INL/EXT-16-40114

NEAMS-IPL MOOSE Framework Activities

Cody Permann
Andrew Slaughter
John Peterson
David Andrš
Fande Kong
Derek Gaston
Richard Martineau

September 2016

INL/EXT-16-40114

NEAMS-IPL MOOSE Framework Activities

Cody Permann
Andrew Slaughter
John Peterson
David Andrš
Fande Kong
Derek Gaston
Richard Martineau

September 2016

Idaho National Laboratory
Modeling and Simulation Department
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

ABSTRACT

Coupled nonlinear multiphysics simulations require special treatment in order to obtain numerically-converged, physically-meaningful solutions. The Multiphysics Object Oriented Simulation Environment (MOOSE) was originally designed to handle "fully-coupled" systems of nonlinear equations, but this approach has definite limitations when applied to problems involving physics with drastically different time and spatial scales, and may also lead to nonlinear systems with prohibitive memory requirements and poor solvability characteristics. In these instances, it may be useful to decouple the solution strategy into "loosely-coupled systems of tightly-coupled equations." This approach involves multiple solve stages and information transfer in different directions. Examples of such approaches include Picard iteration and "operator split" schemes. In this work, we describe recently added framework capabilities that allow developer-built simulations with arbitrary state information to be saved and restored as needed. This capability allows the simultaneous evolution of physical models operating on different time scales to iterate as needed while discarding and recovering from failed steps. In addition to the Picard iteration scheme, new developments on a generalized eigensolver and physics module for solving the level set equation are also discussed.

CONTENTS

FIGURES

ACRONYMS

MOOSE	Multiphysics Object Oriented Simulation Environment
NEAMS	Nuclear Engineering Advanced Modeling and Simulation
PETSc	Portable Extensible Toolkit for Scientific Computation
RELAP-7	Reactor Excursion and Leak Analysis Program (Version 7)
SBO	Station blackout
SLEPc	Scalable Library for Eigenvalue Problem Computations
SUPG	Streamline Upwind Petrov Galerkin
TREAT	Transient Reactor Test Facility

1 Introduction

The MOOSE framework brings together advanced multiphysics coupling techniques, agile software development with high code quality standards, and a vibrant open scientific software development community. The MOOSE framework is used for a number of research and production quality nuclear energy related applications at Idaho National Laboratory, prominent academic institutions, and across the DOE complex. The framework, and the applications based on it, are all developed with the ideas of fast prototyping, ease of interoperability, and high performance on parallel architectures at the forefront of the design.

In this report, we describe several recent enhancements to the framework that have further extended the possibilities for advanced analysis, including, in particular, the ability to model nuclear reactor accident scenarios. One of the key enhancements described in this work is the novel use of the pre-existing restart and recovery system, in conjunction with the `MultiApp` system, in order to enable advanced simulation execution strategies. Simulation state can now be saved within the hierarchical `MultiApp` execution scheme. This new solution scheme gives developers new choices for running multiphysics applications, particularly those with very strong nonlinear effects, or those requiring coupling across disparate time and/or spatial scales.

Another enhancement described in this report is the addition of a generalized eigenvalue solver capability based on SLEPc, a suite of dense and sparse solver algorithms which is itself based on PETSc. The MOOSE framework already makes extensive use of the parallel linear and nonlinear solvers provided by PETSc, making SLEPc a natural fit within the framework as well. libMesh [1], the finite element library underlying MOOSE, has provided C++ interfaces to the SLEPc solvers via its `EigenSystem` and `EigenSolver` classes for some time, although they have not been used to date by any MOOSE-based applications. Efficient generalized eigenproblem solvers are clearly important in nuclear energy applications such as neutron transport, but they are also useful in a wide variety of other contexts including nonlinear stability analysis, structural dynamics, materials science, shape optimization, and quantum chemistry. We anticipate that the addition of this capability to MOOSE will have a significant positive impact on the number of collaborators who find the framework applicable for use in their research.

Finally, we also describe a new physics module for solving the well-known level set equation via the finite element method, which is currently under development. Similar to the generalized eigensolver capability, the level set equation arises in a number of different application areas, including fluid-structure interaction, front tracking, image processing, and snow science. There are a number of challenging aspects to the numerical solution of the level set equation, so it makes sense to develop a robust and well-tested solution strategy in a centralized location such as the MOOSE modules. Such an implementation can then be easily coupled to other MOOSE-based applications with little additional effort. In this report, we describe several recent and ongoing developments in the context of this module, and the remaining work which is to be accomplished.

The rest of this report is arranged in the following way: §2 discusses the `MultiApp`-based Picard iteration strategy and some recent results obtained using it on a coupled multiphysics problem. §3 describes the initial work which has been done on the generalized eigensolver capability, as well as some of the ongoing challenges with its development. §4 gives details on the level set module which is currently under development, as well as some promising initial results. Finally, §5 gives some conclusions and directions for future work.

2 MultiApp Picard Iteration Capability

MOOSE was originally designed to minimize the effort necessary to create "fully-coupled" multiphysics simulations. Programming interfaces were created for scientists and engineers to easily enable pluggable physics operators with extensible interfaces for solving a wide variety of problems. In 2013, new `MultiApp` and `Transfer` systems were added to the MOOSE framework to enable both "loose-coupling" and "tight-coupling" schemes. Utilizing these new systems a first-of-its-kind, full-core nuclear reactor model was simulated on the MOOSE platform [2].

