SoftwareX 22 (2023) 101345

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MOOSE Stochastic Tools: A module for performing parallel, memory-efficient in situ stochastic simulations

Andrew E. Slaughter [a], Zachary M. Prince [a,*], Peter German [a], Ian Halvic [a,b], Wen Jiang [a], Benjamin W. Spencer [a], Somayajulu L.N. Dhulipala [a], Derek R. Gaston [a]

[a] Idaho National Laboratory, Idaho Falls, ID 83415, United States of America
[b] Texas A&M University, College Station, TX 77840, United States of America

ARTICLE INFO

Article history:
Received 4 January 2022
Received in revised form 31 May 2022
Accepted 21 February 2023

Keywords:
Stochastic
Parallel
Multiphysics
MOOSE

ABSTRACT

Stochastic simulations are ubiquitous across scientific disciplines. The Multiphysics Object-Oriented Simulation Environment (MOOSE) includes an optional module – stochastic tools – for implementing stochastic simulations. It implements an efficient and scalable scheme for performing stochastic analysis in memory. It can be used for building meta models to reduce the computational expense of multiphysics problems as well as perform analyses requiring up to millions of stochastic simulations. To illustrate, we have provided an example that trains a proper orthogonal decomposition reduced-basis model. The impact of the module is detailed by explaining how it is being used for failure analysis in nuclear fuel and reducing the computational burden via dynamic meta model training. The module is unique in that it provides the ability to use a single framework for simulations and stochastic analysis, especially for memory intensive problems and intrusive meta modeling methods.

Code metadata

Current code version	N/A (uses continuous stable branch)
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00007
Legal Code License	LGPL 2.1
Code versioning system used	Continuous stable branch with git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	C++17 compiler (GCC or Clang) Memory: 16GB+ Disk: 30GB+ OS: Mac OS 10.13+, Linux (POSIX) Deps: MPI, PETSc, libMesh
If available Link to developer documentation/manual	https://mooseframework.org/modules/stochastic_tools
Support email for questions	https://github.com/idaholab/moose/discussions

1. Motivation and significance

Stochastic simulations are of critical importance to a wide range of problems. Ref. [1] states that a sensitivity analysis "is a prerequisite for model building in any setting…" Within the nuclear energy sector, there exists a primal need for stochastic calculations for performing failure analyses of fuel and reactor structural components [2,3] as well as to complete a probability risk assessment for full reactor systems. For nuclear components, the probability of failure is often very low, thus the number of required simulations to reach a desired value of accuracy in the result can be quite large ($\sim 10^9$) [4]. With this problem in mind, the stochastic tools module within Multiphysics Object-Oriented Simulation Environment (MOOSE) was developed to focus on parallel scaling and memory management.

MOOSE is an open-source framework for building simulation tools that solve systems of coupled equations describing various

* Corresponding author.
E-mail address: zachary.prince@inl.gov (Zachary M. Prince).

https://doi.org/10.1016/j.softx.2023.101345

physical phenomena, and these simulations can execute sub-applications [5,6]. It also provides optional modules that can expand the functionality of the framework in various ways. The Stochastic Tools Module (MOOSE-STM), which is the focus of this paper, was created as a module, rather than as a stand-alone tool, to take advantage of features of the MOOSE framework and allow tight integration with any MOOSE-based application. This architecture also provides high performance, which is achieved by managing parallel execution, data, and memory internally, without the need for aggregation routines or file input/output operations.

The MOOSE-STM is not designed to replace existing general-purpose stochastic analysis packages, such as Dakota [7]; rather, its focus is to provide a tool capable of performing memory efficient, intrusive stochastic MOOSE-based simulations. Furthermore, there are multitudinous varieties of stochastic simulations based on the application; the MOOSE-STM focuses on relevant methods for uncertainty quantification (UQ), sensitivity analysis, failure assessment, and meta-model creation of deterministic systems, particularly multiphysics finite element models. The MOOSE-STM allows for stochastic simulations to be managed in memory and distributed in parallel to enable complex stochastic simulations, including the ability to manipulate all aspects of the simulation.

