Argonne NATIONAL LABORATORY

ANL/NEAMS-20/2

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

Integration of PyARC/Workbench with New Fast Reactor Modeling and Simulation Capabilities

**Nuclear Science and Engineering Division**

**About Argonne National Laboratory**
Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

**DOCUMENT AVAILABILITY**

**Online Access:** U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free at OSTI.GOV (http://www.osti.gov/), a service of the US Dept. of Energy's Office of Scientific and Technical Information.

**Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):**
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandra, VA 22312
**www.ntis.gov**
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

**Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):**
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
**www.osti.gov**
Phone: (865) 576-8401
Fax: (865) 576-5728

**Disclaimer**
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

**ANL/NEAMS-20/2**

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

Integration of PyARC/Workbench with New Fast Reactor Modeling and Simulation Capabilities

prepared by
N. Stauff
Nuclear Science and Engineering Division, Argonne National Laboratory

September 30, 2020

## ACKNOWLEDGMENT

The author would like to acknowledge contributions from various developers and beta-testers in the past years:

- ANL: Y. S. Jung, T. K. Kim, T. Fei, C. H. Lee, B. Feng
- ORNL: R. Lefebvre, B. Langley
- Summer interns: N. Gaughan, P. Lartaud, P. Seurin, K. Zeng, A. Rivas

## EXECUTIVE ABSTRACT

The Workbench initiative was launched in FY-2017 within the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program 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 modeling community 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, through PROTEUS integration within the same input logic as the legacy ARC codes.

In FY-2020, the effort focused on integrating REBUS to ORIGEN-S workflow to enable detailed isotopic composition and decay heat calculation. Additional capabilities were also integrated in response to user requests, such as the generation of the covariance matrix within the nuclear data uncertainties on reactivity coefficients. Significant effort was continued in FY-2020 to train and support users from ANL, and Westinghouse.

The ARC codes are 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) Workbench is to facilitate the deployment of the high-fidelity codes developed within the NEAMS program. The Workbench [1] initiative was launched in FY-2017 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][5].

The ARC suite of codes [6] 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 [7] for multi-group cross-section processing, DIF3D [8] for flux calculation, REBUS-3 [9] for depletion and equilibrium calculations, PERSENT [10] for perturbation theory calculations (perturbation, sensitivity and uncertainty quantification), GAMSOR [11] for gamma heating calculations, and PROTEUS transport solvers [12][13]. These ARC and NEAMS 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 them 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. Their integration was accomplished through the development of the PyARC module within the Workbench.

Both the Workbench [14] and PyARC are being distributed under Open Source Software licenses. The status of the PROTEUS and ARC integration [15] is described in this report, focusing on FY-2020 developments (highlighted in this report) and on description of the new released version 1.1.0 of PyARC. The code 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. Finally, Chapter 4 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.

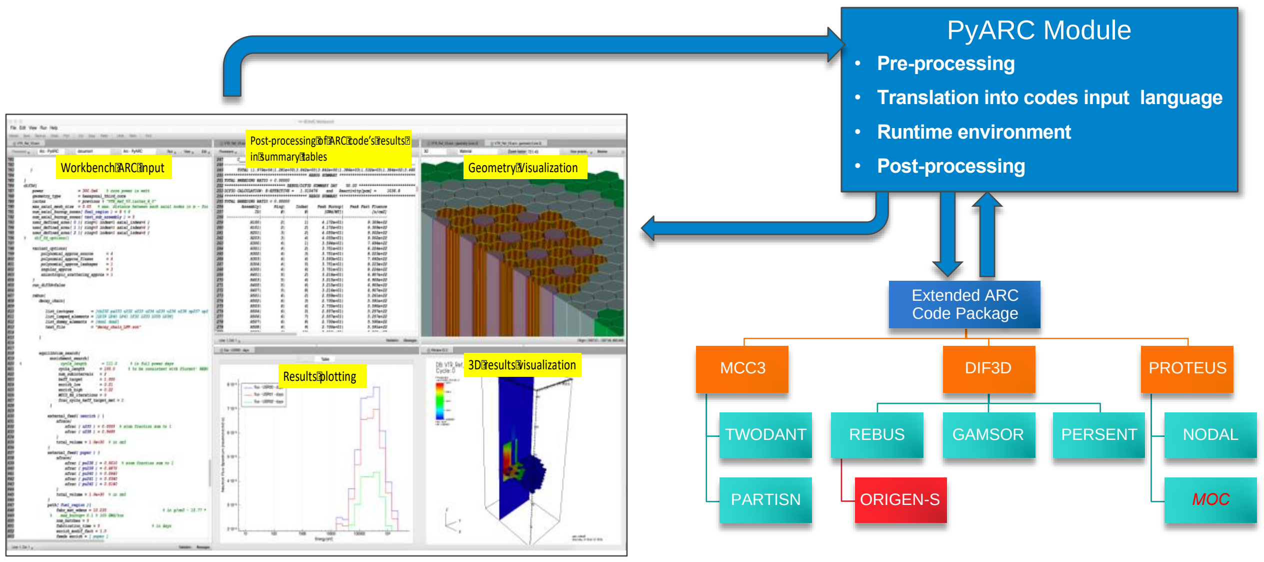


Figure 2-1. Structure of the ARC integration in the Workbench.

### 2.1 The Workbench Interface

The Workbench [1] 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) [16] 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 2.5) 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 **113 templates** were generated for PyARC. Templates are also relied upon by the Dakota/PyARC coupling, as explained in Section 2.4.

### *2.1.3 Visualization*

The Workbench provides different built-in types of visualization capabilities that PyARC 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 [17] 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).

A utility script provides 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.

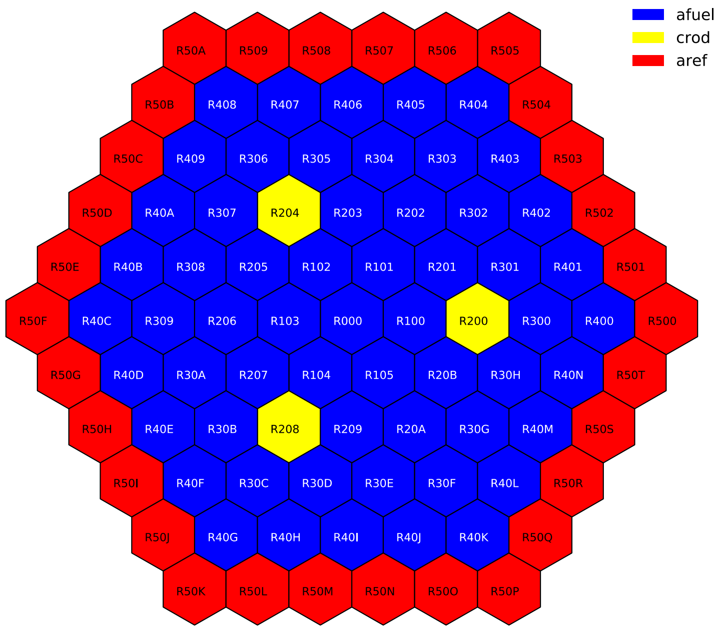

afuel
crod
aref
R50A R509 R508 R507 R506 R505
R50B R408 R407 R406 R405 R404 R504
R50C R409 R306 R305 R304 R303 R403 R503
R50D R40A R307 R204 R203 R202 R302 R402 R502
R50E R40B R308 R205 R102 R101 R201 R301 R401 R501
R50F R40C R309 R206 R103 R000 R100 R200 R300 R400 R500
R50G R40D R30A R207 R104 R105 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 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 more than 14,000 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, PyREBORS, 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, more than **200 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, ORIGEN-S), 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 Workflow*

PyARC takes a ".son" input that is generated through the Workbench interface (or using any text editor) following the formatting defined in Section 2.1.1. Examples of inputs are provided in the training material described in Section 2.5. In addition to the main input, additional input files may be requested to provide decay chain (with REBUS native structure), fission yields of lumped fission products, covariance matrix for uncertainty calculations, etc. A repository of such pre-built files is made available.

PyARC simulations are executed through the "run" button on the Workbench interface. Alternatively, the following commend can be used to execute PyARC input:

```
/link/to/Workbench/rte/entry.sh -i my_pyarc_input.son
```

Every PyARC simulations will generate the following output files:

- "*.summary*": summary of the output files from each ARC simulations.
- "*.timeline.out*": timeline of calculation for real-time check of the status of the calculation, and to save the computational time spent at every step of the workflow.
- "*.inp*": all input files generated by PyARC concatenated into one single file
- "*.out*": all output files generated from ARC runs concatenated in one single file
- "*.vtk*": region-wise results in a VisIt-readable format
- "*.zip*": gathers all input, output, isotxs, and summary files
- "*.user_info.out*": important information about the model, including 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.

Additional output files may be generated when running different codes, as discussed in Section 3.