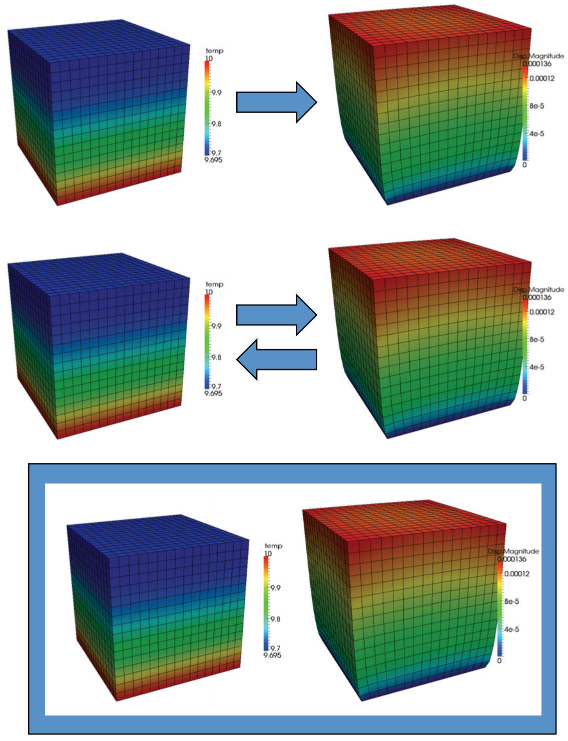


Figure 1: Figures depicting different solver strategies: top loosely-coupled, middle tightly-coupled, bottom fully-coupled

Figure 1 shows the three common coupling types made possible by the MOOSE `MultiApp` system. The top figure shows loose-coupling or operator-split coupling where one model is solved and information from the converged solution is transferred to another simulation and used to obtain a converged solution on the second model. After the second model is solved, time is typically advanced and the process is repeated. The middle figure shows a tight-coupling strategy where models are solved separately but iteration between them occurs until some tolerance is satisfied. Picard/fixed-point iteration [2–4] is an example of a tight-coupling scheme.

"Fixed-point iteration" [5] classically refers to the process of computing the limit of a sequence $\{x_n\}$ defined by $x_{n+1} = f(x_n)$ for a given function $f : \mathbb{R} \to \mathbb{R}$, while "Picard iteration" is classically a technique for proving the existence and uniqueness of solutions to ordinary differential equations [6]. In the present context of solutions to coupled systems of PDEs, we use these terms in a more general context to describe the following solution algorithm: suppose we are given two nonlinear systems of equations

$$\vec{F}(\vec{x};\vec{y}) = \vec{0} \tag{1}$$

$$\vec{G}(\vec{y};\vec{x}) = \vec{0} \tag{2}$$

where $\vec{x} \in \mathbb{R}^N$ is the "primary" unknown for $\vec{F} : \mathbb{R}^N \to \mathbb{R}^N$, but $\vec{F}$ also depends parametrically on a "secondary" unknown $\vec{y} \in \mathbb{R}^M$, and likewise $\vec{y}$ is the primary unknown for $\vec{G} : \mathbb{R}^M \to \mathbb{R}^M$, but $\vec{G}$ also depends parametrically on a secondary unknown $\vec{x}$. An example of this situation would be a fuels performance simulation which requires coolant temperature as a boundary condition coupled to a thermal hydraulics simulation which requires an input wall heat flux value.

A key observation is that solving (1) via a nonlinear solution algorithm (such as the preconditioned Jacobian-free Newton-Krylov method) requires only the approximation of $\frac{\partial \vec{F}}{\partial \vec{x}}$, likewise solving (2) requires only an approximation of $\frac{\partial \vec{G}}{\partial \vec{y}}$, these systems of equations are of size N^2 and M^2, respectively, and need not be solved simultaneously. This suggests the pseudocode shown in Algorithm 1. Note that in the second step of the while loop, the $\vec{G}$ system is solved with the most up-to-date value of $\vec{x}$, namely $\vec{x}_{i+1}$. There is also a natural extension of Algorithm 1 to problems with more than two systems of equations. As the number of systems increases, the order in which they are solved can potentially be important to the robustness of the algorithm and the speed at which it converges.

Algorithm 1 Picard iteration algorithm for solving (1) and (2) in a tightly-coupled manner.

```
Set initial guess $\vec{x}_0$, $\vec{y}_0$
Set $i = 0$
while not converged do
    Solve $\vec{F}(\vec{x}_i;\vec{y}_i) = \vec{0}$ for $\vec{x}_{i+1}$. Requires approximate $\frac{\partial \vec{F}}{\partial \vec{x}}$.
    Solve $\vec{G}(\vec{y}_i;\vec{x}_{i+1}) = \vec{0}$ for $\vec{y}_{i+1}$. Requires approximate $\frac{\partial \vec{G}}{\partial \vec{y}}$.
    Check for convergence of $\|\vec{x}_{i+1} - \vec{x}_i\|$, $\|\vec{y}_{i+1} - \vec{y}_i\|$.
    $i \leftarrow i + 1$
end while
```

Finally, the bottom image in Figure 1 shows a fully-coupled solution strategy where two models are solved together in a single system of equations. This solution strategy has the advantage that highly-nonlinear effects, which might otherwise cause fixed-point iterations to diverge or converge to non-physical solutions, do not arise. Drawbacks of the fully-coupled solution strategy are: (i) the size (in memory) of solver, and especially the preconditioner, may grow unacceptably as more systems of equations are added, and (ii) the nonlinear systems themselves may become more difficult to solve, and/or not amenable to pre-existing, well-known, and highly-researched solvers for the individual systems.