2. Software description

The MOOSE-STM is a part of MOOSE and thus follows the same design. It is composed of "systems" that provide extension points for defining simulation characteristics. Each system provides one or more C++ base classes that application developers inherit from to add functionality [5].

2.1. Software architecture

The MOOSE-STM, as illustrated in Fig. 1, is based around a driving (main) application that runs any number of sub-applications. This leverages pre-existing systems within MOOSE to execute sub-applications from a main application and transfer data to and from those sub-applications. These systems, known as the MultiApp and Transfer systems, are detailed in Ref. [5] and were primarily motivated by the needs of multiscale simulations (e.g., [6]). Communication between these applications can be one- or two-directional, as needed. The MultiApp system manages the allocation of parallel computing resources between the main application and the sub-applications.

While the MultiApp and Transfer systems were not originally developed for stochastic analysis, they provide many of the essential features needed for efficient stochastic analysis on high-performance computing platforms. In the MOOSE-STM, stochastic input (i.e., model perturbations driving the analysis) is generated in the main application, and the Transfer system is used to supply that data to the sub-applications, which are created using the MultiApp system. The stochastic quantities of interest (QoI) (i.e., relevant model outputs) from the sub-application are then transferred back to the main application for processing. This can include tasks as basic output, statistics calculations, or training reduced-order models. At all stages, the data is distributed across processors.

2.1.1. Distribution system

The Distribution system provides a common interface to define arbitrary probability distribution functions. It defines a `Distribution` base class, from which custom objects that define other distributions can be derived. This class contains three pure virtual methods required to be overridden that are available for use by other objects: `pdf`, `cdf`, and `quantile`.

2.1.2. Sampler system

The Sampler system provides a common interface to classes that define specific sampling schemes. It defines the `Sampler` base class, from which custom samplers are derived. These `Sampler` classes must generate a matrix of stochastic input; the virtual `computeSample` method within a user-defined object is responsible for producing values for given indices of the matrix. The content of the matrix is arbitrary, but generally each row represents a single model realization and the columns consist of values for each stochastic input.

`Sampler` objects provide utility functions for computing stochastic input that are accessible through an interface. Objects using this interface obtain a reference to `Sampler` objects and have access to the public application programming interface (API) of this class. Objects using the `Sampler`, as is discussed in Section 2.1.4, call this API to get data for stochastic analysis associated with the current processor without the need to compute the entire set of data.

2.1.3. Surrogate system

The Surrogate system allows for the creation of classes that train and evaluate meta-models (e.g. reduced-order models) to efficiently represent complex phenomena. This system defines two base classes. The first, `SurrogateTrainer` is designed to train these meta models. It operates as a producer of quantities to be used by the second base class of the system, `SurrogateModel`, for evaluating the models. The trainer object declares the model data to be computed and then computes these values, generally based on input from stochastic QoI. The model object can obtain the necessary data from an existing file produced by a trainer or directly from an instantiated trainer object. The later approach allows for the trainer to adjust the training data as a simulation progresses. Section 3 includes an example use of the Surrogate system to train a proper orthogonal decomposition (POD) reduced basis (RB) model.

2.1.4. MultiApp and transfer objects

As noted previously, the MOOSE-STM uses the MOOSE MultiApp and Transfer systems to execute sub-applications from a main application and transfer data between them. In the context of stochastic analysis, a sub-application represents a single simulation executed with stochastic input. The MOOSE-STM extends the `MultiApp` object to include several modes of operation, which are key features of the MOOSE-STM that allow for efficient memory management as detailed in Section 2.3. The module also includes `Transfer` objects specific for transferring data between the main and sub-applications, regardless of execution mode.

2.1.5. Statistics

A motivating factor for creating the MOOSE-STM was to perform stochastic simulations without the need for expensive input/output operations. Consistent with this goal, a means to compute statistics was created to process stochastic QoI data within the main application, using parallel distributed data. A custom `Reporter` object, the `StatisticsReporter` was created to compute statistics as well as confidence level intervals [8] from that data, without requiring that data to be written to or read from files. The underlying calculations utilize C++ templates that allow for specialization, thus allowing for the support of arbitrary QoI types.