## *2.4 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 [18] code and PyARC was demonstrated in [19],[20],[21], and [22]. 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.

### *2.4.1 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.

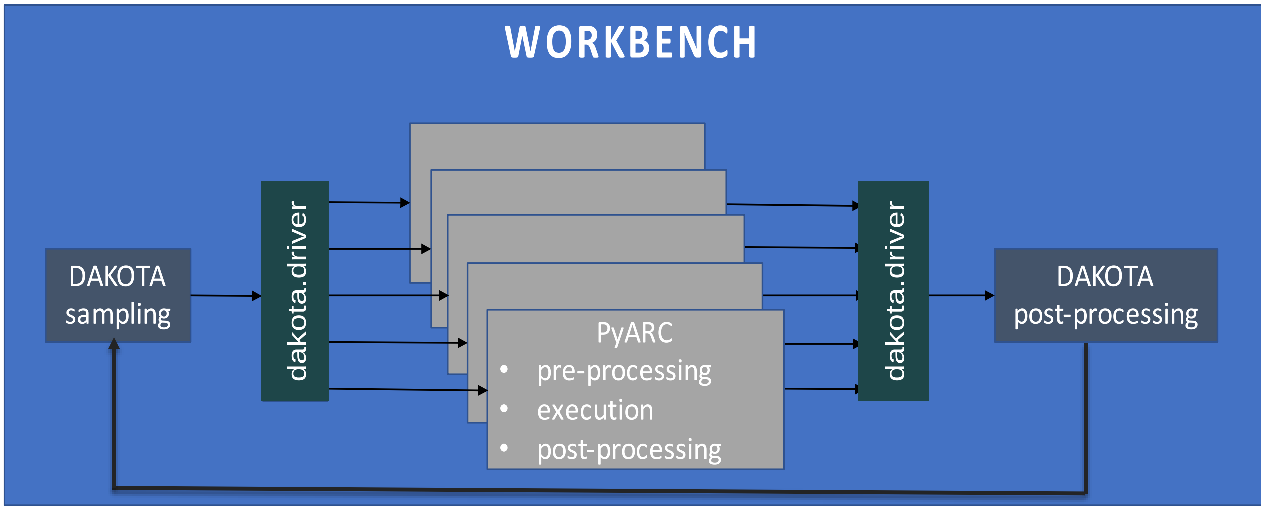


Figure 2-3. Schematic of the Dakota/PyARC coupling.

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.

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.

### *2.4.2 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 [19], [20], and core design optimization with the ARC codes [21], [22].

#### 2.4.2.1 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.

#### 2.4.2.2 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 [10]) and through stochastic sampling (with Dakota [18]) 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 of [5].

## *2.5 Training Material*

Training material is available to assist a user getting started. It consists in a list of sample problems provided within the PyARC released bundle, that are documented and that demonstrate and explain the most popular capabilities:

- Sample #1 - MC$^2$-3 cross-section processing with 1step approach, and modeling full-core with DIF3D diffusion.
- Sample #2 - DIF3D calculation with VARIANT for full-core simulation.
- Sample #3 - MC$^2$-3 cross-section processing with 1D heterogeneous model and 2steps (TWODANT) calculation, modeling full-core with DIF3D finite diffusion.
- Sample #4 - MC$^2$-3 cross-section processing with homogeneous - 1step calculation, and core once-through fuel depletion calculation with REBUS 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 MC$^2$-3 and REBUS equilibrium).
- Sample #7 - GAMSOR calculation for gamma transport 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)
- Sample #14 - REBUS to ORIGEN-S calculation

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. It also contains the inputs to the SFR-UAM benchmark problems [23][5].

# 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, while highlighting the work completed in FY-2020. Continuing efforts are underway to implement additional modeling capabilities based on the ARC and NEAMS codes.

## 3.1 MC$^2$-3 [7]

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 [24] 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 TWODANT [25] or PARTISN [26], 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.

Additional Notes:

- PyARC provides 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 useful 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.

- MC$^2$-3 is also applied for computing the DLAYXS, GAMISO and PMATRX required for PERSENT (delayed neutron fraction), PROTEUS (MSR) and GAMSOR calculations, as further discussed in Sections 3.4, 3.5, and 3.6.1.
- Automatic generation of RZ geometry for TWODANT or PARTISN and 1D geometries for MC$^2$-3 are available 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 of [5]. Sample #3 provides an example to get started with these options.

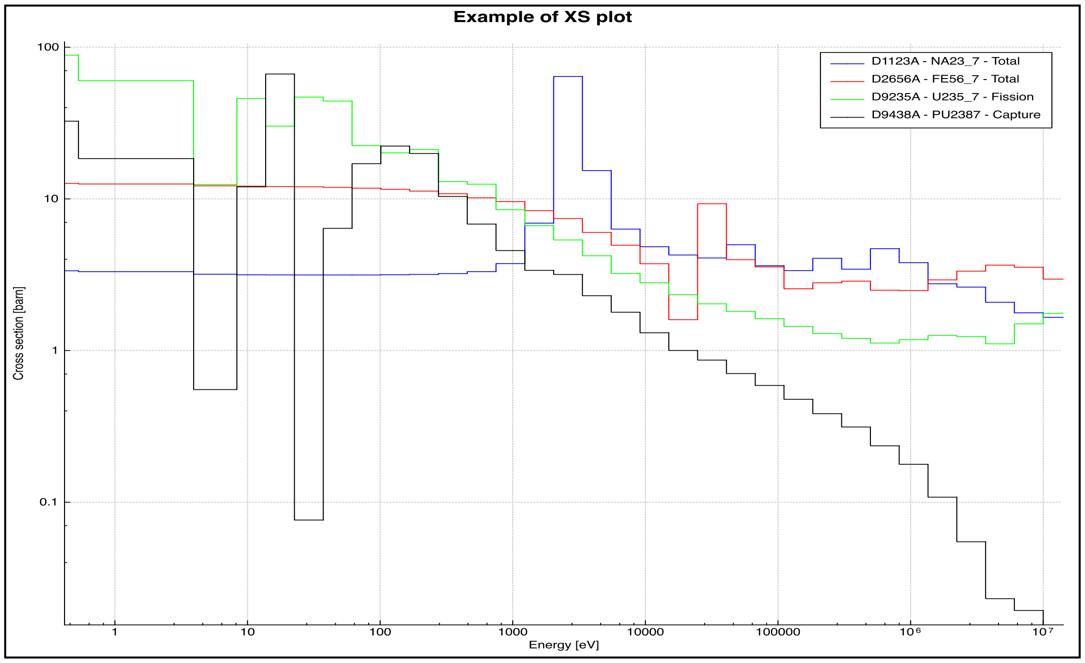


Figure 3-1. Example of multi-group XS plot automatically generated with the Workbench from ISOTXS edit file.

## *3.2 DIF3D [8]*

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 MC$^2$-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 [27]) 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 directly 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.

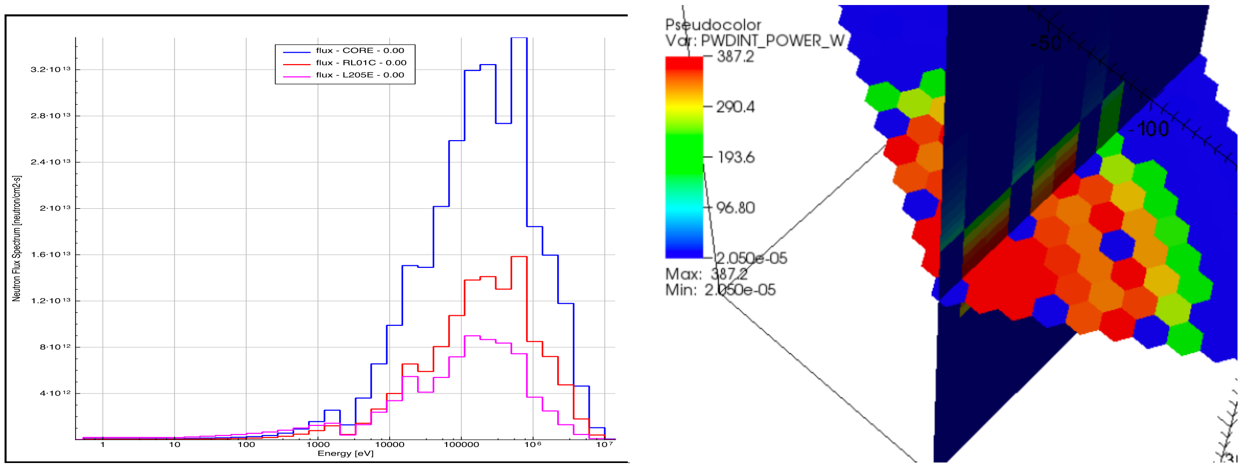


Figure 3-2. Example of visualizations available for DIF3D and REBUS-3: neutron flux spectrum (left) and power map (right) calculated and plotted with the Workbench.