While the tight-coupling capability has been a part of the MOOSE framework for some time, it has previously suffered from several restrictions which reduced its applicability to problems of interest. For example, it was initially not possible to recover from a failed time step in one of the coupled applications unless both applications were using the same time step size. This restriction made it difficult to couple problems which naturally occurred on vastly different time scales. These restrictions inspired a novel usage of the pre-existing backup and recovery system in order to improve the robustness of the tight coupling algorithm.

2.1 Restart System

MOOSE contains an extensible backup and recovery system where user-defined data from any object can be serialized and saved in memory or written to disk. This user-defined data, along with the data from the framework itself, is useful for resuming a simulation after either a normal or unexpected termination. Each object in MOOSE has access to an interface where storage can be reserved based on name and data type. MOOSE maintains this data store for all processes and threads, and can serialize all stored data on demand. The serialized data can then be used for a variety of purposes such as checkpointing the entire system state, communicating data to other processes or threads in a data-structure agnostic fashion, or to execute complex time stepping strategies. Normally these serialized backups are written to disk as a simulation progresses for fault-tolerance purposes, but when using the Picard iteration scheme, backups are maintained in memory so they can be restored quickly during the solution process.

Generally only a small subset of data needs to be saved to enable the restart functionality. Most simulation information such as the system matrices and solver data structures can be regenerated on demand. Distributing the saved information in parallel also means that the process of creating these backups is scalable. This would not be the case if, for instance, all restart data was first communicated to the root processor before being written to disk.

2.2 Implementation

Figure 2 shows a typical `MultiApp` setup in MOOSE. Each node represents a separate simulation containing a separate equation system. MOOSE solves the equation system on each node in turn, in a user-controlled manner. Information can be aggregated or split and transferred from parent to child or child to parent as needed between solves. This is accomplished through the extensible `Transfer` system. By using the existing `MultiApp` and `Restart` systems in concert, it is possible to remove several of the restrictions from the initial Picard solution strategy. MOOSE supports tight-coupling schemes using a Picard iteration strategy while utilizing the `MultiApp` capability. MOOSE Executioners have the ability to "backup" or save the state of any tree node in the `MultiApp` hierarchical structure (see Figure 2). Saving a single node in the tree automatically saves the states of all children nodes from that point on down the tree. Executioners can then maintain any number of states by holding state objects in RAM or writing them to disk. These states can be restored as needed if a determination is made at any point to re-run an existing step or set of steps. Sub-apps that advance the simulation state by several steps during a single parent solve can be restored back to any previously saved state. This capability allows completely separate physics that may have different time scales to iterate as needed until a reasonable convergence tolerance is reached.

To verify the correct operation of the tightly-coupled Picard iteration capability, several test problems were created to verify the state restoration process. Additionally, several highly-nonlinear multiphysics problems have been devised which are difficult or infeasible to solve with a "fully-coupled" scheme or fail to produce reasonable answers when solved with an operator-split or "loosely-coupled" scheme. Figure 3 shows a plot from a coupled radiation transport/fuels performance simulation. Using a loosely-coupling scheme, extreme oscillations in the transport solution begin to occur as high burnup is reached in the fuels performance code. The use of the `MultiApp` Picard iteration capability with varying numbers of iterations is able to damp out the oscillations completely.

A nuclear power station undergoing a "blackout" scenario was simulated using a single fuel rod under normal operating conditions. The simulation is run for several months to accumulate a high