2.2. Workflow

The following presents a typical workflow for performing stochastic simulations with the MOOSE-STM:

1. Build multiphysics model using MOOSE input(s) [5].

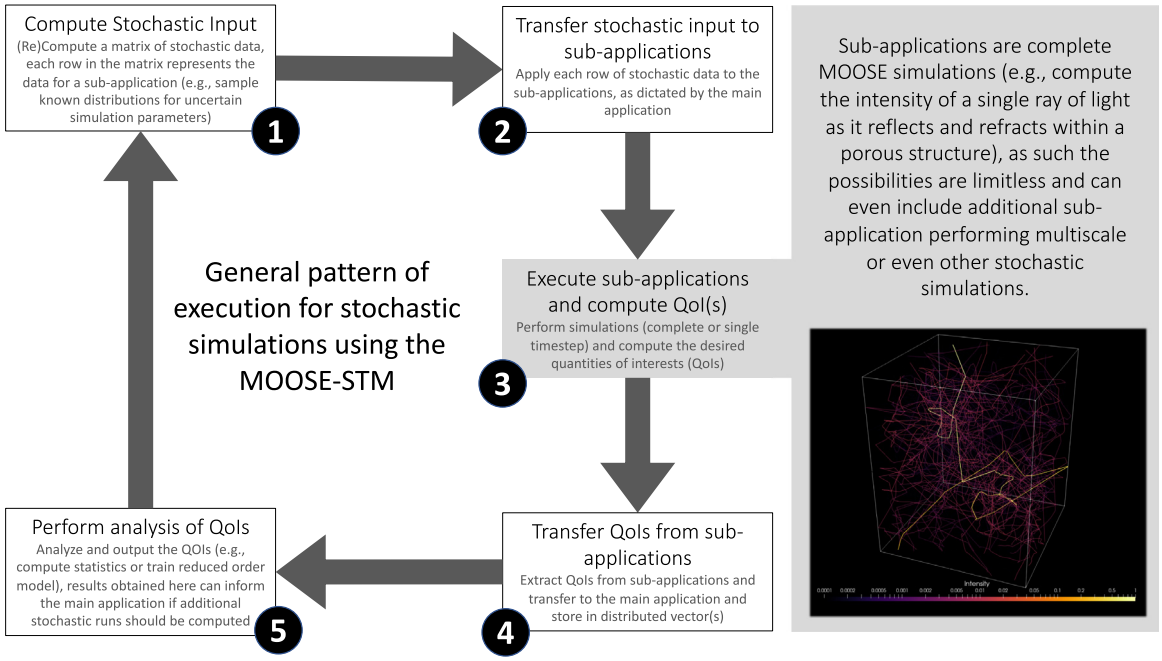

Fig. 1. Flow-diagram of the general use pattern for running stochastic simulations with the MOOSE-STM.

```
Real
MonteCarloSampler::computeSample(dof_id_type row_index,
                                 dof_id_type col_index)
{
  return _distributions[col_index]->quantile(getRand());
}
```

2. Define uncertain or design parameters with probability distributions and QoIs.
3. Define sampling strategy based on application, for instance: Monte Carlo for UQ or Sobol for sensitivity analysis.
4. Define post-processing objects for evaluating quantities like statistics, sensitivities, or meta-model training.

2.3. Software functionalities

Stochastic simulations, using the MOOSE-STM, involve a main application that is responsible for executing sub-applications with stochastic input. The sub-applications can be any MOOSE-based simulation, and the stochastic inputs can be any set of parameters used by this simulation.

Within the main application, the `Sampler` object(s) create stochastic input, typically by using a random number generator to sample distribution functions. Based on the available data, the MultiApp system is used to create sub-applications for each row of sample data, and the Transfer system passes the row of data to each sub-application. After the data on the sub-application has been modified, it executes either a complete solve of a full transient simulation or a single time step, depending on the configuration, and computes aggregation values, such as the average value of a field. The main application retrieves the aggregate values for output or additional calculations. It is important to note that every one of the code objects used within this process is customizable.

