INL/EXT-20-59954

# Summary of Bison documentation and UX milestones - NEAMS FY20 Report

August 2020

Idaho National Laboratory

D. Yushu
S. A. Pitts
D. vanWasshenova
A. Recuero
D. J. McDowell
A. D. Lindsay
B. W. Spencer
D. Schwen

Idaho National Laboratory

*INL is a U.S. Department of Energy National Laboratory operated by Batelle Energy Alliance, LLC*

## DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed 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. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

INL/EXT-20-59954

# Summary of Bison documentation and UX milestones - NEAMS FY20 Report

*D. Yushu*[1]
*S. A. Pitts*[1]
*D. vanWasshenova*[1]
*A. Recuero*[1]
*D. J. McDowell*[1]
*A. D. Lindsay*[2]
*B. W. Spencer*[1]
D. Schwen[1]

**September 2020**

[1]Computational Mechanics and Materials, Idaho National Laboratory
[2]Computational Frameworks, Idaho National Laboratory

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

# Contents

# 1 Introduction

This summary report contains an overview of work performed under the work package entitled "MS-20IN020107 - BISON advanced numerical model development and usability improvements - INL", which is focused on the development and support of the fuel performance code BISON [1]. The second chapter lists FY20 milestones titles, completion schedule, and milestone level. Subsequent chapters summarize and demonstrate completion of milestones and activities. The last chapter outlines FY21 proposed future work.

## 1.1 Executive summary

Improvements were made to the engineering scale fuel performance code BISON in several key areas. Mechanical and thermal contact solves are an important component of light water reactor fuel simulation, in which the closing of a gas gap between fuel and cladding largely governs the thermomechanical properties of a fuel rod. We implemented a novel mortar method using dual basis functions, which permit a complexity reduction of the numerical solve. This new capability was demonstrated and proven to work in chapter 3. Thermal contact was also improved through the addition of missing terms to the Jacobian matrix, resulting in a tangible performance increase.

With a growing and maturing code base we emphasized the reduction in complexity of the user facing part of BISON, namely the input file syntax. We realized that certain frequent use cases dominate the application of BISON. To simplify these frequent use cases, that are based on common reactor concepts, we developed a short hand syntax that removes repetitive boilerplate from our input files. All the physics and parameterizations required for common nuclear materials (including thermal and mechanical properties) are available through compact action block in BISON input files introduced in chapter 4. For the advanced user, each of these compact blocks can be automatically expanded to enable full customizability.

In chapter 5 we present work on improving the convergence behavior of systems with multiple inelastic mechanics models applied (such as creep, relocation, smeared cracking). Analysts have identified these simulations as challenging for BISON. We have worked on converting large portions of the BISON physics models over to the new automatic differentiation system. Automatic differentiation provides a perfect preconditioning matrix and can improve the robustness and accelerate the numerical convergence of simulations significantly. We have demonstrated this on several use cases.

To facilitate the uptake of BISON by users of the Windows operating system we investigated the possibility of building executables on that platform directly. As described in chapter 6 this proved challenging. While substantial headway was made resulting in multiple patches to MOOSE and libMesh, ultimately our recommendation is to utilize the Windows Subsystem for Linux in its version 2 (WSL2), which is available on Windows 10, to build BISON executables. We have documented the process and have improved user tools for building BISON input files to support this approach.

A large software product such as BISON requires significant effort to maintain a high software quality standard. In FY20 BISON was audited and found to comply with the NQA-1 standard. A growing user base requires support and at the same time is a valuable resource for feedback. We conducted our first BISON user survey. These efforts are summarized in chapter 7.

# Bibliography

[1] R. L. Williamson, J. D. Hales, S. R. Novascone, M. R. Tonks, D. R. Gaston, C. J. Permann, D. Andrs, and R. C. Martineau. Multidimensional multiphysics simulation of nuclear fuel behavior. *Journal of Nuclear Materials*, 423:149–163, 2012.

# 2 Milestone summary and completion schedule

FY-2020 Milestones and the completion dates are listed in Table 2.1. The milestones are listed by topic. Following chapters contain a short description of each milestone and references to related detailed documentation. Where applicable, a representative technical result from the work is included.

Table 2.1: FY-2020 Milestones for BISON advanced numerical model development and usability improvements

| Milestone | Completion Date | MS Level |
|---|---|---|
| Contact improvements through Mortar and Automatic Differentiation | 8/31 | M3 |
| Reduced complexity Bison input files | 7/31 | M3 |
| Improve numerical convergence for stacked inelastic models | 7/31 | M3 |
| Turn-key executable Bison for redistribution | 3/31 | ACT |
| Identify and address remaining issues in design and user documentation to ensure full NQA-1 compliance and document usability improvements | 9/30 | M2 |

# 3 BISON Contact Improvement

Most nuclear fuel systems analyzed by BISON incorporate fuel surrounded by one or more protective layers of either cladding or other materials that contain the fuel and fission products. Capturing the effects of the mechanical and thermal interactions between these materials is essential for accurate modeling of the performance of the fuel system. These interactions are modeled in BISON using contact enforcement techniques, which allows for large relative sliding between these materials.

Both mechanical and thermal contact inherently introduce significant nonlinearities to the system of solved nonlinear equations. Because of this, efforts to improve BISON's robustness have often focused on improving the handling of contact. With the preconditioned Jacobian-Free Newton Krylov (JFNK) scheme used for most BISON simulations for robustness improvements improving the accuracy of the Jacobian contributions for a given physics model is often the initial focus. Prior efforts to improve contact robustness have largely focused on improving the Jacobian contributions for mechanical contact, but thermal contact is also an important source of nonlinearities, although it has not received as much focus.

Efforts to improve contact robustness in BISON in Fiscal Year 2020 have focused on three main areas:

1. **Improved Jacobian entries for traditional node/face thermal contact algorithms**: Prior to the work documented here, some of the entries in the Jacobian matrix corresponding to coupling of the response between the two interacting surfaces in thermal contact were not implemented. These entries were added, and resulted in significant robustness and speed improvements for most BISON models that use the standard contact algorithms.