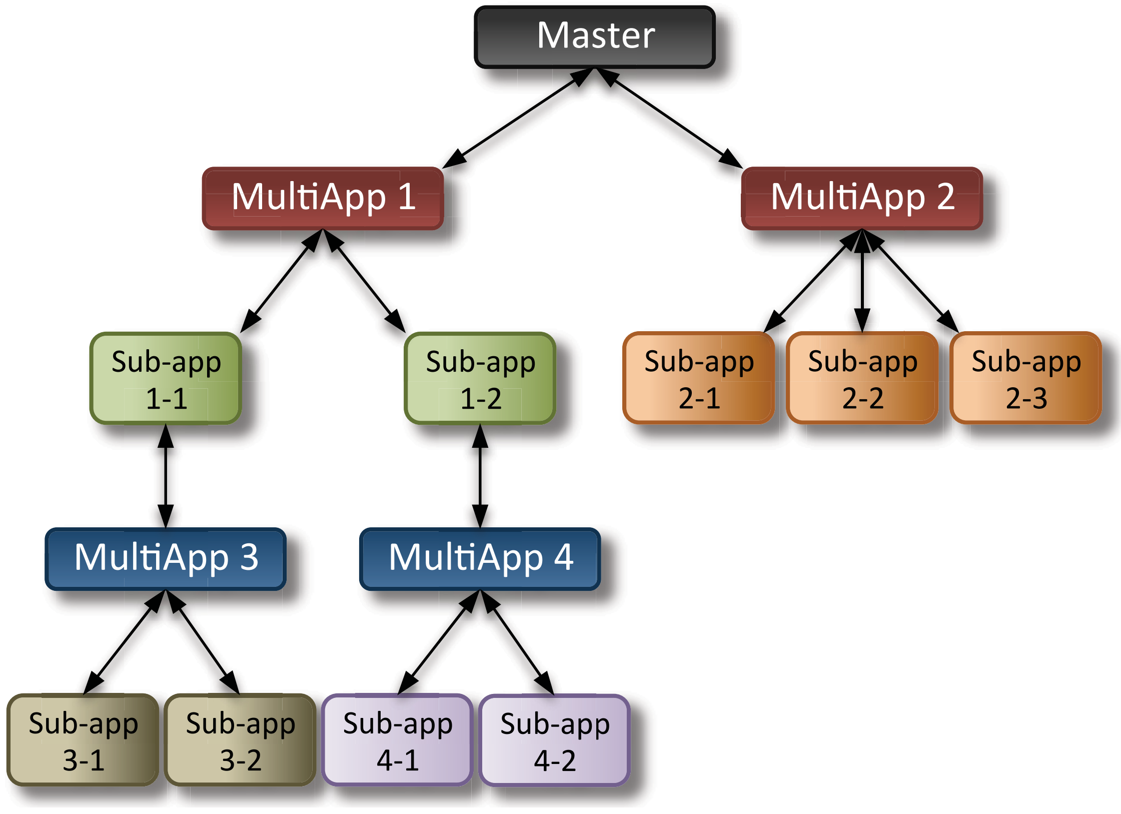


Figure 2: `MultiApp` simulation structure.

fuel burnup characteristics. Simultaneously a two-phase coolant channel was simulated within a Picard iteration tightly-coupled to the fuel simulation. At a specified time, the coolant channel circulation pumps are stopped and the reactor is scrammed simulating a station blackout (SBO) condition. Decay heat in the highly burned up fuel continues to produce heat while the lack of circulation provides insufficient cooling to the rod. As the coolant channel's temperature continues to rise, it begins to boil further reducing the ability to remove heat from the rod. The rod eventually heats and expands to the point where the cladding materials undergo plastic deformation. The complex interaction of these tightly-coupled physics models place high demands on the execution strategy to reach convergence among all models. The ability to run execute these models with and without advanced Picard iteration techniques greatly improves the analysis capabilities already in the framework. A screen capture of the simulation is shown in Figure 4.

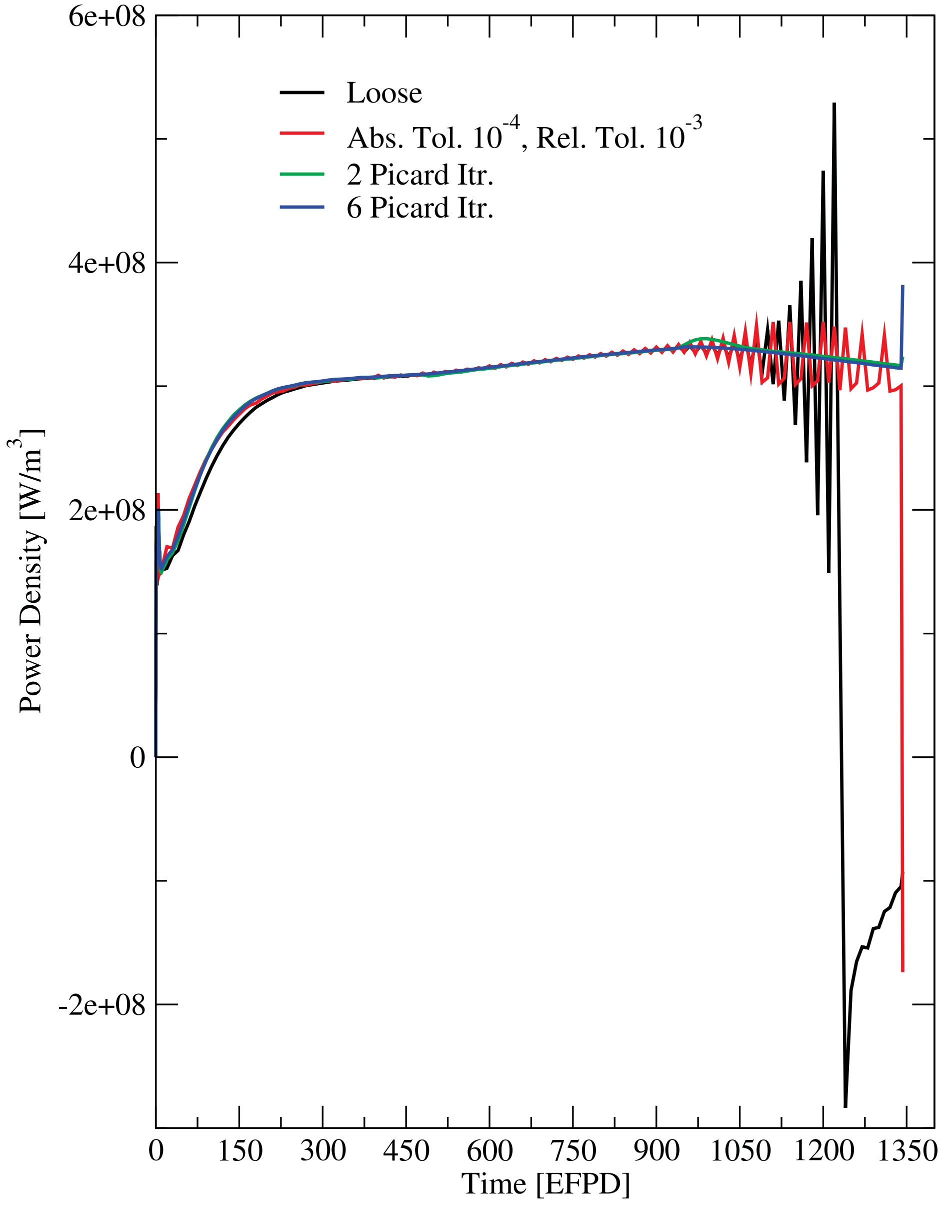


Figure 3: Different computed power density histories for loosely-coupled (black line), tightly-coupled (Picard, green and blue lines), and fully-coupled (red line) solution strategies.