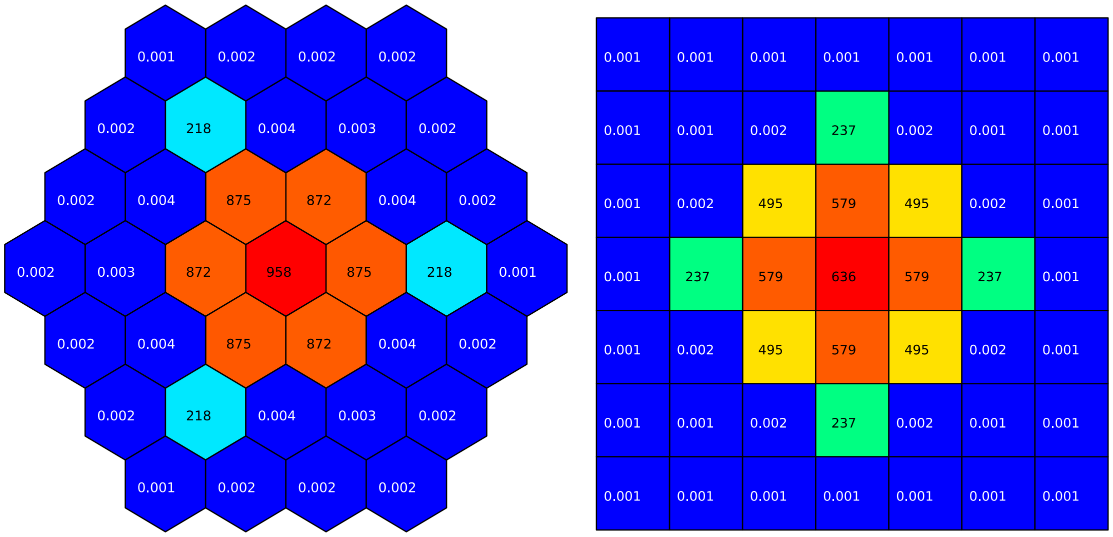


Figure 3-3. Example of 2D plot visualization of peak power density per assembly for hexagonal and Cartesian geometry.

### 3.3 REBUS-3 [9]

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 optimized enrichment (computed for equilibrium calculation), etc.

PyARC integration of REBUS-3 allows its user specifying its own decay chain 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. Some examples of decay chains (and associated fission yields) are provided in AdditionalFiles.

The once-through depletion capability is illustrated in Sample #4. The original option to re-calculate the multi-group cross-sections at different time-steps 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.

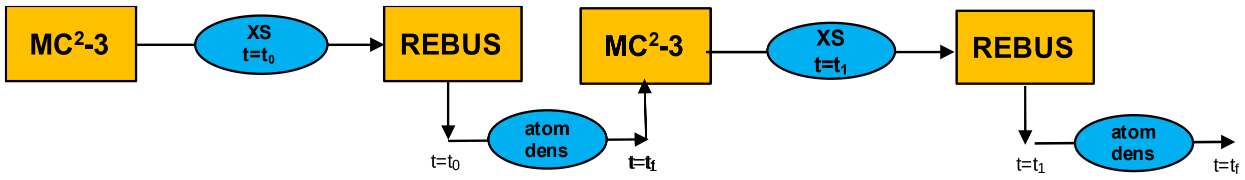


Figure 3-4. Multi-step depletion procedure implemented in PyARC.

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 MC$^2$-3 at beginning of equilibrium cycle, and the equilibrium search calculation performed with REBUS-3, as illustrated in Figure 3-5.

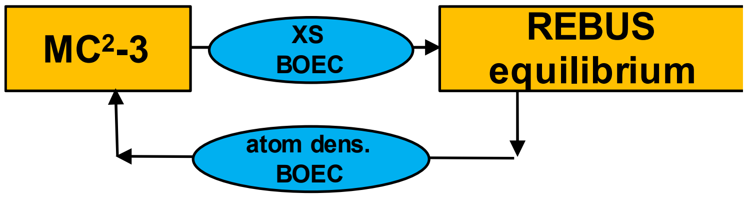


Figure 3-5. Equilibrium cross-section iteration procedure implemented in PyARC.

The explicit fuel management capability of REBUS is 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.

The REBUS-3 code relies on simplified decay chain tracking “only” ~200 isotopes. For higher fidelity depletion calculations that are typically needed for decay heat simulations, the ORIGEN-S code can be used, as discussed in Section 3.7.

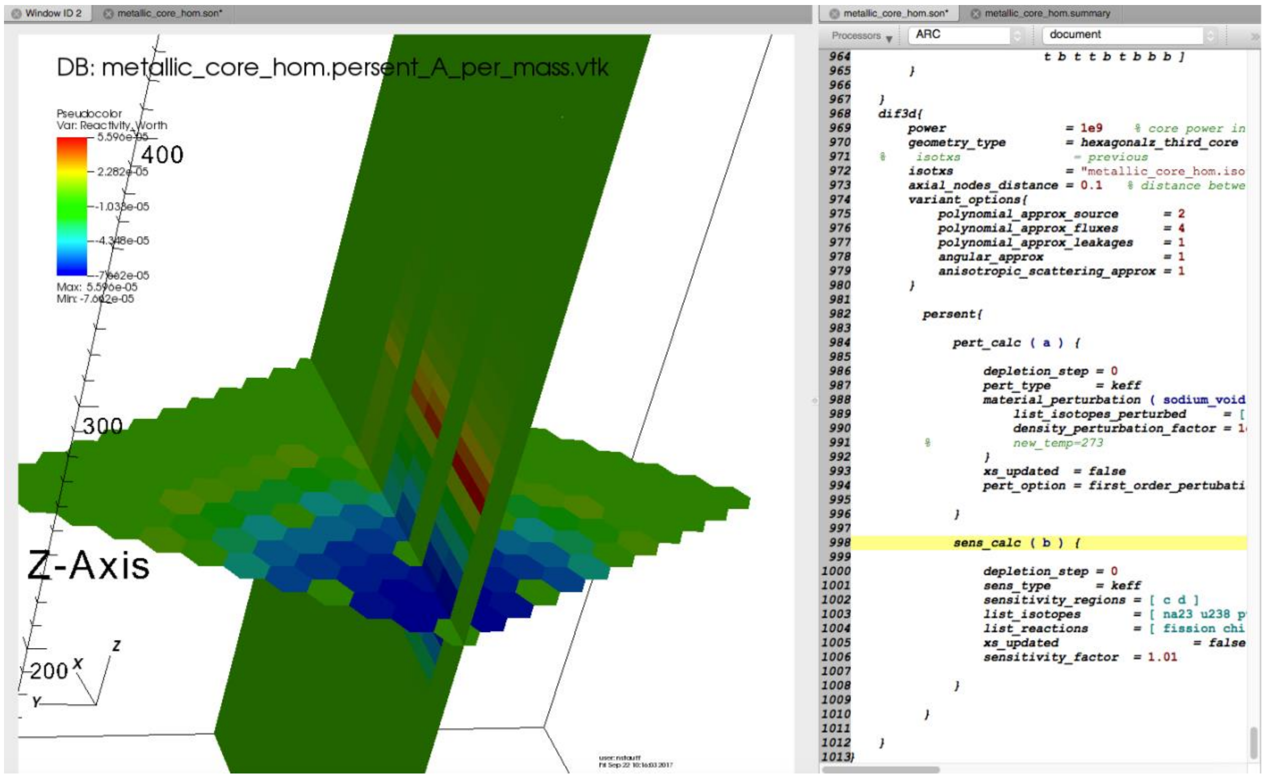
## *3.4 PERSENT [10]*

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 [28], 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 [17]: 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.

Perturbation calculations can 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.

New in V1.1.0: CovMat Utility

This utility has been developed in FY-2020 to streamline generation of covariance matrix of nuclear data uncertainties on reactivity coefficients, which is used for uncertainty propagation through transient simulations [29]. The workflow implemented in this utility is detailed in Appendix A.

## *3.5 GAMSOR [11]*

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 MC$^2$-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 is fully integrated within the PyARC interface and Sample #7 provides 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.