2. **Testing of mortar contact enforcement algorithms for BISON models**: Mortar methods offer the potential for significant further improvements both in solution accuracy and robustness over the node/face algorithms that are currently used for most BISON simulations. General-purpose methods for mortar-based mechanical and thermal contact were developed in prior years, but had not yet been tested on BISON simulations. This work focused on developing mortar-based implementations of the thermal contact models used by BISON, and on applying mortar contact for the first time in practical BISON simulations.

3. **Development of advanced dual-mortar contact enforcement algorithms**: A dual-basis version of the mortar formulation that offers improved characteristics regarding its interface to the solver has been implemented and tested on basic problems.

Details of these developments are provided in the following sections. It is important to note that at the current time, the mortar contact implementations in BISON are limited to two dimensions (2D). While they offer the potential for significant improvements in solution robustness and accuracy, these methods will still require more development before they are fully usable for all production BISON simulations. For this reason, in addition to pushing forward application and development of mortar methods, some of the work conducted here was also focused on improving the node/face algorithm, which is currently used in production and will likely be continued for a significant time in the future.

## 3.1 Improved Jacobian Entries for Thermal Contact

Thermal contact in MOOSE/BISON ultimately involves computing a heat flux between two surfaces that are dependent both on the size of the gap between the surfaces and the temperatures of the two surfaces. In addition to the mortar algorithms still under development, two variants of a node/face enforcement algorithm are available in BISON, and are currently used for the majority of BISON models:

1. A true node/face algorithm computes the flux at the locations of nodes on a designated secondary side of the interface, and applies integrated fluxes as point sources at the locations of the secondary nodes and their projected positions on the primary side of the interface.

2. A "quadrature-based" variant of the node/face algorithm creates pseudo-nodes at the quadrature points on both surfaces involved in the contact interaction, and applies integrated fluxes at these points on both faces. These fluxes are computed separately on the two sides of the interface, and this technique generally produces results that are smoother than those of the true node/face algorithm. This technique is currently used by the majority of BISON models.

Both of these techniques are implemented using the same set of classes, and share significant code.

For coupled thermomechanical contact, the effect of the contact constraints on the residual for the thermal and mechanical degrees of freedom associated with a contact interaction can be expressed as:

$$\begin{bmatrix} \Delta\boldsymbol{r}_{u,p} \\ \Delta\boldsymbol{r}_{u,s} \\ \Delta\boldsymbol{r}_{t,p} \\ \Delta\boldsymbol{r}_{t,s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}_{uu,pp} & \boldsymbol{K}_{uu,ps} & \boldsymbol{K}_{ut,pp} & \boldsymbol{K}_{ut,ps} \\ \boldsymbol{K}_{uu,sp} & \boldsymbol{K}_{uu,ss} & \boldsymbol{K}_{ut,sp} & \boldsymbol{K}_{ut,ss} \\ \boldsymbol{K}_{tu,pp} & {\color{blue}\boldsymbol{K}_{tu,ps}} & \boldsymbol{K}_{tt,pp} & {\color{red}\boldsymbol{K}_{tt,ps}} \\ {\color{blue}\boldsymbol{K}_{tu,sp}} & \boldsymbol{K}_{tu,ss} & {\color{red}\boldsymbol{K}_{tt,sp}} & \boldsymbol{K}_{tt,ss} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{u}_{p} \\ \Delta\boldsymbol{u}_{s} \\ \Delta\boldsymbol{t}_{p} \\ \Delta\boldsymbol{t}_{s} \end{bmatrix} \tag{3.1}$$

where $\Delta\boldsymbol{r}$ is the change in the residual due to contact, $\boldsymbol{u}$ is the vector of displacement degrees of freedom and $\boldsymbol{t}$ is the vector of temperature degrees of freedom. The $u$ and $t$ subscripts indicate the displacement and temperature variables, and the $p$ and $s$ subscripts indicate the primary and secondary surfaces.

The terms in the Jacobian matrix in Equation 3.1 that were already implemented prior to the present work are shown in black. As part of the present effort to improve contact robustness, two separate change sets that were merged into MOOSE implement the terms shown in red and blue, which contain the derivatives of the flux applied to one side with respect to the temperature and displacement on the other side of the interface. These terms are all fully implemented for the "quadrature-based" option, which is most commonly used, while all terms except for $\boldsymbol{K}_{tt,ps}$ are implemented for the pure node/face algorithm.

The default algorithm for preconditioning in BISON and all MOOSE-based applications is a block-diagonal approach, which only includes coupling between the degrees of freedom pertaining to a given variable, so there is no coupling between temperature and displacement by default. Because the terms highlighted in blue in Equation 3.1 were not previously implemented, the Jacobian had significant errors, and using the full tangent matrix for preconditioning often had the effect of decreasing robustness and increasing run-time. Adding the terms in red affected all BISON analyses that include contact (because they are part of the block-diagonal matrix). Because they are in the off-diagonal blocks of the matrix, the terms in blue only have an effect on the solution if those off-diagonal blocks are included in the Jacobian.

Historically, very few BISON analyses actually used the off-diagonal Jacobian blocks, in large part because the newly-implemented terms were missing, which, in many cases, it actually hurt performance by including them. Adding the sets of terms described here has had a significant positive benefit for analyses that use only block-diagonal preconditioning, and an even greater benefit if the full Jacobian is used. To assess the effects of these changes, timing of the full suite of BISON assessment cases was evaluated before and after these changes, as described in the following section.

### 3.1.1 Assessment Suite Performance Analysis

The addition of the cross-gap coupling terms to the Jacobian in BISON had a significant positive impact on the run-time performance of its assessment cases. To take full advantage of these code changes, it is important to use the full matrix for preconditioning, rather than the block diagonal matrix, which was used prior to these changes by the majority of assessment cases.

To assess the impact of the Jacobian improvements on run time, an ad-hoc statistical analysis was performed to quantify their impact and provide insight into which specific assessment cases benefitted from these changes. The analysis was performed by gathering three independent runs of the full BISON assessment suite with and without the changes. The reference runs were gathered *without* the changes taken from the January 1, 2020, release of BISON, establishing the control group for the test. These cases did not use the full tangent matrix for preconditioning, instead using a version of the code shortly before the cross-gap coupling terms were added. The other three runs were collected independently and include the cross-gap coupling terms, and also include modifications to the input file to use the full tangent matrix for preconditioning.