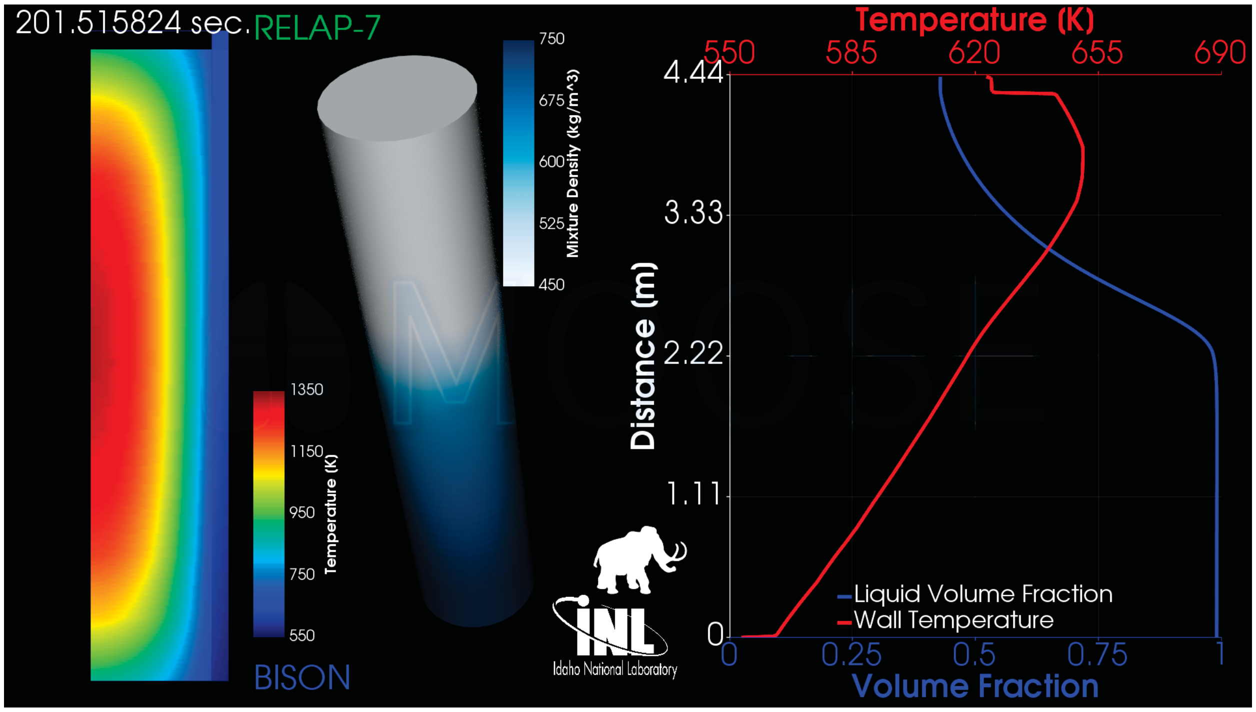


Figure 4: Coupled BISON/RELAP-7 accident scenario solution.

3 EigenSolver

Eigenvalue problems arise in many applications of physical simulations such as solid mechanics and transport theory. While the MOOSE framework has supported two methods (inverse iteration [7] and Newton-type method [8]) for solving some eigenvalue problems in the past, a fully-integrated solution compatible with all of the customizable MOOSE systems has been lacking. To create the most general solution method possible, we started with the generalized eigenvalue problem:

$$Ax = \lambda Bx. \tag{3}$$

where the A and B matrices typically represent different operators within the same PDE. For example, in transport theory, the entries in the A matrix represent all of the events neutrons may undergo in a background media including collision, streaming, scattering, and absorption. The entries in the B matrix represent the fission events, and the entries in the x vector represent the angular fluxes. The purpose of solving this eigenvalue problem is to determine the strength of the fission process by varying the scalar coefficient λ to obtain the self-sustained neutron populations without external sources.

For sophisticated multiphysics problems, the matrices A and B may depends on the eigenvector x, and then (3) is reformulated in the more general form:

$$\mathcal{A}(x) = \lambda\mathcal{B}(x), \tag{4}$$

where $\mathcal{A}$ and $\mathcal{B}$ are nonlinear operators, and $\mathcal{A}(\cdot)$ and $\mathcal{B}(\cdot)$ are interpreted as the corresponding nonlinear functions. Most algorithms for solving (3) or (4) perform a Rayleigh-Ritz projection for extracting the spectral approximations. This procedure reduces the problem to a low-dimensional subspace, and solutions of the original problem are obtained by solving the low-dimensional subspace problem. Solver algorithms such as power iteration [7], subspace iteration [9], Arnoldi's method [10], Krylov-Schur [11], and the generalized Davidson method [12] differ in the type of subspace used, how the subspace is constructed, and other characteristics aimed at improving convergence, etc. Among these subspace methods, the simplest one is the (inverse) power method, which is described in Algorithm 2.

Algorithm 2 The inverse iteration algorithm. The solve step which updates $x^{(i+1)}$ requires either a linear or nonlinear solve depending on whether we are solving (3) or (4), respectively.

```
Input: operators $\mathcal{A}$ and $\mathcal{B}$, and an initial guess $(\lambda^{(0)}, x^{(0)})$
for $i = 0, \ldots, n_{\max}$ do
    $x^{(i+1)} = \lambda^{(i)}\mathcal{A}^{-1}\mathcal{B}x^{(i)}$
    $\lambda^{(i+1)} = \lambda^{(i)}\frac{\|\mathcal{B}x^{(i)}\|}{\|\mathcal{B}x^{(i+1)}\|}$
    if converged then
        break
    end if
end for
Output: $(\lambda^{(i+1)}, x^{(i+1)})$
```

Algorithm 2 has been implemented in MOOSE using a Newton-type method to solve the generalized eigenvalue problem. These two algorithms work effectively for transport problems, but are difficult to maintain because they are outside the original scope of problems MOOSE was designed to solve. Additionally, Algorithm 2 is not easily extensible to applications which require the computation of more than one eigenvalue, since it computes only the smallest eigenvalue.