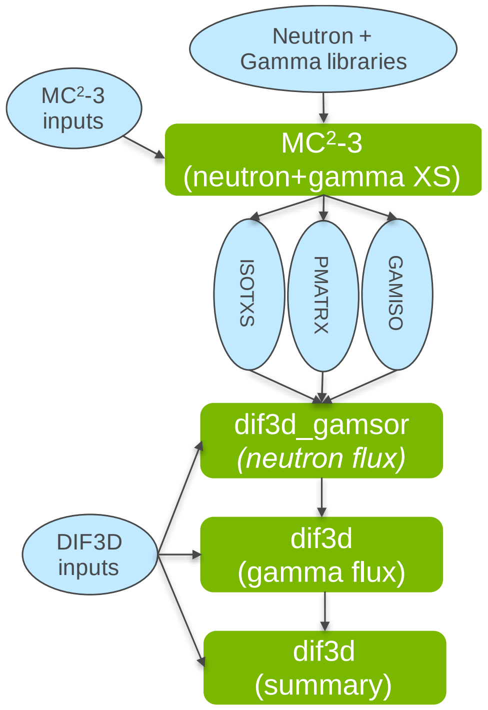
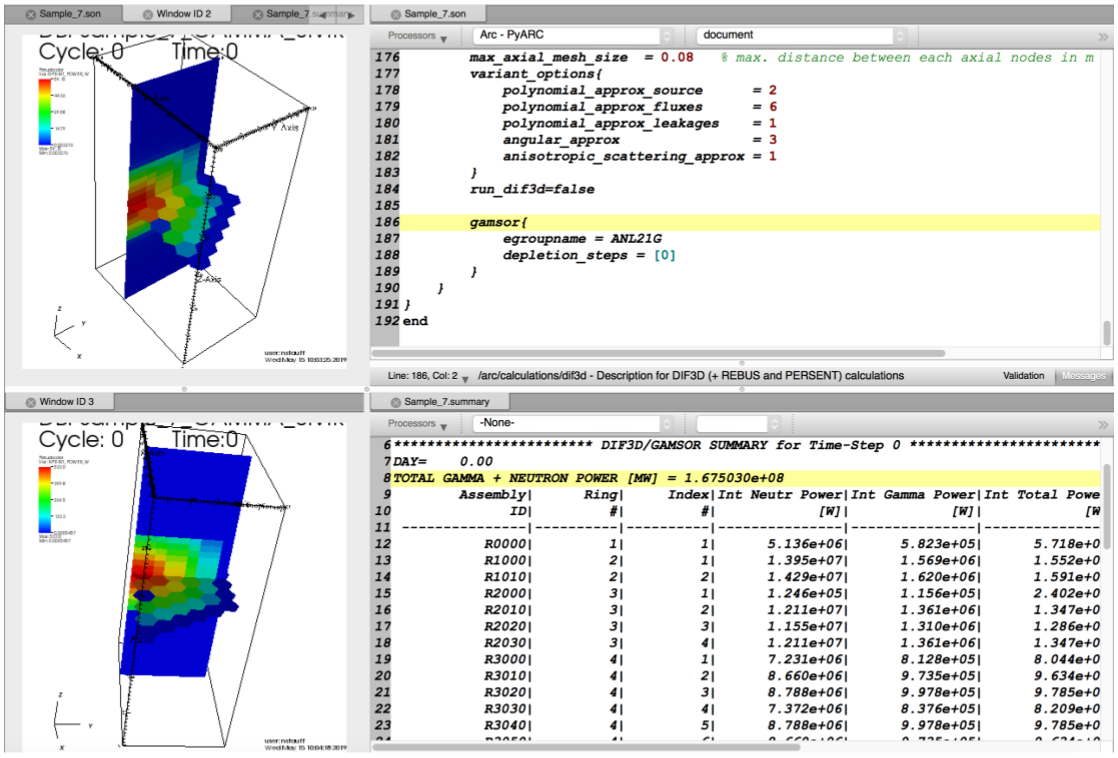


Figure 3-7. GAMSOR workflow implemented in PyARC.



```
        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
```

```
*********************** 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

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. The PROTEUS codes were integrated into the NEAMS Workbench interface 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:

- *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.

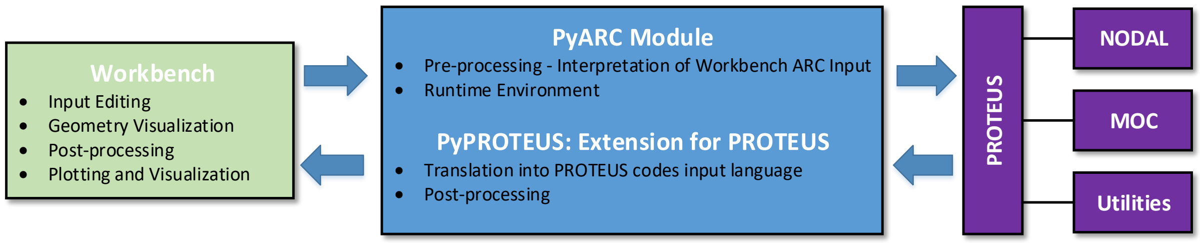

Workbench
Input Editing
Geometry Visualization
Post-processing
Plotting and Visualization
PyARC Module
Pre-processing - Interpretation of Workbench ARC Input
Runtime Environment
PyPROTEUS: Extension for PROTEUS
Translation into PROTEUS codes input language
Post-processing
PROTEUS
NODAL
MOC
Utilities

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

### 3.6.1 *PROTEUS-NODAL [12]*

The PROTEUS-NODAL code is a nodal transport solver based on homogenized assemblies that provides a conventional fidelity level in a consistent PROTEUS code framework. 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).

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.

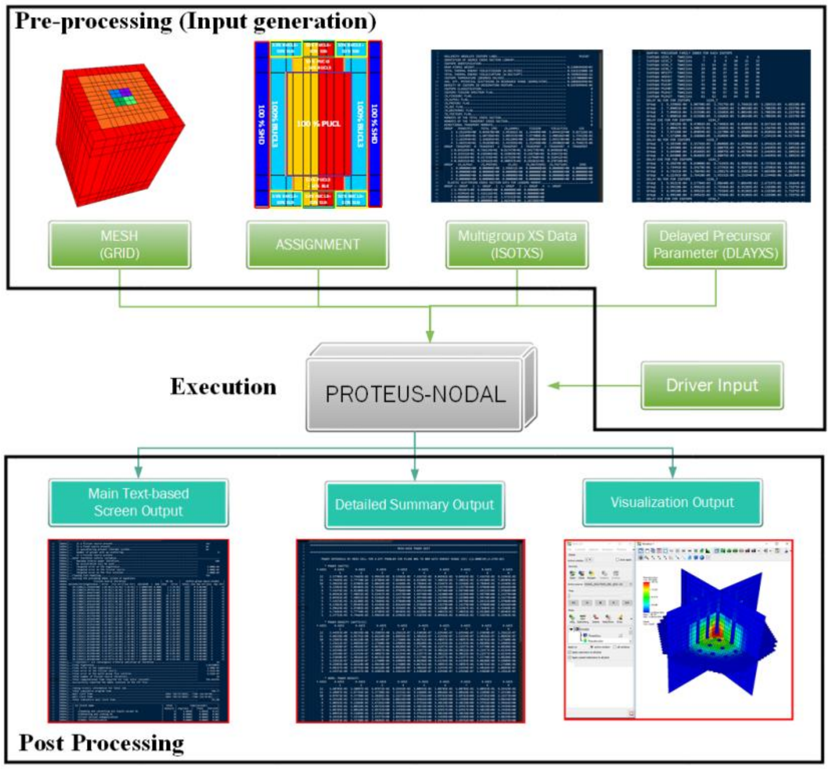

Pre-processing (Input generation)
MESH (GRID)
ASSIGNMENT
Multigroup XS Data (ISOTXS)
Delayed Precursor Parameter (DLAYXS)
Execution
PROTEUS-NODAL
Driver Input
Main Text-based Screen Output
Detailed Summary Output
Visualization Output
Post Processing

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

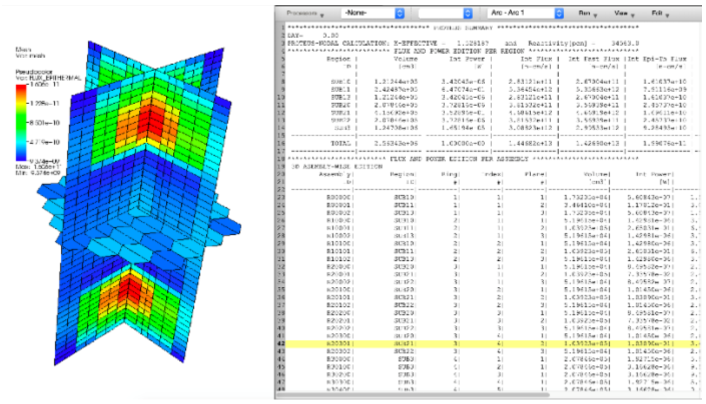
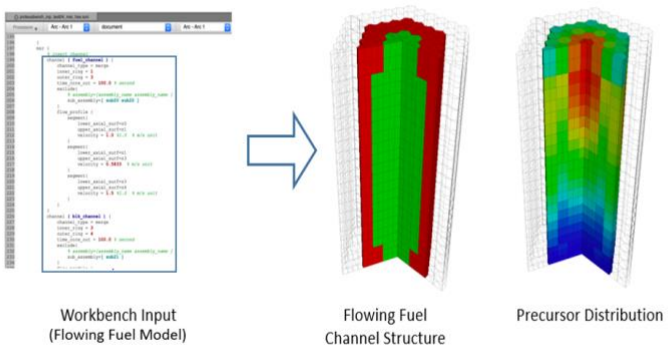


Figure 3-11. Example of post-processing for PROTEUS-NODAL: visualization of flux map (left) and assembly-wise summary table (right).

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 streamlined within the execution logic of PROTEUS-NODAL to provide delayed neutron precursor parameters. A sample input #13 demonstrating the MSR modeling was developed within the released tutorial.