Assessment cases that did not pass all three runs both before and after the changes were dropped from the analysis. Dropping these cases allowed for a more accurate comparison by not taking into account tests that may have had inaccurate performance times due to crashing or timing out. We also only looked at current assessment cases that utilize the thermal contact model as these were the cases that would've been the most heavily impacted due to the code-change. As a result, 79 cases out of 282 were compared. Many of these cases were new additions with no previous data to compare. Future analyses will take these cases into account.

While the mean percent change in run times was $-7.65\%$, many cases saw much larger improvements. Figure 3.1 compares the impact of the code changes for each assessment case. A very small number of cases showed a significant slowdown when using these changes. These cases are the subject of ongoing investigation.

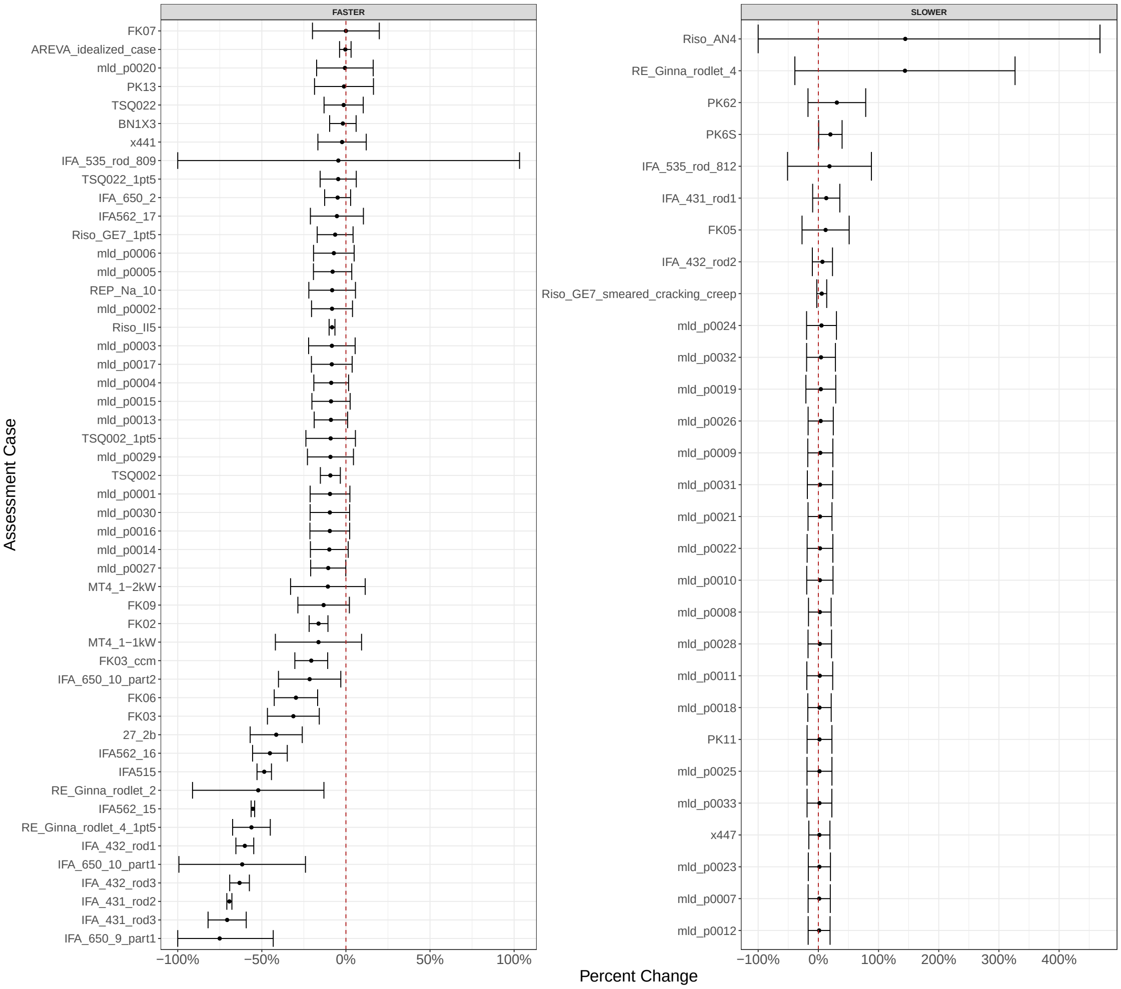


Figure 3.1: Assessment cases that were significantly impacted by the new addition.

### 3.1.2 Conclusions and Future Work

Adding the missing Jacobian terms outlined in Equation 3.1 resulted in significant run-time and robustness improvements in models that use the “quadrature-based” option for thermal contact together with the node/face mechanical contact algorithm. The Jacobian matrix now accounts for all interactions between the degrees of freedom involved in contact. It is still a known issue that not all of these entries are completely correct because computing the derivatives of the BISON contact model is non-trivial. Further work to refine these terms would give further, less dramatic, improvements in solution robustness. The framework for including all terms has now been developed, so once the mathematical form of those derivatives is refined, including corrected forms would be relatively straightforward.

Although there is still some room for improvements in the Jacobians for node/face contact, they are now largely correct. Future work in contact enforcement in BISON will largely be focused on development of mortar methods, as described in the following sections. Mortar methods are expected to improve both the accuracy and robustness of the solution.

## 3.2 Mortar-Based Thermomechanical Contact

The node/face algorithm employed in most BISON simulations suffers from the fact that its behavior can be quite discontinuous, which presents challenges for both solution robustness and accuracy. When a node or integration point is suddenly projected onto a new face on the primary surface due to relative movement of the contact surfaces, a force or excitation to the system will appear. From a physical standpoint, the enforcement of thermal and mechanical contact on an intermediate subdomain, such as the one carried out by mortar, ensures that solids gradually enter into contact. That is, mortar segments — on which constraints are enforced — are created incrementally based on the contacting bodies' geometries. This gradual approach is more natural, as it does not rely on discrete evaluating points, and improves the system's convergence properties. On the other hand, from a computational perspective, the work performed this fiscal year on thermomechanical contact ensures that this key aspect of the simulation is compliant with the effort to enable automatic differentiation in BISON. In other words, the C++ classes created to enable mortar thermomechanical contact allow for an automatic computation of the Jacobian of the generalized forces, which may act on temperature and displacement variables.