PETSc [13], the numerical linear algebra library used by MOOSE, is also used by SLEPc [14], a library for solving generalized eigenvalue problems of the type discussed here. The goal of this work was to provide the infrastructure needed to support the interfaces in SLEPc in both the MOOSE framework and the underlying libMesh library. The new design creates a more flexible and extensible solution scheme, while simultaneously reducing the maintenance burden of the existing interfaces. Having developers with experience in more than one of the PETSc, SLEPc, libMesh, and MOOSE projects was key to the feasibility and success of this work. Although full integration of the generalized eigensolver is an ongoing effort, prototype implementations have already enabled its use in multiple MOOSE-based applications. Continued iteration on the current design, and FY 2017 efforts toward redesigning the MOOSE executioner, will continue to enhance this capability and make it more accessible to scientists and engineers using MOOSE-based applications.

4 Level Set Module

The level set method is commonly used for front tracking problems [15]. It is modeled with the multi-dimensional advection equation

$$\frac{\partial u}{\partial t} + \vec{v} \cdot \nabla u = 0, \tag{5}$$

where t is time, $\vec{v}$ is a known velocity field, and u is the variable representing the interface. Traditionally u is taken to be the signed distance from the interface. This problem poses significant challenges: numerical stabilization techniques must be employed to avoid node-to-node oscillations, and intermediate re-initialization steps must be undertaken to evolve the interface in a physically-consistent manner.

The basic requirements for solving the level set equation in MOOSE are summarized in the following milestone objectives:

1. Solve the level set equation numerically using the Galerkin finite element method.
2. Apply a standard stabilization scheme.
3. Implement a basic re-initialization scheme.

In order to test and demonstrate the use of the level set module, a benchmark rotating bubble problem requiring each of the three aforementioned components was undertaken.

The weak form of (5) is: find $u_h \approx u$ such that

$$\left(\frac{\partial u_h}{\partial t}, \psi_i\right) + (\vec{v} \cdot \nabla u_h, \psi_i) = 0 \tag{6}$$

holds for all admissible test functions, ψ_i. Here u_h is the finite element approximation to u, and we have employed the inner-product notation $(u, v) \equiv \int_\Omega uv \; \mathrm{d}x$ for convenience. Solving (6) in MOOSE requires the addition of a single physics kernel for the second term on the left-hand side of (6). This kernel is included in the level set module as `LevelSetAdvection`, and allows one to use coupled variables to represent the velocity field.

As with any pure convection equation, implementing the level set equation with the Galerkin finite element method requires numerical stabilization if node-to-node oscillations are to be avoided and standard solution accuracy is to be maintained. The level set module also includes kernels which implement the Streamline Upwind/Petrov-Galerkin (SUPG) stabilization method [16, 17] for (6), which is given by

$$\left(\frac{\partial u_h}{\partial t} + \vec{v} \cdot \nabla u_h, \psi_i + \frac{h}{2\|\vec{v}\|}(\vec{v} \cdot \nabla \psi_i)\right) = 0, \tag{7}$$

where h is the element length, and $\|\vec{v}\|$ is the Euclidean norm of the velocity vector. We note that the h-dependent terms add stability to the numerical scheme by introducing the symmetric, velocity-dependent artificial viscosity contribution:

$$\left(\frac{h}{2\|\vec{v}\|}\vec{v} \cdot \nabla u_h, \vec{v} \cdot \nabla \psi_i\right) \tag{8}$$

in a so-called "consistent" manner. That is, if the true solution u satisfies (5), it is easy to see, by inspection, that it will also satisfy (7), since the strong form of the residual appears in the first "slot" of the inner product. The stabilization terms are implemented in the code as two additional kernels: `LevelSetTimeDerivativeSUPG` and `LevelSetAdvectionSUPG`.

The advected variable u is often used to track a moving front, and therefore it is desirable that the total amount of u present in Ω remain the same throughout the simulation. This is equivalent to stating that the area within a given contour of u does not change. It is also frequently desirable that u remain a signed distance function throughout the course of the simulation. There is a vast amount of research into methods which guarantee either conservation, the signed distance property, or both. The level set module in MOOSE currently implements a conservative "re-initialization" algorithm that transforms u_h into a smooth function in the range $[0, 1]$ rather than a signed distance, which is useful for certain types of problems such as phase identification.