```
    }
    arc {
      % insert channel
      channel ( fuel_channel ) {
        channel_type = merge
        inner_ring = 1
        outer_ring = 3
        time_zone_out = 100.0 % second
        exclude{
          % assembly=(assembly_name assembly_name )
          sub_assembly=( [illegible] [illegible] )
        }
        flow_profile {
          segment{
            lower_axial_surf=s0
            upper_axial_surf=s1
            velocity = 1.0 [illegible] % 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 = 1.5 [illegible] % m/s unit
          }
        }
      }
      channel ( blk_channel ) {
        channel_type = merge
        inner_ring = 3
        outer_ring = 4
        time_zone_out = 100.0 % second
        exclude{
          % assembly=(assembly_name assembly_name )
          sub_assembly=( [illegible] )
        }
```

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

### *3.6.2 PROTEUS-MOC [13]*

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. 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 [30]. Alternatively, the ANL mesh toolkit [31] 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.

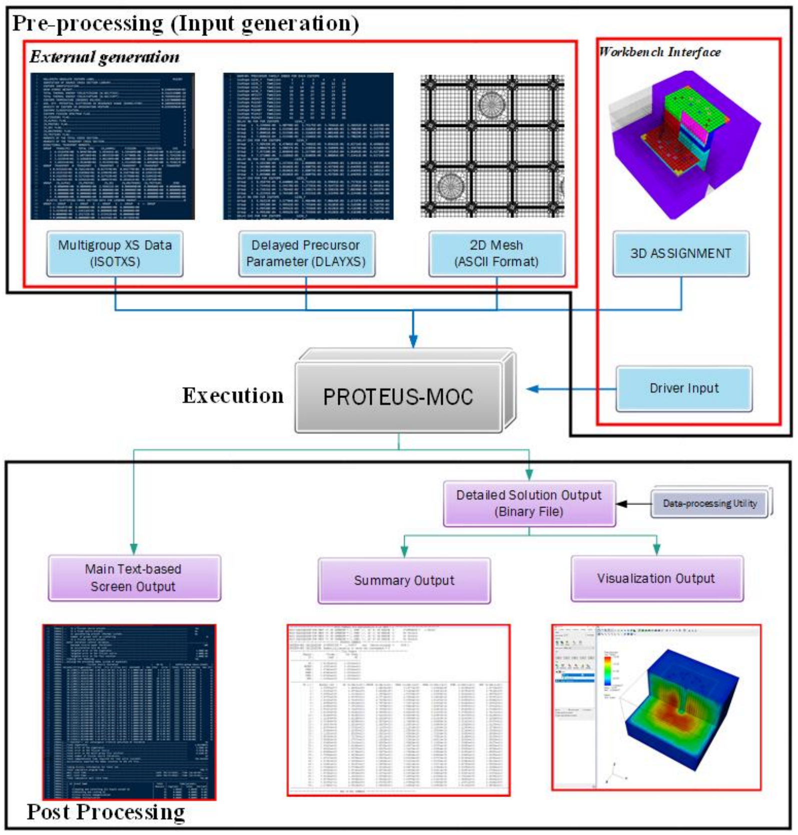


Figure 3-13. PROTEUS-MOC workflow implemented in PyARC.

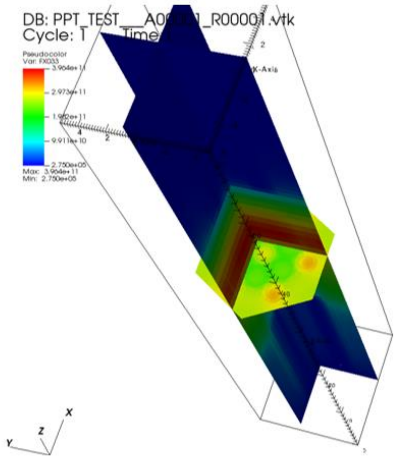
For the transient calculation mode, the additional input keywords are available in the Workbench interface to define perturbations of materials and temperatures for 3D geometrical regions of interest as a function of time. Based upon these user-defined transient descriptions, the pre-process logic can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

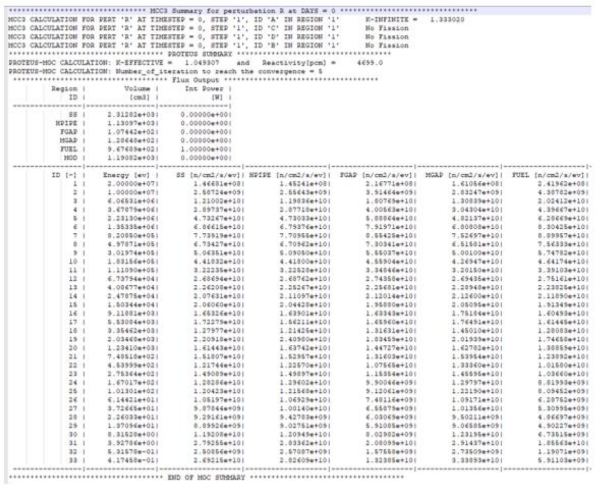
PROTEUS-MOC is executed via the runtime environment of PyARC. Once the calculation is completed, the PROTEUS-MOC code produces two basic types of outputs as:

- *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|        3.22828e+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|        2.25267e+10|        2.25681e+10|        2.20948e+10|        2.23825e+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|        1.49089e+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|        6.28762e+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|        6.73515e+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 |     4.17468e-01|        2.69215e+10|        2.82609e+10|        1.32385e+10|        3.33853e+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).

## *3.7 ORIGEN-S [32]*

New in V1.1.0

The ORIGEN-S code deployed within the SCALE package is used by ARC users for detailed depletion and decay calculations. This is required to compute decay heat, detailed isotopic composition, radio-activity, neutron sources, etc. ORIGEN-S traces more than 1,300 nuclides by solving Bateman equations using pre-generated one-group neutron libraries. A coupling procedure was developed by ARC users to generate problem-dependent one-group cross sections and to reproduce the REBUS depletion simulation using ORIGEN-S. This procedure was integrated into PyARC in FY-2020 within the PyREBORS.py function. Figure 3-15 shows the coupling procedures for depletion calculations using ORIGEN-S along with the REBUS-3 physics code.

In this PyREBORS procedure, problem-dependent effective one-group cross sections are obtained from the depletion calculations using the REBUS-3 code. However, since the physics code handles only about tens or hundreds of nuclides in the depletion calculation, the available one-group cross-sections are limited to the nuclides that are modeled by the ARC codes (limited to the ~200 isotopes available in the MC$^2$-3 library). The one-group cross sections are generated by using the COUPLE code [32] in the coupling-procedure with the ORIGEN-S code. It is noted that the SCALE code package has multi-group libraries with 238, 200, 49, or 40 group structures for thermal and fast systems. Thus, the one-group cross sections is generated by condensing those libraries using the problem-dependent neutron spectra obtained from the depletion calculations using REBUS-3.

The user input of the new "*rebus_to_origens*" block in the PyARC is shown in Figure 3-16 (extracted from Sample #14 in the tutorial). The ORIGEN-S calculation re-produces the once-through burnup simulation modeled with REBUS in the PyARC sub-assemblies selected by the user. The rebus calculation can be skipped by providing a pre-generated REBUS output using the "*rebus_output_file*" option. The one-group XS (capture, fission and (n,2n)) computed by REBUS-3 will be used by ORIGEN-S for the isotopes specified by "*list_isotope_XS_transfer*". The other 1-gp XS (for other actinides, reactions, fission products, etc.) are computed based on a detailed flux structure that can be specified with an external file "*detailed_flux.isotxs*", or by using the REBUS-generated flux spectrum (option by default) - the flux is automatically being linearly interpolated by PyARC to match the ORIGEN structure selected. The ORIGEN-S irradiation is then computed based on the initial power or on the flux at each step in every sub-assembly selected (using the "*power_or_flux*" option). Following irradiation, decay calculations is completed using the cumulative steps specified in "*decay_cumul_steps*".

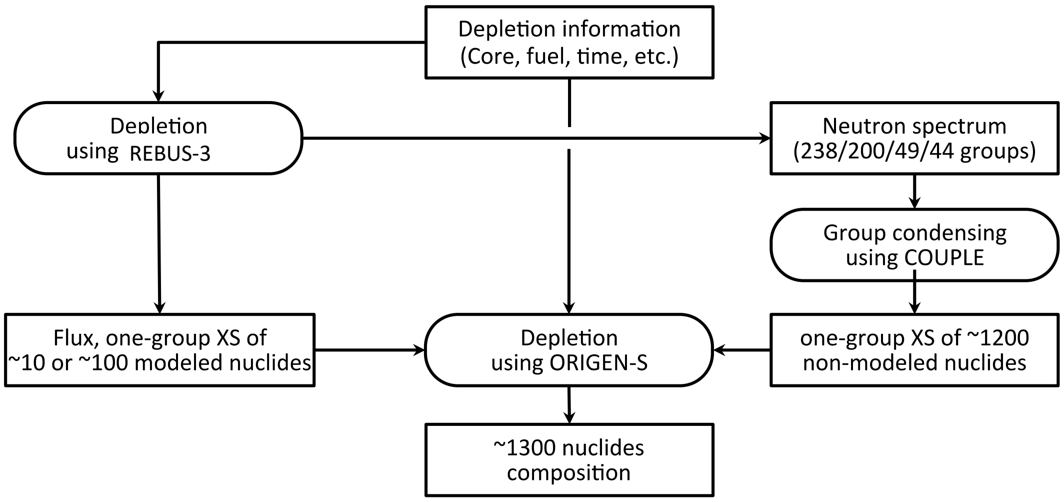
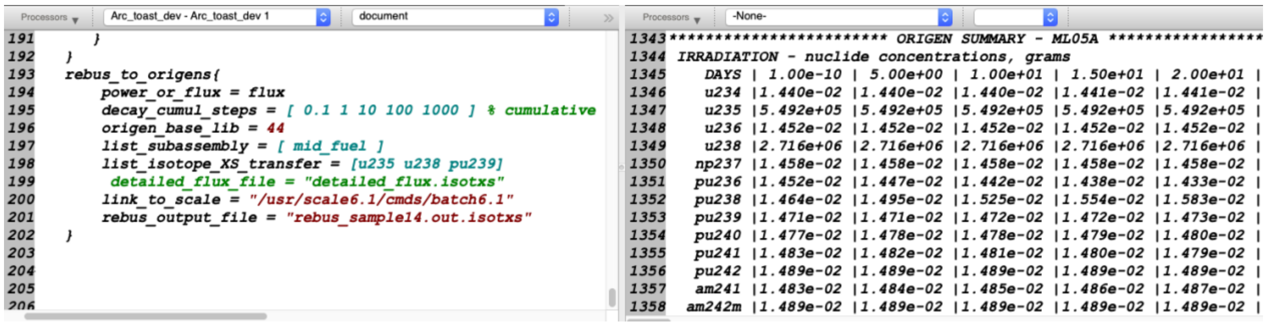


Figure 3-15. Workflow implemented of the REBUS-3 to ORIGEN-S coupling.