A key feature of the MOOSE-STM is the ability to execute simulations in memory, avoiding the need to spawn a unique instance of an executable for each stochastic input. The method of operation known as the "normal" mode launches a separate sub-application for each sample and will work for all MOOSE-based applications, even those that run non-MOOSE codes. The MOOSE-STM also includes two other executioner modes: "batch-reset" and "batch-restore". These are meant to ameliorate the memory limitations of spawning a large number of instances of a sub-application model.

These two batch modes operate by creating `MultiApp` instances that are reused for each stochastic simulation assigned to the processor (i.e., a batch). The "batch-reset" mode destroys the simulation and re-creates the next within the single instance of the sub-application. This reduces memory usage, but does not improve performance because the objects must still be created for each simulation, as is true in "normal" mode. For some simulations with code objects that cannot be modified after creation, such as those involving perturbations of the mesh geometry, this mode is a necessity since the associated code entities cannot be modified after creation. For simulations that do not have this limitation, the "batch-restore" mode leverages the backup system of MOOSE, which was originally developed for large-scale Picard iterations [9]. The backup system stores the minimum amount of information regarding a simulation that allows the simulation to stop and be restarted based on a checkpoint.

To demonstrate the implications of these differing methods of operation, we solved a 3D transient diffusion problem with two stochastic inputs using a Monte Carlo method. Fig. 2 demonstrates the memory improvement of the two batch methods relative to the "normal" method, as well as the performance improvement of the batch-restore method. The simulations were executed on a AMD EPYC 7702 64-Core machine running CentOS 8. Ten replicates of the analysis were conducted, and the timing reported includes the mean time for each analysis with the error bars giving the minimum and maximum times recorded.

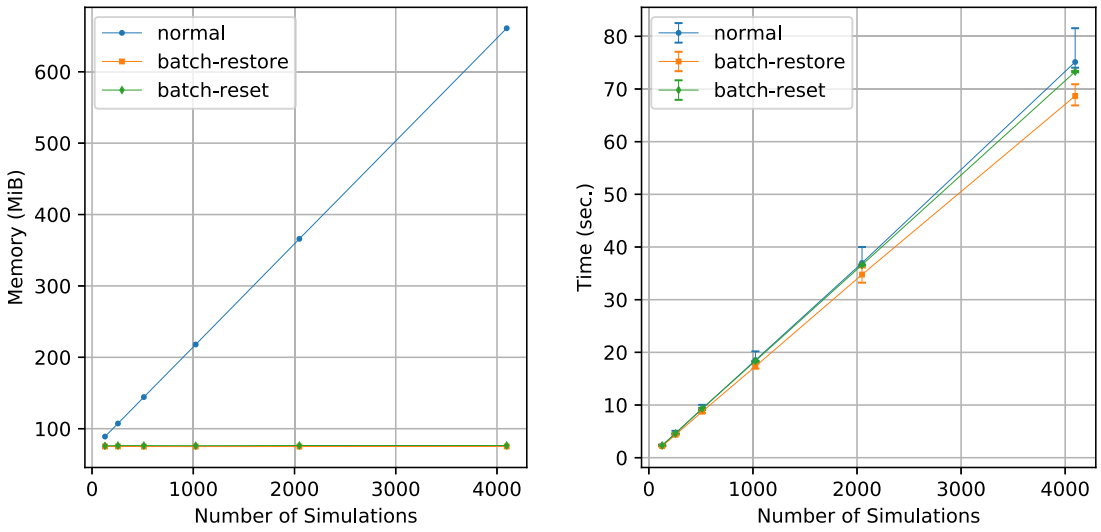

Fig. 2. Per process memory use (left) and total execution time (right) with an increasing number of simulations for the three operation modes for stochastic simulations using MOOSE-STM on 32 processors.

Table 1
Weak scaling analysis on up to 64 processors using 128 stochastic simulations per processor.