The weak form of the re-initialization equation is: find U_h such that

$$\left(\frac{\partial U_h}{\partial \tau}, \psi_i\right) + (\nabla \psi_i, (-f + \epsilon \nabla U_h \cdot \hat{n}_*)\hat{n}_*) = 0, \tag{9}$$

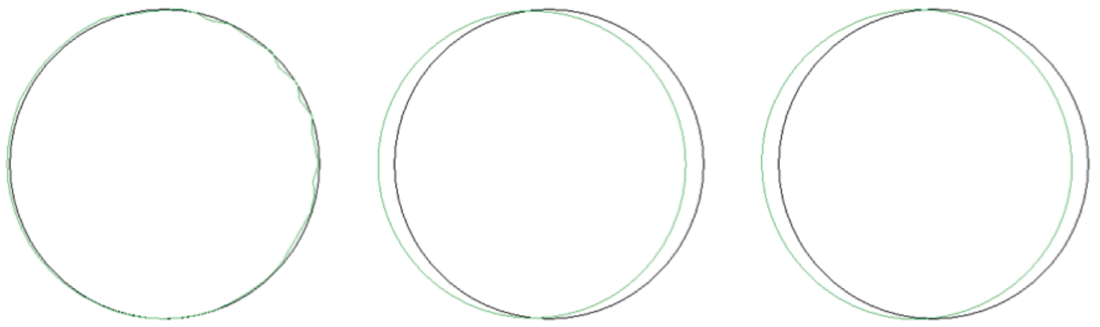
holds for every admissible ψ_i, where τ is the pseudo time during the re-initialization, $\hat{n}_*$ is the normal vector computed from the level set variable u_h at pseudo time $\tau = 0$, ϵ is the interface thickness, $f \equiv U_h(1 - U_h)$, and $U_h(\tau = 0) = u_h$. Steady-state for U_h is detected when

$$\frac{\|U_h^{m+1} - U_h^m\|}{\Delta\tau} < \delta, \tag{10}$$

where δ is a problem-dependent tolerance, and $U_h^m \equiv U_h(\tau = m\Delta\tau)$. When steady-state is achieved, u_h is set equal to the re-initialized solution U_h, and the entire process is repeated at time $t + \Delta t$.

The majority of the MOOSE level set module implementation effort was focused on the addition of this re-initialization algorithm. Improvements to the MOOSE `MultiApp` system, such as a more robust solution copy function and an improved full-solve `MultiApp`, allowed the level set equation and the re-initialization problem to be solved on the same variable at each time step.

A typical benchmark problem for the level set equation is the rotating bubble. The problem involves initializing u_h with a "bubble" of radius 0.15 at $(0.0, 0.5)$ for $\Omega = [-1, 1]^2$. This bubble is then advected with the given velocity field $\vec{v} = (4y, -4x)$, so that, at $t = \pi/2$, the bubble should return to its original position.



(a) Level set equation only. (b) Level set with SUPG. (c) Level set with re-initialization.

Figure 5: Initial condition (black contour) and final state (green contour) after two revolutions at time $t = \pi$ for the basic level set equation (a), SUPG method (b), and re-initialization method (c).

Figure 5 shows the results of running this problem using the level set equation alone, the SUPG stabilized level set equation, and the re-initialized level set equation. Figure 6 is a plot of the area of the circle during the three simulations. Note that in the unstabilized, un-reinitialized level set equation, both area conservation and stability issues are readily apparent. The instabilities are especially obvious in Figure 6, where the drastic area changes are due to numerical oscillations in the solution field. Adding SUPG stabilization helps ameliorate the stability concern but it suffers from loss of area conservation. The re-initialization scheme is both stable and area-conserving.

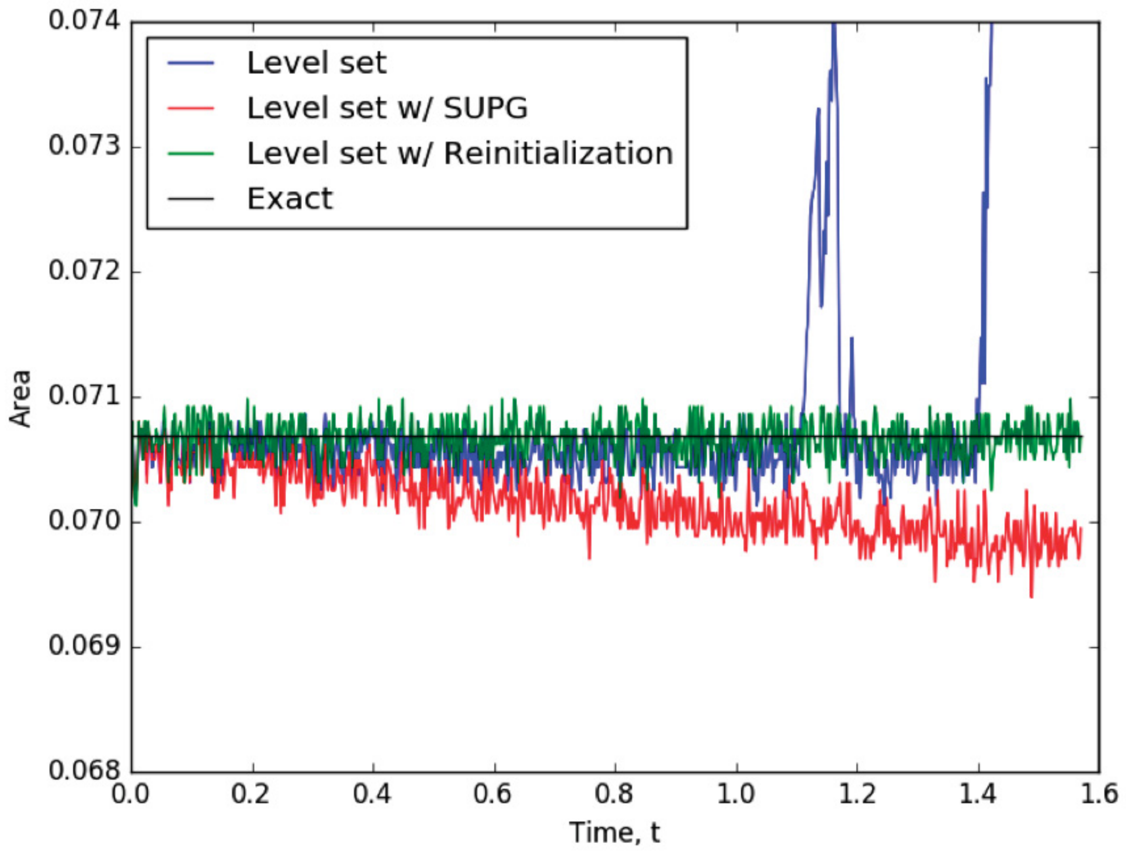


Figure 6: Comparison of area inside the bubble during simulations described in Figures 5 (a), (b), and (c).

The re-initialization methods performs well, but it is computationally expensive and picking the pseudo timestep size $\Delta\tau$, steady-state criteria δ, and interface thickness ϵ correctly for the re-initialization problem is a non-trivial difficulty. Nevertheless, the level set module provides a valuable starting point and proof-of-concept for other researchers interested in the method, and the existing algorithm can no doubt be tuned to the needs of specific applications in terms of conservation and computational cost requirements.

The following additional tasks should be completed to make the level set module more useful and robust for real-world applications:

1. Develop automated techniques means for setting the various re-initialization tuning parameters ($\Delta\tau$, ϵ, etc.).

2. Implement a signed-distance-preserving re-initialization scheme based on established methods, e.g. [18].

3. Implement additional stabilization techniques such as the Galerkin Least Squares [19] method and “shock/discontinuity capturing” schemes [20, 21].

4. Create module-specific input file syntax for level set problems to simplify input file generation and usage.

5. Solve additional benchmark problems with various stabilization and re-initialization schemes, and investigate different mesh refinement and adaptivity strategies for said problems.

5 Conclusions

Several advanced capabilities have been added to the MOOSE framework to enable new solution strategies. Supporting a Picard iteration strategy in the `MultiApp` system allows for the solution of highly-nonlinear models, and the coupling of physics across disparate time scales. These capabilities are already being used to study nuclear reactor accident scenarios involving radiation transport, high-fidelity multiscale fuels performance, and multiphase coolant channel models. These capabilities are also being used to support other DOE modeling objectives related to the TREAT restart project. The addition of an improved eigenvalue solver and the level set module continue to expand the types of pluggable physics that engineers and scientists are able to implement and study. As each new capability is added to the framework portfolio, an emphasis on coupling and flexibility is made to ensure ease of use by the applications and programs that rely on the toolkit. This crosscutting, consolidated strategy reduces costs to the various funding programs, and maximizes the amount of science performed by applications built with the MOOSE framework.

6 References

1. B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006.

2. D. R. Gaston, C. J. Permann, J. W. Peterson, A. E. Slaughter, D. Andrš, Y. Wang, M. P. Short, D. M. Perez, M. R. Tonks, J. Ortensi, and R. C. Martineau. Physics-based multiscale coupling for full core nuclear reactor simulation. *Annals of Nuclear Energy*, 84:45–54, October 2015.

3. J. D. Hales, M. R. Tonks, F. N. Gleicher, B. W. Spencer, S. R. Novascone, R. L. Williamson, G. Pastore, and D. M. Perez. Advanced multiphysics coupling for LWR fuel performance analysis. *Annals of Nuclear Energy, Special Issue on Multi-Physics Modelling of LWR Static and Transient Behaviour*, pages 98–110, October 2015.

4. S. R. Novascone, B. W. Spencer, J. D. Hales, and R. L. Williamson. Evaluation of coupling approaches for thermomechanical simulations. *Nuclear Engineering and Design*, 295:910–921, December 2015.

5. L. Burden. *Numerical Analysis, 3rd ed.* PWS Publishers, 1985. Section 2.2, Fixed-Point Iteration.

6. G. Teschl. *Ordinary Differential Equations and Dynamical Systems.* American Mathematical Society, 2012.

7. Y. Saad. *Numerical Methods for Large Eigenvalue Problems: Revised Edition*, volume 66. SIAM, 2011.

8. D. A. Knoll, H. Park, and C. Newman. Acceleration of k-eigenvalue/criticality calculations using the Jacobian-free Newton-Krylov method. *Nuclear Science and Engineering*, 167(2):133–140, 2011.

9. Z. Bai and G. W. Stewart. Algorithm 776: SRRIT: A Fortran subroutine to calculate the dominant invariant subspace of a nonsymmetric matrix. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):494–513, 1997.

10. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the solution of algebraic eigenvalue problems*, volume 11. SIAM, 2000.

11. G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, 2002.

12. R. B. Morgan and D. S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM Journal on Scientific and Statistical Computing*, 7(3):817–825, 1986.

13. Satish Balay et al. PETSc Users Manual. ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

14. V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):351–362, 2005.

15. S. Osher and R. Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Science & Business Media, 2006.

16. A. N. Brooks and T. J. R. Hughes. Streamline Upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32:199–259, 1982.

17. J. Donea and A. Huerta. *Finite element methods for flow problems*. John Wiley & Sons, 2003.

18. C. Min. On reinitializing level set functions. *Journal of Computational Physics*, 229(8):2764–2772, April 2010.

19. T. J. R. Hughes, L. P. Franca, and G. M. Hullbert. A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least–squares method advective–diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73:173–189, 1989.

20. T. J. R. Hughes and M. Mallet. A new finite element formulation for computational fluid dynamics: II. Beyond SUPG. *Computer Methods in Applied Mechanics and Engineering*, 54:341–355, 1986.

21. F. Shakib, T. J. R. Hughes, and Z. Johan. A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier–Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 89:141–219, 1991.