```
        }
    }
    rebus_to_origens{
        power_or_flux = flux
        decay_cumul_steps = [ 0.1 1 10 100 1000 ] % cumulative
        origen_base_lib = 44
        list_subassembly = [ mid_fuel ]
        list_isotope_XS_transfer = [u235 u238 pu239]
         detailed_flux_file = "detailed_flux.isotxs"
        link_to_scale = "/usr/scale6.1/cmds/batch6.1"
        rebus_output_file = "rebus_sample14.out.isotxs"
    }
```

```
************************* ORIGEN SUMMARY - ML05A *****************
 IRRADIATION - nuclide concentrations, grams
     DAYS |  1.00e-10 |  5.00e+00 |  1.00e+01 |  1.50e+01 |  2.00e+01 |
     u234 |1.440e-02 |1.440e-02 |1.440e-02 |1.441e-02 |1.441e-02 |
     u235 |5.492e+05 |5.492e+05 |5.492e+05 |5.492e+05 |5.492e+05 |
     u236 |1.452e-02 |1.452e-02 |1.452e-02 |1.452e-02 |1.452e-02 |
     u238 |2.716e+06 |2.716e+06 |2.716e+06 |2.716e+06 |2.716e+06 |
    np237 |1.458e-02 |1.458e-02 |1.458e-02 |1.458e-02 |1.458e-02 |
    pu236 |1.452e-02 |1.447e-02 |1.442e-02 |1.438e-02 |1.433e-02 |
    pu238 |1.464e-02 |1.495e-02 |1.525e-02 |1.554e-02 |1.583e-02 |
    pu239 |1.471e-02 |1.471e-02 |1.472e-02 |1.472e-02 |1.473e-02 |
    pu240 |1.477e-02 |1.478e-02 |1.478e-02 |1.479e-02 |1.480e-02 |
    pu241 |1.483e-02 |1.482e-02 |1.481e-02 |1.480e-02 |1.479e-02 |
    pu242 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |
    am241 |1.483e-02 |1.484e-02 |1.485e-02 |1.486e-02 |1.487e-02 |
   am242m |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |1.489e-02 |
```

Figure 3-16. Example of coupled depletion PyARC input and summary output.

The ORIGEN input and output are provided back in the "**.zip/origen_XXXX.**", and the main results are extracted in the ".summary" file as shown in Figure 3-16. The results extracted are the nuclide concentration for all the actinides concentrations throughout irradiation and decay, together with their contribution to the total radioactivity and decay heat computed after discharge throughout decay simulation.

Important Notes:

- Only the ORIGEN-S version from the Scale 6.1 package was tested.
- The depletion implemented only represent the initial loading of heavy nuclei, while activation products are not tracked yet.
- The default branching ratio of Am-241 are representative of a fast neutron spectrum.
- The ORIGEN-S irradiation is only used to re-produce once-through depletion calculation in a sub-assembly – it is currently not suitable to represent equilibrium search and assembly shuffling problems.

This methodology coupling REBUS-3 to ORIGEN-S is demonstrated in Appendix B comparing the masses of the main heavy nuclides of the discharged fuel compositions provided by REBUS-3 to the detailed composition generated by ORIGEN-S. Future work should include addressing some of the limitations of the current implementation above-discussed.

# 4 Conclusions and Future Work

This report details the status of the ARC and NEAMS codes capabilities integrated into the NEAMS Workbench. 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, 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, through PROTEUS integration within the same input logic as the legacy ARC codes.

In FY-2020, effort focused on integrating the REBUS to ORIGEN-S coupling capability for detailed irradiation calculations, and the CovMat utility to generate correlations within nuclear data uncertainties on reactivity feedbacks. Those capabilities were integrated as requested by PyARC users.  Various minor improvements were completed to enable additional modeling options and to respond to user requests.

The ARC and NEAMS codes are currently used at ANL, Westinghouse, INL, and NCSU through the Workbench by nuclear engineers for LFR, MSR, micro-reactor, and SFR core design analyses [22], [33], [34], [35], [36]. Several in-persons training were organized in FY-2020 to ANL, Westinghouse, and at the NRC.

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] N. Stauff, P. Lartaud, Y. S. Jung, K. Zeng, J. Hou, "Status of the NEAMS and ARC neutronic fast reactor tools integration to the NEAMS Workbench," ANL/NEAMS-19/1, Sept. 30, (2019).

[6] 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.

[7] 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).

[8] 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).

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

[10] M. A. Smith, C. Adams, W. S. Yang, E. E. Lewis, "VARI3D & PERSENT: Perturbation and Sensitivity Analysis," Argonne National Laboratory, ANL/NE-13/8 Rev. 3, Apr 30 (2020).

[11] 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.

[12] 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).

[13] 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).

[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] 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).

[16] 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).

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

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

[19] 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).

[20] Kaiyue Zeng, Nicolas E. Stauff, Jason Hou, T. K. Kim "Development of multi-objective core optimization framework and application to sodium-cooled fast test reactors," Progress in Nuclear Energy, Vol 120, February (2020) 103184.

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

[22] 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).

[23] 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.

[24] 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).

[25] 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).

[26] 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).

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

[28] 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).

[29] Nicolas E. Stauff, K. Zeng, G. Zhang, G. Aliberti, J. Hu, T. Fanning, and T. K. Kim, "Uncertainty quantification of ABR transient safety analysis – nuclear data uncertainties," BEPU 2018, May 13-19, Lucca, Italy (2018).

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

[31] 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).

[32] "Scale: A Comprehensive Modeling and Simulation Suite for Nuclear Safety Analysis and Design" ORNL/TM-2005/39 Version 6.1 (June 2011).

[33] 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).

[34] 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).

[35] 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

[36] 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).

[37] 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).

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

# Appendix A : Description of the CovMat Script

The CovMat script is developed to compute the covariance matrix used for propagating nuclear data uncertainties on reactivity feedback coefficients through transients analyses. Detailed description of the methods employed is provided in [29]. This appendix summarizes the method used for computing this matrix, and details the workflow implemented in the CovMat script.

## *A.1 Method Description*

The PERSENT code is used to calculate sensitivity coefficients on reactivity coefficients based on the forward and adjoint fluxes calculated by DIF3D. This sensitivity analysis is performed in 33 energy macro-groups using the generalized perturbation theory to calculate the sensitivity coefficients ($S_{j,x,p} = \frac{\partial k}{\partial \sigma_{j,x,p}} \frac{\partial \sigma_{j,x,p}}{k}$, with *p*, *x*, and *j* the indices representing the isotope, the cross-section type, and energy group, respectively) for k-effective (*k*) or other reactivity effects, to variations of a cross-section $\sigma$. The nuclear data uncertainties associated with the ENDF/B-VII library [37] are provided in 33 energy macro-groups with the covariance matrix COMMARA-2.0 [38]. The uncertainties $I_i^2$ of the neutronic feedback coefficients *i* in each core region (Doppler effect, density expansion coefficients, control rod worth…) are obtained for each isotope *p*, reaction *x*, and energy group *j* by applying the "sandwich equation": $I_i^2 = S_i^T D S_i$, where *D* is the COMMARA-2.0 covariance matrix and $S_i$ is the sensitivity matrix associated to this reactivity coefficient.