Mortar enforcement of thermal and mechanical contact was previously available in MOOSE, but had not yet been developed to the point where it could be tested on realistic fuel performance models. Notably, there was no way to use the gap conductance model employed for light water reactor (LWR) fuel performance simulations. In this fiscal year, a mortar implementation of the LWR gap conductance model was developed and tested.

### 3.2.1 Implementation Details

In its current form, the new mortar-based gap conductance class computes the overall gap conductance in the following way

$$h_{gap} = h_g + h_r + h_s \tag{3.2}$$

where $h_g$ is the overall gas conductance, which takes into account the various gases present, $h_r$ is the conductance due to radiation, and $h_s$ is the contribution to conductance due to solid-solid contact.

Mortar contact can be set up using a MOOSE Action, which takes care of the creation of MOOSE and BISON objects, including constraints, auxiliary variables, and auxiliary kernels. An example of how the block for mortar thermal contact looks in a BISON input file is shown in the following excerpt:

```
[ThermalContactMortar]
  [./thermal_contact]
    variable = lm
    secondary_variable = temp
    primary_boundary = 100
    primary_subdomain = 10000
    secondary_boundary = 101
    secondary_subdomain = 10001

    emissivity_clad = 0.8
    emissivity_fuel = 0.5
    gascond_scalef =  1.0
    contact_pressure = frictionless_normal_lm

    min_gap = 1e-5
    meyer_hardness_model = MATPRO
    interaction_layer = true
    layer_thickness = layer_thickness_action
    use_displaced_mesh = true
  [../]
[]
```

The Mortar action sets up all the necessary variables needed to perform an LWR simulation. Additional command blocks to create lower-dimensional blocks can be manually added by the user or through the contact action. In the cases presented here, the lower-dimensional blocks were created in the input file as follows

```
[Mesh]
  ...
  [./secondary]
    type = LowerDBlockFromSidesetGenerator
    new_block_id = 10001
    new_block_name = 'secondary_lower'
    sidesets = '10' # pellet_outer_surface
    input = smeared_pellet_mesh
  [../]
  [./primary]
    type = LowerDBlockFromSidesetGenerator
    new_block_id = 10000
    sidesets = '5' # clad_inside_right
    new_block_name = 'primary_lower'
    input = secondary
  [../]
[]
```

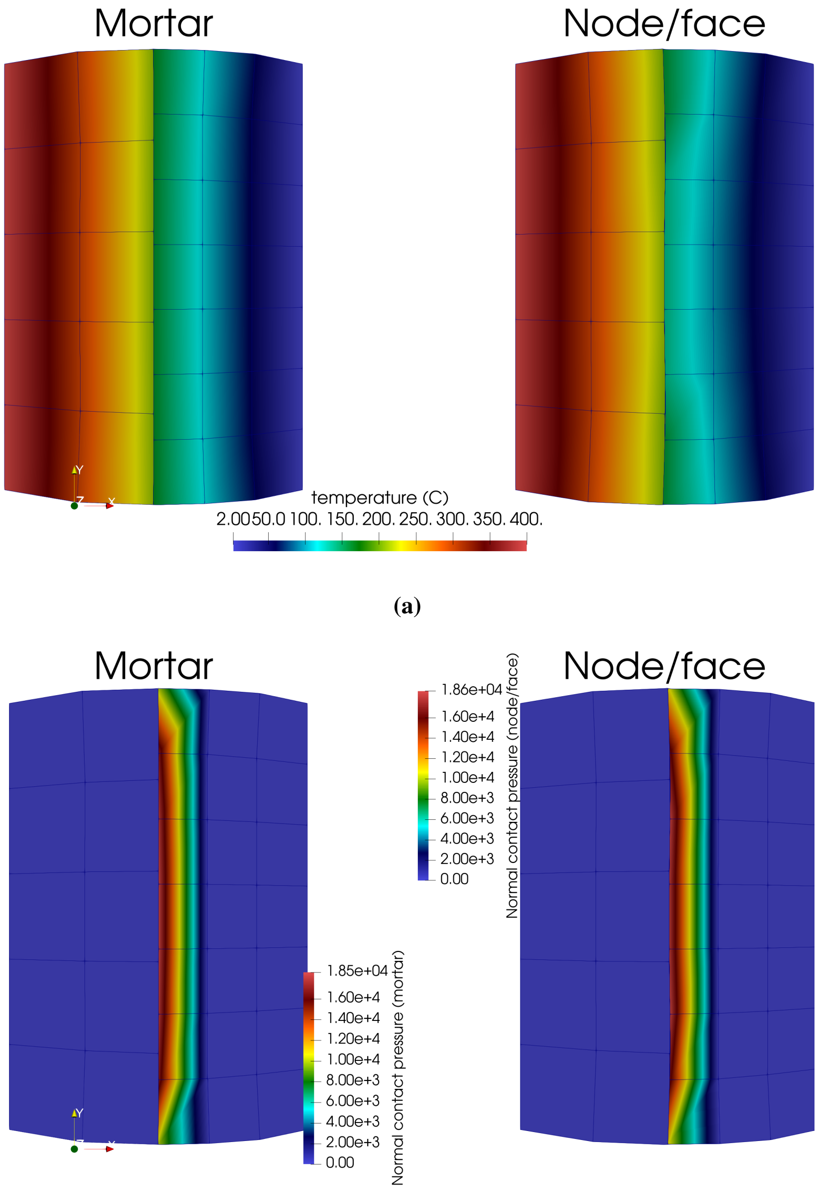
On these lower-dimensional blocks, the mortar constraints are enforced through Lagrange multipliers. `NormalMortarMechanicalContact` constraint allows the distribution of mortar constraint residuals throughout the system dimensions.

The implementation has been tested in a small problem and an LWR assessment case, and compared to the node on face thermomechanical results, as shown in Sections 3.2.2 and 3.2.3.