Processors	Simulations	Time (s)		
		Normal	Batch-reset	Batch-restore
1	128	67.9 (62.1, 69.2)	68.2 (67.6, 69.3)	61.7 (61.6, 62.1)
2	256	68.4 (62.5, 69.8)	68.3 (68.0, 68.4)	62.3 (62.0, 62.5)
4	512	68.2 (62.2, 69.3)	68.3 (68.2, 68.7)	62.9 (62.7, 63.0)
8	1024	68.9 (62.8, 70.5)	68.9 (68.4, 70.0)	62.6 (62.4, 62.8)
16	2048	69.1 (62.6, 71.0)	69.5 (68.5, 73.1)	62.5 (62.3, 62.7)
32	4096	70.0 (63.0, 75.3)	69.1 (68.7, 69.9)	62.8 (62.5, 63.6)
64	8192	75.5 (66.7, 82.8)	73.2 (72.7, 75.1)	66.3 (65.6, 67.1)

Similarly, a weak scaling study was performed on up to 64 cores using the same problem and machine. Table 1 includes the timing for a weak scaling analysis using 128 stochastic simulations per processor. This is a simple problem designed to run quickly on a workstation, so the overhead of communication and the startup/shutdown of the processes begin to significantly contribute to the simulation time. This overhead is less evident for larger simulations. For example, Ref. [2] demonstrated a strong scaling study of 1,000,000 stochastic simulations up to 768 processors.

3. Illustrative example

To illustrate the use of MOOSE-STM, we demonstrate the creation of a reduced-order model of a fixed-source diffusion–reaction problem from [10]. The problem consists of four material regions, as shown in Fig. 3, with Regions 1, 2, and 3 acting as fixed sources. The solution depends on the values of the diffusion and reaction coefficients as well as the source terms. The uncertainty for each term follows a uniform distribution, $U(a, b)$, where a and b are the limits of the distribution. The diffusion coefficients (units of cm) use a $U(0.2, 0.8)$ for Regions 1–3 and $U(0.15, 0.6)$ for Region 4. The reaction coefficients (units of 1/cm) for Regions 1–4 use $U(0.0425, 0.17)$, $U(0.065, 0.26)$, $U(0.04, 0.16)$, and $U(0.005, 0.02)$, respectively. The fixed sources (units of $\frac{n}{\text{cm}^3\ \text{s}}$, where n is number of particles), in Regions 1–3, use $U(5, 20)$.

A POD-RB model was selected for this example. This method requires access to the solution of the reaction–diffusion equation to train the model, thus MOOSE-STM is a natural fit. The POD-RB model is trained by using the `PODReducedBasisTrainer` object

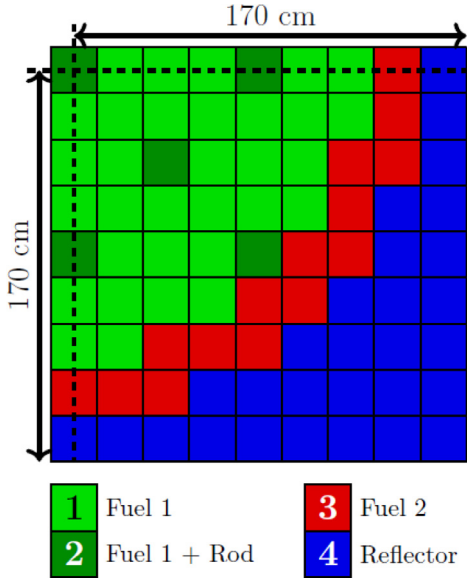

Fig. 3. Geometry of an illustrative example diffusion–reaction problem for nuclear reactor core simulations showing the four material regions (reproduced from [10]).

in the main input file. The POD-RB training phase operates on the model variable representing the flux, which is computed by the sub-application by solving the reaction–diffusion equation, and the value of 1×10^{-9} is used for the allowable reconstruction error. Input parameters must be specified in the Trainer block to specify how each stochastic input impacts the model (i.e., whether it scales an operator, source, or boundary condition). At the conclusion training, the POD-RB model is outputted to a file using the `SurrogateTrainerOutput` object.

Upon completion of the training phase, a linear combination of the resulting reduced-order basis functions can be used as a surrogate for the full-order model solution for multiphysics or statistical analysis calculations. In a separate input file, the trained model is loaded using the `PODReducedBasisSurrogate` object within the "Surrogates" block. The performance, on a single processor, of the surrogate model is shown in Table 2.