Uncertainties from nuclear data are propagated through transients by perturbing the neutronic parameters based on a stochastic sampling approach as detailed in [29]. However, such methodology usually assumes independent uncertainties. This is a poor assumption in the case of nuclear data uncertainty propagation since the uncertainties of different feedback coefficients can be traced back to a few uncertain cross-sections: a change in one of these cross sections affects some of the reactivity coefficients involved in a similar way. Independent propagation will end up under-estimating the total uncertainty as shown in [29] (due to forced-independent parameters "cancelling" each other). Consequently, the method developed to perform nuclear data uncertainty propagation was developed to allow taking into account two types of correlations within the uncertainties of different reactions:

- <u>spatial correlations</u> between uncertainties in different core regions (for instance, correlations between uncertainties on the sodium density coefficient in the driver fuel and in the gas plenum)
- <u>reaction-wise correlations</u> between the uncertainties of different reactivity coefficients (for instance, correlation between the uncertainties on the Doppler and sodium density coefficients)

The objective of the approach described here is to enable propagating the uncertainties through transients in a dependent way with the definition of the correlation and covariance matrices. In equation (1), $I^2$ represents the total variance of the system over all the *i=1…n* reactivity coefficients over the different core regions evaluated. If *i* represents a feedback coefficient in different spatial regions, then $I^2$ indicates the total uncertainty of the reactor for that specific coefficient. If *i* represents different feedback coefficient (e.g. Doppler, density coefficients, …), then $I^2$ does not have a physical meaning but represents mathematically the total contribution of the uncertainties from these different coefficients. The correlation matrix *C* is defined by the following Equation (1) and (2) with the associated correlation coefficients $\rho_{i,j}$ between the uncertainty of reactions *i* and *j*. This correlation matrix *C* is symmetric because *D* is symmetric.

$$I^2 = \sum_i S_i^T D \sum_i S_i = \sum_i I_i^2 + 2 \sum_i \sum_{j<i} S_i^T D S_j = (I_1 \cdots I_n) C \begin{pmatrix} I_1 \\ \vdots \\ I_n \end{pmatrix} \\ = \sum_i I_i^2 + 2 \sum_{i,j\ (i\neq j)} I_j I_i \rho_{i,j} \tag{1}$$

$$\rho_{i,j} = \frac{S_i^T D S_j}{I_j I_i} = \frac{S_i^T D S_j}{\sqrt{S_i^T D S_i} \cdot \sqrt{S_j^T D S_j}} \ ; \quad C = \begin{pmatrix} 1 & \cdots & \rho_{1,n} \\ \vdots & 1 & \vdots \\ \rho_{n,1} & \cdots & 1 \end{pmatrix} \tag{2}$$

The correlations inform on the dependence relationship of the different uncertainties on the reactivity coefficients. However, for propagating dependent uncertainties, one requires a covariance matrix. The co-variance term between reactions *i* and *j* that appears in Equation (1) may also be written with equation (3). The associated covariance matrix *Σ* for the uncertainties is described in Equation (4). By definition, the covariance matrix *Σ* should be symmetric and positive definite so it should allow Cholesky decomposition. This decomposition of the *Σ* matrix into its lower triangular matrix *L* using the Cholesky equation described in Equation (5), allows generating a random vector *x* following a *N(0, Σ)* distribution (normal distribution, mean is zero, and covariance matrix is *Σ*). Starting from vector *z* generated by uncertainty propagation code (such as Dakota [18]) following a *N(0, I)* distribution (normal distribution, mean is zero and covariance matrix is identity matrix), the vector *x* is estimated according to Equation (6) and is used to perturb reactivity coefficient in transient simulations.

$$For\ i \neq j: cov(i,j) = I_j I_i \rho_{i,j} = S_i^T D S_j \tag{3}$$

$$\Sigma = \begin{pmatrix} I_1^2 & \cdots & cov(1,n) \\ \vdots & I_i^2 & \vdots \\ cov(n,1) & \cdots & I_n^2 \end{pmatrix} = \begin{pmatrix} S_1^T D S_1 & \cdots & S_1^T D S_n \\ \vdots & S_i^T D S_i & \vdots \\ S_1^T D S_n & \cdots & S_n^T D S_n \end{pmatrix} \tag{4}$$

$$\Sigma = LL^T \tag{5}$$

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = L \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \tag{6}$$

### *A.2 Workflow Implemented in CovMat Script*

This section illustrates the different steps followed in the CovMat script to generate the correlation coefficients. This example includes a reactor that consists of three reactor zones XXXX, YYYY, and ZZZZ and 2 different reactions “F” and “C”. The covariance matrix that needs to be filled out is shown in Table 1.

In this example, subscript "F" represents fuel density perturbation and subscript "C" represents coolant density perturbation. Subscript "K" refers to the reference case. Subscripts XXXX, YYYY, and ZZZZ refer to reactivity worth for different reactor zones. The sensitivity of $S_{K,XXXX}$ represents the sensitivity on un-perturbed K-eff calculated with PERSENT (through PyARC/sens_calc with the option "print_by_zone=true"). The sensitivity of $S_{F,XXXX}$ represents the sensitivity on perturbed K-eff calculated with PERSENT (through PyARC/sens_calc with the option "print_by_zone=true" and "sens_type = keff"). An example of input used to generate the sensitivities is shown in the tutorial. Those sensitivities need to be computed prior to running the CovMat utility, and stored appropriately with their "K/*XXXX_sens_K.sens*" file names.

Table 1. Example of correlation matrix.

| Reactions | | F | F | C |
|---|---|---|---|---|
| | regions | XXXX | YYYY | ZZZZ |
| F | XXXX | $S^T_{KF,XXXX}DS_{KF,XXXX}$ | $S^T_{KF,XXXX}DS_{KF,YYYY}$ | $S^T_{KF,XXXX}DS_{Kc,ZZZZ}$ |
| F | YYYY | $S^T_{KF,YYYY}DS_{KF,XXXX}$ | $S^T_{KF,YYYY}DS_{KF,YYYY}$ | $S^T_{KF,YYYY}DS_{Kc,ZZZZ}$ |
| C | ZZZZ | $S^T_{KC,ZZZZ}DS_{KF,XXXX}$ | $S^T_{KC,ZZZZ}DS_{KF,YYYY}$ | $S^T_{KC,ZZZZ}DS_{Kc,ZZZZ}$ |

The first step of the CovMat utility computes the sensitivities on the region-wise reactivity coefficients. For instance, $S_{KF,XXXX}$ is the sensitivity in region XXXX for reactivity coefficient F: δρ(F). The $S_{KF,XXXX}$ sensitivity coefficient and its associated variance ($S^T_{KF,XXXX}DS_{KF,XXXX}$) is calculated with PERSENT using the following input. This calculation will also generate the "*XXXX_sens_KF.sens*" files that will be used in the following step.

```
SENSITIVITY_FILE      KEFF_K          XXXX_sens_K.sens
SENSITIVITY_FILE      KEFF_F          XXXX_sens_F.sens
SENSITIVITY_DIFF      DRHO_F_XXXX     KEFF_K              KEFF_F
SENSITIVITY_EDITS     DRHO_F_XXXX     XXXX_sens_KF.sens
SENSITIVITY_DOUQ      DRHO_F_XXXX     YES YES YES
COMMARA_INPUT         COMMARA.inp
COMMARA_IGNORE        YES
MISSINGDATA
COMMARA_ISOTOPE       C____7          CARBON
        ⋮                 ⋮               ⋮                   ⋮
```

The second step consists in computing the co-variance parameters, such as $S^T_{KF,YYYY}DS_{KF,XXXX}$ using the "bilinear" calculation from PERSENT [10]. This calculation relies on the sensitivity files for the reactivity coefficients generated within the first step. An example of PERSENT input used to generate $S_{KF,YYYY}^{T} D S_{KF,XXXX}$ is provided thereafter.