### 3.2.2 Numerical Results of a Simple Problem

To test the mortar implementation of the LWR gap conductance model, a simple problem of two 2D blocks that come into mechanical and thermal contact is tested here. Though small, this problem includes all the features and variables involved in a full-scale LWR simulation. That is, all the terms in Equation (3.2) play a role in the solution. In addition, the parameters shown in Section 3.2.1 are considered here.

Figure 3.2 shows a comparison between this reduced-order LWR-like problem between a mortar approach and the existing node/face approach. The results in terms of temperature and displacements are very similar. However, a closer look at the contact surfaces and temperature distribution shows the mortar solution provides smoother results. These are justified by the constraint enforcement on a subdomain, as opposed to the point-like enforcement used by the node/face approach.



**(a)**

**(b)**

Figure 3.2: Temperature and contact pressure distributions throughout the two blocks show good agreement between recently-implemented mortar contact and the existing node on face formulation.

To better show quantification on the temperature results, the difference between the two implementations is shown in Table 3.1. These results serve as an initial verification of the implementation and allow us to move on to more challenging cases.

| Node on contacting surface | Mortar | Node-on-face | Difference (%) |
|---|---|---|---|
| 0 | 207.022 | 203.633 | 1.66 |
| 1 | 205.784 | 201.565 | 2.09 |
| 2 | 205.081 | 208.036 | 1.42 |
| 3 | 205.081 | 208.036 | 1.42 |
| 4 | 205.784 | 201.565 | 2.09 |
| 5 | 207.022 | 203.633 | 1.66 |

Table 3.1: Temperature values at the nodes of the secondary surface.

### 3.2.3 Numerical Results of an Assessment Case

This subsection shows a brief comparison of one BISON assessment problem simulated with node/face and mortar thermomechanical constraint enforcement, respectively. The assessment case is a super ramp PK62, available in BISON's test suite. As outlined, the changes to the input file to use thermomechanical mortar are: (1) Creation of lower-dimensional blocks, (2) addition of mortar contact constraints, (3) and addition of mortar thermomechanical contact action with LWR, nuclear problem parameters.

Figures 3.3 and 3.4 show a side-by-side comparison of the node/face formulation with kinematic contact and the mortar formulation results for temperature and mechanical contact pressure. Temperature distributions in the fuel and clad are very close; maximum and minimum temperatures and their gradients through the fuel are virtually the same. Contact pressure results are, however, different. Mechanical contact between fuel and clad is more sensitive to the general system state. For this reason, and due to a slight mismatch in the time step, mechanical contact is present in a larger area when mortar contact is used at this stage of the simulation.

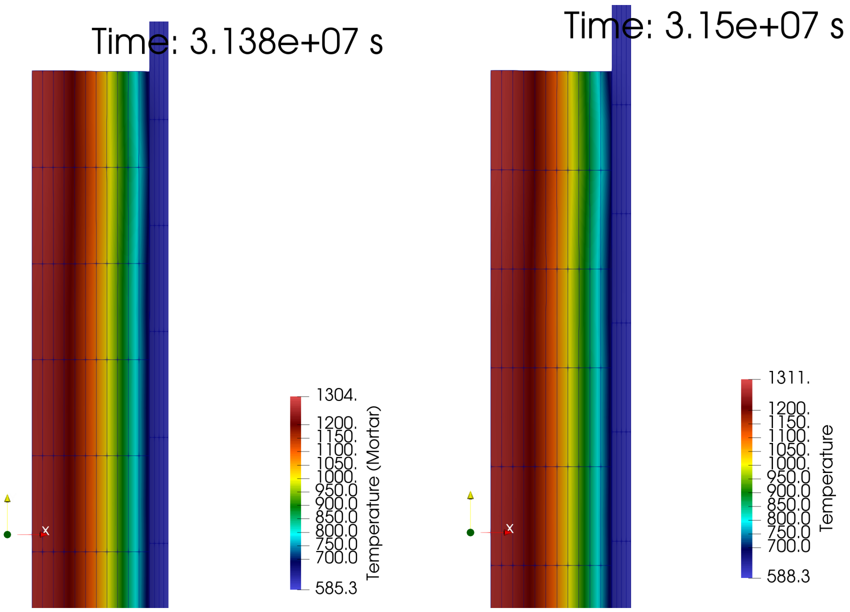
Some results on numerical performance for a serial simulation are shown in Figure 3.5. Mortar thermomechanical contact converges consistently and in few nonlinear iterations, which is the case for all mortar-based frictionless contact cases tested. Nonetheless, it requires a larger computer effort. The use of automatic differentiation and its C++ types tend to add to the wall time per nonlinear iteration, which may partially explain the results presented in Figure 3.5(b).

### 3.2.4 Conclusions and Future Work

Thermomechanical contact has been developed to a point where it can be used to simulate and analyze practical BISON simulations. Despite the initial satisfactory results, future work on the mortar implementation is necessary to achieve a higher degree of maturity and represent a competitive, computationally efficient alternative.

The numerical performance results shown in this section highlight the need for investigating strategies to speed up the mortar implementation and better benefit from its enhanced Jacobian approximations. In addition, thermomechanical contact needs to be verified against a larger number of assessment cases and simulation scenarios.