Table 2
Comparison of reaction–diffusion model execution times for the training of the POD-RB, as well as 1000 evaluations for the full-order model and various RB evaluations.

Process	Execution time (s)
1000 runs of full-order model	780
Training of POD-RB model using 100 samples	116
1000 runs of POD-RB using four basis functions	0.6
1000 runs of POD-RB using eight basis functions	0.9
1000 runs of POD-RB using 16 basis functions	1.6

Table 3
TRISO fuel failure probability results computed using Adaptive Importance Sampling which is an adaptive Monte Carlo algorithm and regular Monte Carlo using MOOSE-STM.

Method	Failure probability (P_f)	Coefficient of variation	# Model calls
Regular Monte Carlo	0.975E−4	0.1	1E6
Adaptive importance sampling	1E−4	0.073	5000

4. Impact

The MOOSE-STM is built to be efficient, scalable, and extendable, giving users the ability to perform tractable stochastic analyses on their MOOSE-based models with the MOOSE-STM out-of-the-box and developers the ability to implement custom routines without worrying about efficiency or parallelism. Furthermore, being a MOOSE module, the MOOSE-STM can be utilized in any MOOSE-based application, which covers a multitude of various physics [5]. The following examples present the impact and application of the MOOSE-STM to the failure probability analysis of nuclear fuel and structures.

In Ref. [2], the MOOSE-STM was used to compute the failure probability of tristructural isotropic (TRISO) coated fuel particle. This application shows the computational and memory efficiency of the MOOSE-STM when running and processing a large number of samples (on the order of 10^6), particularly the usefulness of the "batch-restore" feature. In Ref. [11], the MOOSE-STM sampling scheme is extended to perform adaptive sampling strategies [12], such as a Markov Chain Monte Carlo for TRISO fuel failure probability analyses. The modularity of the MOOSE-STM allows the implementation of these advanced sampling strategies for an efficient uncertainty quantification and the inference of response statistics for a number of important applications. Table 3 shows the impact of performing adaptive sampling by reducing the number of required samples by 2–3 orders of magnitude. In Ref. [3], the Grizzly code [13] creates a custom Sampler to efficiently perform probabilistic fracture mechanics evaluations of reactor pressure vessels. This application demonstrates the ability to extend the MOOSE-STM's functionalities and use them in optimal ways for application-specific stochastic analyses.

5. Conclusions

The MOOSE-STM provides a unique approach to performing stochastic simulation through embedding sampling, model execution, statistics, and meta modeling within a simulation framework. This approach allows for an analysis to be performed in-memory and distributed in parallel, thus allowing for a memory intensive analysis to be conducted in a timely and scalable fashion, without the need for extensive file input/output. Additionally, since these tools are native to the simulation framework, they can access all aspects of a simulation, which presents the ability to perform intrusive analysis methodologies in a general fashion for MOOSE-based applications.

The MOOSE-STM has demonstrated impact for research at Idaho National Laboratory in the nuclear energy sector, as demonstrated by the applications presented here. From a non-technical perspective, a native stochastic analysis tool within a simulation framework has impacted research by making these tools accessible with a uniform interface. This allows scientists to build complete applications that include traditional models as well as stochastic analyses, including creating tests and documentation with a familiar framework.

As an open-source project, there are numerous opportunities to extend this and apply it to other applications, and researchers, scientists, and industry partners are encouraged to explore and improve the capabilities of the effort. Finally, MOOSE, and consequently the MOOSE-STM, is developed with a process that meets nuclear quality assurance standards [14]. As such, it is aiding in the adoption of analyses by industry for the design of next generation nuclear power systems (e.g., [15]).

CRediT authorship contribution statement

Andrew E. Slaughter: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Roles/Writing – original draft, Writing – review & editing. **Zachary M. Prince:** Conceptualization, Methodology, Software, Supervision, Roles/Writing – original draft, Writing – review & editing. **Peter German:** Software, Formal analysis, Visualization, Roles/Writing – original draft, Writing – review & editing. **Ian Halvic:** Software. **Wen Jiang:** Formal analysis, Validation, Visualization, Roles/Writing – original draft, Writing – review & editing. **Benjamin W. Spencer:** Formal analysis, Validation, Visualization, Roles/Writing – original draft, Writing – review & editing. **Somayajulu L.N. Dhulipala:** Formal analysis, Validation, Visualization, Roles/Writing – original draft, Writing – review & editing. **Derek R. Gaston:** Conceptualization, Funding acquisition, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by DOE, under DOE Idaho Operations Office Contract DE-AC07-05ID14517. Accordingly, the U.S. government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. government purposes.

References

[1] Chan K, Tarantola S, Saltelli A, Sobol IM. Variance-based methods. John Wiley & Sons, Inc.; 2000, p. 167–97.
[2] Wen J, Hales J, Spencer B, Collin B, Slaughter A, Novascone S, et al. TRISO particle fuel performance and failure analysis with BISON. J Nucl Mater 2021;548.
[3] Spencer B, Hoffman W, Backman M. Modular system for probabilistic fracture mechanics analysis of embrittled reactor pressure vessels in the Grizzly code. Nucl Eng Des 2019;341:25–37. http://dx.doi.org/10.1016/j.nucengdes.2018.10.015, URL https://www.sciencedirect.com/science/article/pii/S0029549318308112.
[4] Melchers RE, Beck AT. Structural reliability analysis and prediction. John Wiley & Sons; 2017, http://dx.doi.org/10.1002/9781119266105.

[5] Permann CJ, Gaston DR, Andrš D, Carlsen RW, Kong F, Lindsay AD, et al. MOOSE: Enabling massively parallel multiphysics simulation. SoftwareX 2020;11:100430.
[6] Gaston D, Permann C, Peterson J, Slaughter A, Andrš D, Wang Y, et al. Physics-based multiscale coupling for full core nuclear reactor simulation. Ann Nucl Energy 2015;84:45–54.
[7] Adams B, Bohnhoff W, Dalbey K, Ebeida M, Eddy J, Eldred M, et al. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.12 user's manual. 2020.
[8] Tibshirani RJ, Efron B. An introduction to the bootstrap. In: Monographs on statistics and applied probability, vol. 57, Chapman and Hall New York; 1993, p. 1–436.
[9] Matthews C, Laboure V, DeHart M, Hansel J, Andrš D, Wang Y, et al. Coupled multiphysics simulations of heat pipe microreactors using DireWolf. Nucl Technol 2021.
[10] Prince Z, Ragusa J. Parametric uncertainty quantification using proper generalized decomposition applied to neutron diffusion. Internat J Numer Methods Engrg 2019;119(9):899–921.
[11] Dhulipala SL, Jiang W, Spencer BW, Hales JD, Shields MD, Slaughter AE, Prince ZM, Labouré VM, Bolisetti C, Chakroborty P. Accelerated statistical failure analysis of multifidelity triso fuel models. J Nucl Mater 2022;563:153604. http://dx.doi.org/10.1016/j.jnucmat.2022.153604.
[12] Au S, Beck J. A new adaptive importance sampling scheme for reliability calculations. Struct Saf 1999;21(2):135–58. http://dx.doi.org/10.1016/S0167-4730(99)00014-4, URL https://www.sciencedirect.com/science/article/pii/S01674730990001444.
[13] Spencer BW, Hoffman WM, Biswas S, Jiang W, Giorla A, Backman MA. Grizzly and BlackBear: Structural component aging simulation codes. Nucl Technol 2021;207(7):981–1003. http://dx.doi.org/10.1080/00295450.2020.1868278.
[14] Slaughter A, Permann CJ, Miller J, Alger B, Novascone S. Continuous integration, in-code documentation, and automation for nuclear quality assurance conformance. Nucl Technol 2021;1–8. http://dx.doi.org/10.1080/00295450.2020.1826804.
[15] Shaver D, Hu R, Vegendla P, Zou L, Merzari E. Initial industry collaborations of the center of excellence. 2019, http://dx.doi.org/10.2172/1556071, URL https://www.osti.gov/biblio/1556071.