```
SENSITIVITY_FILE       DRHO_F_XXXX          XXXX_sens_KF.sens
SENSITIVITY_FILE       DRHO_F_YYYY          YYYY_sens_KF.sens
BILINEAR               COV_XXXX_YYYY_FF     DRHO_F_XXXX           DRHO_F_YYYY
SENSITIVITY_DOUQ       COV_XXXX_YYYY_FF     YES YES YES
COMMARA_INPUT          COMMARA.inp
COMMARA_IGNORE         YES
MISSINGDATA
COMMARA_ISOTOPE        C____7               CARBON
        ⋮                    ⋮                     ⋮                   ⋮
```

Through these two steps, one can build the full covariance matrix. Since the covariance matrix generated can be extremely large, the PERSENT calculations can be parallelized and run on several CPUs. The option is also provided in the CovMax to include only the reaction-wise or spatial correlations.

The final step is the computation of the Cholesky decomposition (equation (5)) using built-in Python procedures. It should be mentioned that the covariance matrix *Σ* calculated with PERSENT may not always be positive and symmetric due to rounding approximations and to not fully converged sensitivity calculations. Consequently, a procedure was developed that forces the symmetry of the matrix, calculates its eigenvalues, replaces negative ones with small positive ones ($10^{-20}$), and re-constructs the Σ covariance matrix.

The tutorial developed in PyARC provides an example test problem, and the associated output generated through this CovMat utility script.

## Appendix B : Verification Test Of REBUS-3 To ORIGEN-S Coupling

The verification test discussed in this Appendix is based on the Tutorial Sample #14 example with a fast-spectrum reactor modeling exercise. In this analysis, the higher detailed decay chain using Lumped Fission Products was employed. The calculation card was also modified by specifying higher core power (1,000 MW) through longer and more numerous time-steps (5 time-steps of 200 EFPDs) in order to reach higher discharged burnup of ~100 GWd/t.

The discharged heavy nuclei isotopic compositions are displayed in Table 2 comparing the REBUS-3 and REBUS-to-ORIGEN-S simulations. The ORIGEN-S simulations use the 1-group cross-sections computed at every time-step with REBUS-3 for all the heavy actinides listed in Table 2. Two options of the "rebus_to_origens" calculations are compared: the OS(Power) uses the initial power density calculated by REBUS-3 in the sub-assembly, while the OS(Flux) uses the total flux calculated by REBUS-3 at each time-steps.

Table 2. Comparison of discharged mass between REBUS-3 and different REBUS-to-ORIGEN-S coupled approaches.

| (kg) | REBUS BOL | REBUS EOL | OS(Power) EOL | Diff (R-OS)/R | OS(Flux) EOL | Diff (R-OS)/R |
|---|---|---|---|---|---|---|
| U234 | 1.44E-05 | 1.64E-01 | 1.66E-01 | 1.1% | 1.66E-01 | 1.2% |
| U235 | 5.49E+02 | 2.54E+02 | 2.54E+02 | 0.0% | 2.54E+02 | 0.0% |
| U236 | 1.45E-05 | 5.51E+01 | 5.52E+01 | 0.2% | 5.52E+01 | 0.3% |
| U238 | 2.72E+03 | 2.44E+03 | 2.44E+03 | 0.0% | 2.44E+03 | 0.0% |
| Np237 | 1.46E-05 | 5.31E+00 | 5.21E+00 | -1.9% | 5.22E+00 | -1.8% |
| Np238 | 1.46E-05 | 1.23E-02 | 1.21E-02 | -1.2% | 1.21E-02 | -1.5% |
| Np239 | 1.47E-05 | 8.30E-01 | 8.36E-01 | 0.7% | 8.32E-01 | 0.2% |
| Pu238 | 1.46E-05 | 1.21E+00 | 1.18E+00 | -2.6% | 1.18E+00 | -2.5% |
| Pu239 | 1.47E-05 | 1.46E+02 | 1.46E+02 | -0.1% | 1.46E+02 | 0.0% |
| Pu240 | 1.48E-05 | 1.37E+01 | 1.37E+01 | -0.1% | 1.37E+01 | 0.0% |
| Pu241 | 1.48E-05 | 8.40E-01 | 8.37E-01 | -0.3% | 8.39E-01 | -0.1% |
| Pu242 | 1.49E-05 | 4.86E-02 | 4.83E-02 | -0.6% | 4.85E-02 | -0.2% |
| Am241 | 1.48E-05 | 2.44E-02 | 2.49E-02 | 2.3% | 2.50E-02 | 2.5% |
| Am242m | 1.49E-05 | 7.56E-04 | 7.71E-04 | 2.1% | 7.74E-04 | 2.4% |
| Am243 | 1.50E-05 | 2.31E-03 | 2.29E-03 | -0.9% | 2.30E-03 | -0.4% |
| Cm242 | 1.49E-05 | 1.87E-03 | 1.91E-03 | 2.0% | 1.92E-03 | 2.2% |
| Cm243 | 1.50E-05 | 4.87E-05 | 4.97E-05 | 1.9% | 4.99E-05 | 2.3% |
| Cm244 | 1.50E-05 | 3.23E-04 | 3.19E-04 | -1.2% | 3.21E-04 | -0.7% |
| Cm245 | 1.51E-05 | 2.25E-05 | 2.23E-05 | -1.3% | 2.23E-05 | -0.9% |
| Cm246 | 1.51E-05 | 1.35E-05 | 1.35E-05 | -0.1% | 1.35E-05 | -0.1% |
| Total ACT | 3.27E+03 | 2.92E+03 | 2.92E+03 | 0.0% | 2.92E+03 | 0.0% |
| Total FP | 5.06E-05 | 3.47E+02 | 3.47E+02 | 0.1% | 3.48E+02 | 0.2% |

A very good agreement is observed with less than 0.5% of discrepancy for the most important nuclides such as U-238 and Pu-239. Discrepancies obtained with ORIGEN-S are less than 2% and are mostly observed for Minor Actinides. The discrepancies in the discharged mass of the main heavy nuclei are acceptable and can be explained by differences in the decay chain and the one-group cross-sections of non-major nuclides and reactions.

Both ORIGEN-S irradiation options (using Flux and Power renormalization) provide very close results in terms of discharged actinide composition. In fact, a very small impact is also observed in Table 3 on the radio-activity and decay heat computed after discharge. To conclude, these results confirm proper implementation of the REBUS-3 to ORIGEN-S coupling strategy and its different renormalization approaches.

Table 3. Comparison of decay results between different REBUS-to-ORIGEN coupled approaches.

| **Depletion - Years** | **0.1** | **1.0** | **10** | **100** | **1,000** | **10,000** | **1.E+5** |
|---|---|---|---|---|---|---|---|
| Irradiation Method | **Radioactivity, curies** | | | | | | |
| OS(Flux) Total ACT | 4.00E+8 | 3.45E+5 | 1.14E+5 | 8.33E+4 | 2.35E+4 | 1.23E+4 | 7.96E+3 |
| OS(Flux) Total FP | 1.63E+9 | 1.06E+8 | 2.17E+7 | 3.30E+6 | 3.65E+5 | 2.14E+2 | 1.99E+2 |
| OS(Power) Total ACT | 4.02E+8 | 3.46E+5 | 1.14E+5 | 8.33E+4 | 2.35E+4 | 1.23E+4 | 7.97E+3 |
| OS(Power) Total FP | 1.64E+9 | 1.07E+8 | 2.17E+7 | 3.30E+6 | 3.65E+5 | 2.14E+2 | 1.99E+2 |
| *Diff (F-P)/F* | *-0.7%* | *-0.3%* | *0.0%* | *0.0%* | *0.0%* | *-0.1%* | *0.0%* |
| Irradiation Method | **Thermal power, watts** | | | | | | |
| OS(Flux) Total ACT | 1.03E+6 | 1.62E+3 | 1.03E+03 | 9.66E+02 | 7.32E+02 | 3.84E+02 | 2.47E+02 |
| OS(Flux) Total FP | 2.02E+7 | 4.16E+5 | 8.25E+04 | 9.12E+03 | 1.01E+03 | 1.01E-01 | 9.74E-02 |
| OS(Power) Total ACT | 1.04E+6 | 1.62E+3 | 1.04E+03 | 9.66E+02 | 7.32E+02 | 3.84E+02 | 2.47E+02 |
| OS(Power) Total FP | 2.04E+7 | 4.17E+5 | 8.25E+04 | 9.12E+03 | 1.01E+03 | 1.01E-01 | 9.74E-02 |
| *Diff (F-P)/F* | *-0.6%* | *-0.2%* | *-0.1%* | *0.0%* | *0.0%* | *0.0%* | *0.0%* |

**Nuclear Science and Engineering Division**
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC