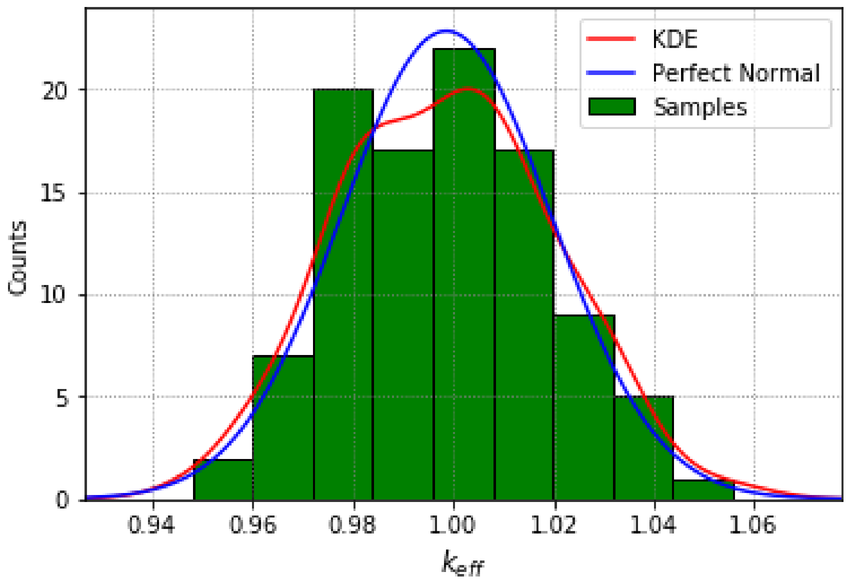


Figure C-7. Statistical distribution of k_{eff}.

For each perturbed core calculation, only one sample of input parameters is generated and assigned to the whole core. Besides that, this work also investigates the impact of different perturbation options. The input manufacturing parameters are perturbed based on the following three perturbation options and the uncertainties of output responses are compared:

- Option #1: One sample of input parameters is generated and assigned to the whole core;
- Option #2: Two samples of fuel rod radius, cladding radius duct thickness and gap thickness is generated, each assigned to fuel assemblies in inner core region and outer core region, respectively. The fuel rod is axially divided into 5 sections with different fuel enrichments, respectively for inner core fuel and outer core fuel assemblies. Therefore, 10 samples of density related input parameters are generated, each assigned to the fuel sections.

Table C-6 presents the absolute difference of the uncertainties obtained with two perturbation options. It is found that perturbation option 2 in general tends to predict a larger uncertainty, due to the increase of statistical uncertainty introduced with the random sampling. It is also observed that different perturbation options have large impact of the evaluation of core uncertainties, as the relative difference between two perturbation options can be as large as 100% in uncertainty prediction of some output responses. It is therefore important to take into account the spatial correlation in uncertainty evaluation.

Table C-5 – Impact of manufacturing uncertainties on core performance with perturbation option 2.

	k_{eff}	β_{eff}	$\Delta\rho_{\text{CR}}$	$\Delta\rho_{\text{cool}}$	$\Delta\rho_{\text{str}}$	$\Delta\rho_{\text{fuel}}$	$\Delta\rho_{\text{ax}}$	$\Delta\rho_{\text{rad}}$
All	1.90%	0.22%	4.23%	8.99%	2.23%	2.09%	2.12%	3.85%
Fuel density	0.21%	0.01%	0.52%	0.66%	0.18%	0.45%	0.45%	0.46%
Coolant density	0.00%	0.01%	0.02%	0.35%	0.01%	0.02%	0.02%	0.11%
HT9 density	0.03%	0.07%	0.11%	1.25%	1.26%	0.06%	0.06%	0.94%
B4C density	0.00%	0.00%	0.46%	0.08%	0.02%	0.01%	0.01%	0.02%
Fuel rod radius	1.71%	0.01%	3.58%	7.65%	0.74%	1.89%	1.62%	3.72%
Cladding thickness	0.07%	0.00%	0.81%	0.74%	0.33%	0.33%	0.98%	0.34%
Duct thickness	0.12%	0.00%	0.04%	0.13%	0.12%	0.11%	0.18%	0.34%
Gap thickness	0.19%	0.02%	0.01%	0.23%	0.19%	0.27%	0.41%	0.34%

Table C-6 – Impact of different perturbation options (in pcm).

	k_{eff}	β_{eff}	$\Delta\rho_{\text{CR}}$	$\Delta\rho_{\text{cool}}$	$\Delta\rho_{\text{str}}$	$\Delta\rho_{\text{fuel}}$	$\Delta\rho_{\text{ax}}$	$\Delta\rho_{\text{rad}}$
All	40.00	42.42	42.69	60.00	42.43	50.00	42.90	42.19
Fuel density	1.00	10.00	-62.00	1.00	1.00	3.00	2.30	-3.24
Coolant density	0.01	10.00	1.00	2.00	1.00	0.10	0.04	1.00
HT9 density	1.00	0.10	9.00	3.00	4.00	4.50	1.00	3.00
B4C density	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Fuel rod radius	2.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Cladding thickness	3.20	0.10	0.30	3.23	-5.60	-7.56	1.23	2.13
Duct thickness	21.32	2.23	39.00	3.24	2.30	-7.80	3.24	2.30
Gap thickness	3.00	1.20	1.20	2.32	2.34	3.43	2.34	2.34