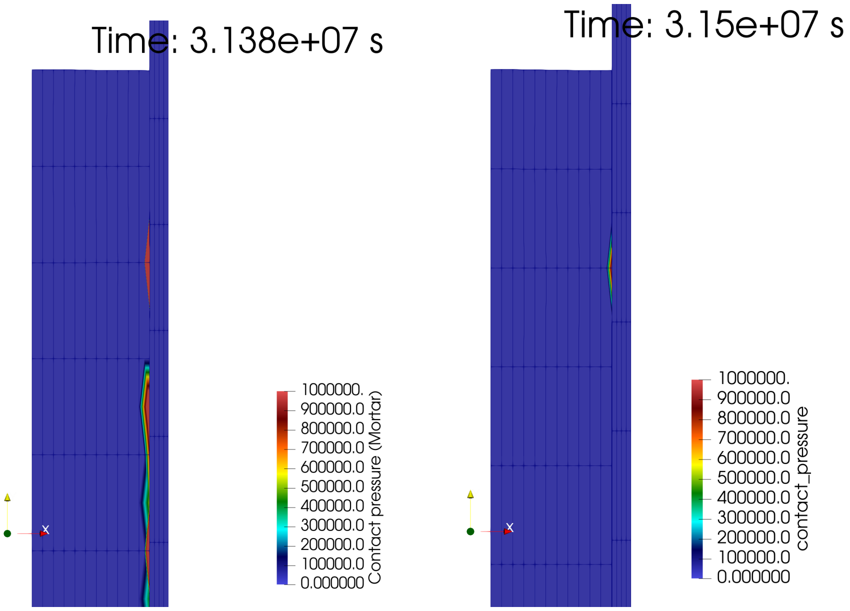
Various other topics remain. Convergence with frictional contact can be challenging and requires the use of scaling parameters which may sometimes be difficult to select by the user. In addition, MOOSE's mortar implementation allows for a point-wise or integral enforcement of the Karush-Kuhn-Tucker conditions. The difference betwen these two approaches in terms of performance and accuracy on BISON assessment cases remains to be investigated.



**(a)** Mortar

**(b)** Node/face

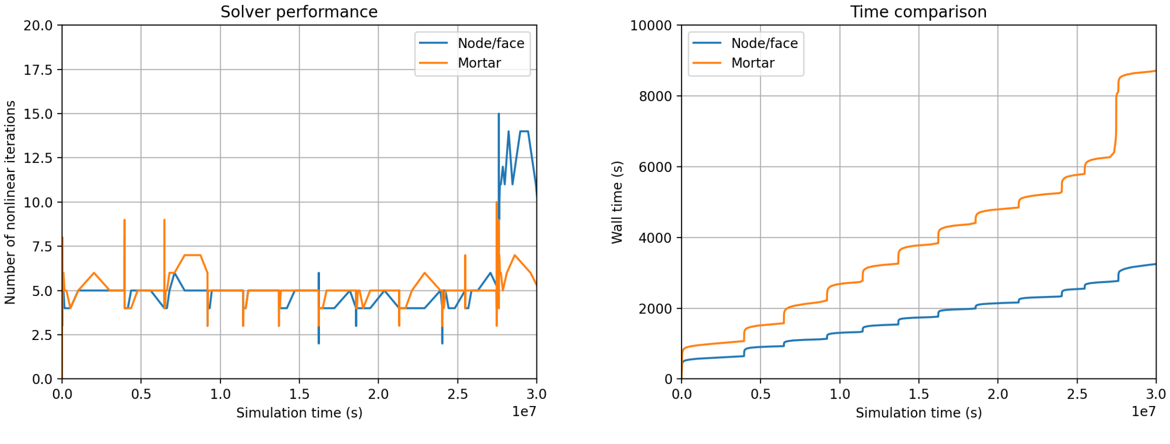
Figure 3.3: Temperature distribution detail for PK62 BISON assessment case



**(a)** Mortar

**(b)** Node/face

Figure 3.4: Contact pressure distribution detail for PK62 BISON assessment case



**(a)** Step's nonlinear iterations

**(b)** Accumulated wall time

Figure 3.5: Numerical performance comparison between thermomechanical enforcement approaches

## 3.3 Dual Mortar Method Development

Mortar methods, which were originally introduced as an abstract domain decomposition technique, are characterized by: (1) consistently treating the contact interaction through an exact evaluation of the surface integrals in the weak formulation and (2) an imposition of the occurring interface constraints in a weak sense [1, 2]. This allows for a variationally consistent treatment of non-penetration and frictional sliding conditions in the context of mechanical contact analysis [1]. In terms of numerical analysis, the mortar finite element method allows the coupling of different discretization schemes and triangulations across subdomains [3].

In the standard mortar finite element method — used to implement thermomechanical contact in BISON — the matching at the interface is achieved by using a modified trace space that serves as the basis of Lagrange multipliers. A feasible combination of the primal and trace spaces needs to satisfy the so-called *inf-sup* condition in order to acquire the desired stability and approximation properties. An alternative choice for the discrete Lagrange multiplier space is the dual space [4]. In contrast to the standard mortar approach, it generates interface coupling conditions that are much easier to realize without impinging upon the optimality of the method. Besides, the dual mortar approach readily allows local condensation of the Lagrange multipliers and thus preserves the positive definiteness of the system matrix. This indicates a broader range of robust preconditioners/solvers can be used to solve the simplified system efficiently.

We investigated a dual-basis function approach for the discretization of the Lagrange multiplier based mortar contact. Dual basis functions offer the promise of avoiding saddlepoint matrices and the ability to condense out degrees of freedom. Both result in a system that is easier to converge and is numerically more well behaved.

There are two common approaches to discretize the primal variable and the Lagrange multipliers: the standard approach and the dual approach. The standard approach leads to a combination of shape functions that need to satisfy the *inf-sup* condition. In contrast, the dual approach constructs dual Lagrange multiplier shape functions based on a biorthogonal condition with the primal variable shape functions.

### 3.3.1 Background

Since the applicability of dual mortar approach is not limited to the type of the continuous problem, we discuss this approach based on a general discrete quasi-static form

$$\boldsymbol{f}_{\text{int}}(\boldsymbol{u}) + \boldsymbol{f}_{\text{co}}(\boldsymbol{u}, \boldsymbol{\lambda}) = \boldsymbol{f}_{\text{ext}}, \tag{3.3}$$

where $\boldsymbol{u}$ and $\boldsymbol{\lambda}$ denote the discretized primal variable and Lagrange multipliers, respectively. The $\boldsymbol{f}_{\text{int}}$ denotes the nonlinear internal forces, $\boldsymbol{f}_{\text{ext}}$ denotes the external forces resulting from the standard finite element discretization. The $\boldsymbol{f}_{\text{co}}$ represents the contact forces arising from the integral over the secondary (slave) interface.

#### 3.3.1.1 Spatial discretization of primal variables

The standard finite element interpolation is employed to approximate the primal variable. We denote the primal variable by $u(\boldsymbol{x})$,

$$u(\boldsymbol{x}) \approx \sum_{k=1}^{n_u} \phi_k(\boldsymbol{x}) \boldsymbol{u}_k, \tag{3.4}$$

where $\boldsymbol{x}$ represents the spatial location, $n_u$ represents the number of degrees of freedom (DOFs) of $u$, $\boldsymbol{u}_k$ denotes the nodal discrete primal variable, and $\phi_k(\boldsymbol{x})$ is the standard shape function.

#### 3.3.1.2 Spatial discretization of Lagrange multipliers

The Lagrange multiplier $\lambda$ defined on the secondary interface is approximated by

$$\lambda(\boldsymbol{x}) \approx \sum_{k=1}^{n_\lambda} \psi_k(\boldsymbol{x}) \boldsymbol{\lambda}_k, \tag{3.5}$$

where $n_\lambda$ represents the number of DOFs of $\lambda$ along the interface, $\boldsymbol{\lambda}_k$ denotes the nodal discrete Lagrange multiplier, and $\psi_k(\boldsymbol{x})$ is the basis function. As we employ dual mortar approach, the shape function $\psi_k(\boldsymbol{x})$ is the dual basis function fulfilling the following biorthogonal condition along the contact interface $\Gamma$

$$\int_\Gamma \psi_j(\boldsymbol{x}) \phi_k(\boldsymbol{x}) \, \mathrm{d}s = \delta_{jk} \int_\Gamma \phi_k(\boldsymbol{x}) \, \mathrm{d}s, \tag{3.6}$$

where $\delta_{jk}$ is the Kronecker delta function. The biorthogonal condition can be assumed to hold in every lower-dimensional element $\Gamma_e$ (i.e.)

$$\int_{\Gamma_e} \psi_j(\boldsymbol{x}) \phi_k(\boldsymbol{x}) \, \mathrm{d}s = \delta_{jk} \int_{\Gamma_e} \phi_k(\boldsymbol{x}) \, \mathrm{d}s. \tag{3.7}$$

There are many possible choices of dual basis function that satisfy Equation (3.7). For readers interested in this, refer to (e.g., [4].

In our study, the dual basis functions are assumed to be linear combinations of the standard finite element shape functions

$$\psi_j(\boldsymbol{x}) = \sum_{j=1}^{n_{e,\lambda}} a_{kj} \phi_k(\boldsymbol{x}), \tag{3.8}$$

where $n_{e,\lambda}$ is the number of DOFs of the Lagrange multiplier in the lower-dimensional element $\Gamma_e$, $a_{jk}$ denote the undetermined coefficients. One can obtain $a_{jk}$ by substituting Equation (3.8) into the element level biorthogonal condition (i.e., Equation (3.7)). This results in the following local system of equations:

$$\mathcal{M}_e \mathcal{A}_e = \mathcal{D}_e, \tag{3.9}$$

where

$$\mathcal{M}_e = \begin{bmatrix} \int_{\Gamma_e} \phi_1 \phi_1 \, \mathrm{d}s & \dots & \int_{\Gamma_e} \phi_{n_{e,\lambda}} \phi_1 \, \mathrm{d}s \\ \vdots & \ddots & \vdots \\ \int_{\Gamma_e} \phi_1 \phi_{n_{e,\lambda}} \, \mathrm{d}s & \dots & \int_{\Gamma_e} \phi_{n_{e,\lambda}} \phi_{n_{e,\lambda}} \, \mathrm{d}s \end{bmatrix},$$

$$\mathcal{A}_e = \begin{bmatrix} a_{1,1} & \dots & a_{1,n_{e,\lambda}} \\ \vdots & \ddots & \vdots \\ a_{n_{e,\lambda},1} & \dots & a_{n_{e,\lambda},n_{e,\lambda}} \end{bmatrix}, \tag{3.10}$$

$$\mathcal{D}_e = \begin{bmatrix} \int_{\Gamma_e} \phi_1 \, \mathrm{d}s & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \int_{\Gamma_e} \phi_{n_{e,\lambda}} \, \mathrm{d}s \end{bmatrix}.$$

By solving Equation (3.9), $\mathcal{A}_e = \mathcal{M}_e^{-1} \mathcal{D}_e$, we obtain the coefficients for the dual basis functions, and thereby a linear independent system that is formed by $\{\psi_j\}_1^{n_{e,\lambda}}$. Solving the local system in Equation (3.9) is computationally trivial. Therefore, the support of the dual basis $\psi_j$ is the almost the same as that of the standard bass $\phi_k$.

### 3.3.2 Simplification of the matrix system

After discretization, we can write out the global matrix system. The global matrix loses the positive-definite property due to the existence of Lagrange multipliers. This typically introduces difficulties to the solution of the matrix system. Fortunately, by using the dual basis for the discretization of the Lagrange multipliers, one can simplify the matrix system, and significantly reduce the computational cost. This is achieved by condensation of the Lagrange multipliers.

The original global system of equations can be written as a block matrix as follows:

$$\begin{bmatrix} K_{1,ii} & K_{1,ic} & & & \\ K_{1,ci} & K_{1,cc} & & & M \\ & & K_{2,ii} & K_{2,ic} & \\ & & K_{2,ci} & K_{2,cc} & D \\ & M^\intercal & & D^\intercal & 0 \end{bmatrix} \begin{bmatrix} u_{1,i} \\ u_{1,c} \\ u_{2,i} \\ u_{2,c} \\ \lambda \end{bmatrix} = \begin{bmatrix} r_{1,i} \\ r_{1,c} \\ r_{2,i} \\ r_{2,c} \\ 0 \end{bmatrix} \tag{3.11}$$

where the first subscript $(\cdot)_1$ and $(\cdot)_2$ denote the primary and secondary subdomains, respectively. The second subscript $(\cdot)_{\cdot,c}$ denotes the part of the subdomain that is in contact and $(\cdot)_{\cdot,i}$ denotes the rest of the subdomain. The blocks $K_{\cdot,\cdot}$ denote the respective stiffness matrices. The block $D$ represents the coupling between the Lagrange multipliers and the primal variable in the secondary subdomain. The block $M$ denotes the coupling between the Lagrange multipliers and the primal variable in the primary subdomain.

Owing to the biorthogonality property of the dual basis functions, the integral matrix $D$ become diagonal. Thus the discrete Lagrange multipliers can be eliminated by condensation at negligible computational cost,

$$\lambda = D^{-1}(r_{2,c} - K_{2,ci} u_{2,i} - K_{2,cc} u_{2,c}). \tag{3.12}$$

By substituting Equation (3.12) into Equation (3.11), we obtain a simplified linear system of equations that contains only the primal variable DOFs,

$$\begin{bmatrix} K_{1,ii} & K_{1,ic} & & \\ K_{1,ci} & K_{1,cc} & -MD^{-1}K_{2,ii} & -MD^{-1}K_{2,cc} \\ & & K_{2,ii} & K_{2,ic} \\ & M^\intercal & & D^\intercal \end{bmatrix} \begin{bmatrix} u_{1,i} \\ u_{1,c} \\ u_{2,i} \\ u_{2,c} \end{bmatrix} = \begin{bmatrix} r_{1,i} \\ r_{1,c} - MD^{-1} r_{2,c} \\ r_{2,i} \\ 0 \end{bmatrix} \tag{3.13}$$

This condensed system (i.e., Equation (3.13)) is positive definite. Therefore, state-of-art iterative solution techniques, such as multigrid methods [5], are applicable. As a post-processing step, the dual Lagrange multipliers can be recovered from the displacement following Equation (3.12).

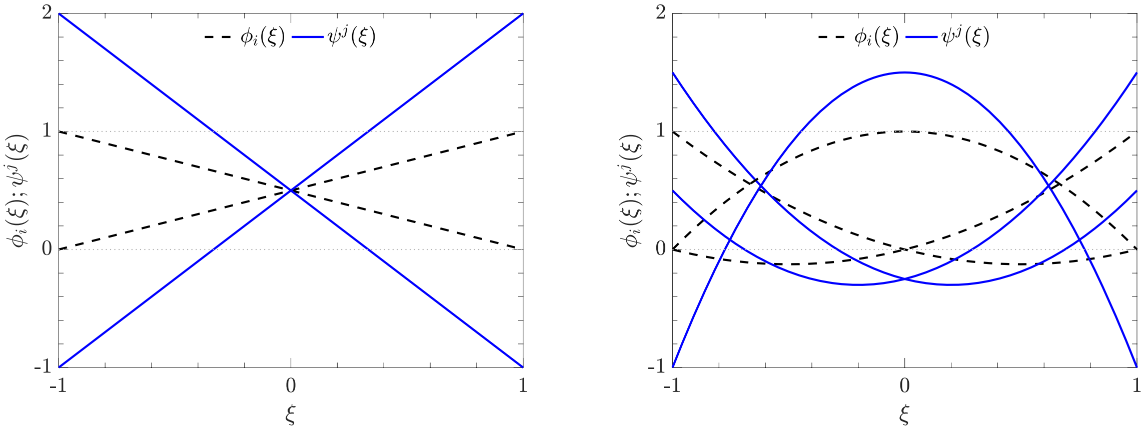
### 3.3.3 Numerical Results

In this section, we include some initial numerical results to demonstrate the improvements brought by the dual-mortar approach. Note that the results shown in Sections 3.3.3.2 and 3.3.3.3 correspond to a mortar-based diffusion problem in two subdomains. Extending the current dual mortar implementation to mortar-based contact problems is readily achievable and is an ongoing process during this fiscal year.

#### 3.3.3.1 Dual basis functions

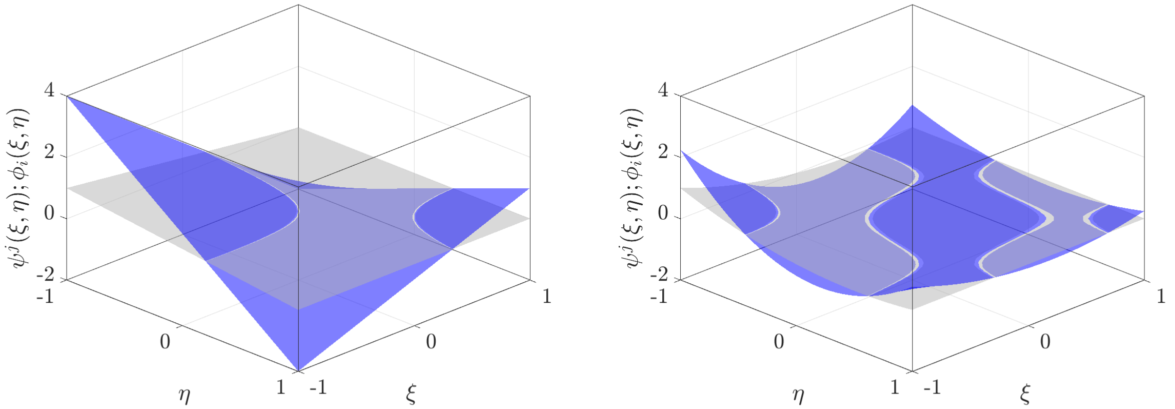
The dual basis functions described in Section 3.3 have been implemented in libMesh in order to support the mortar-based mechanical contact problems in BISON and other problems that employ Lagrange multipliers to enforce continuity conditions.

As an illustration, we show examples of the standard Lagrange basis functions and corresponding dual-basis functions in Figures 3.6 and 3.7. Here, linear and quadratic basis functions are plotted for 1D (in Figure 3.6) and 2D (in Figure 3.7) cases, respectively. All the basis functions are plotted in the reference frame (in terms of $\xi$ and $\eta$) for an element that has unit size in the physical frame. Note that the dual basis functions keep the original properties of the original basis functions (e.g., order of approximation). The maximum values of the dual basis functions are not unit or zero at the grid points. However, the sum of all the local nodal shape functions remain to be 1 at the grid points. This ensures that the values of vector $\boldsymbol{\lambda}$ (see Equation (3.5)) are equal to the true value of the discretized Lagrange multiplier variable.



**(a)** **(b)**

Figure 3.6: Standard and dual 1D Lagrange basis functions. Standard basis functions (denoted by $\phi_i(\xi)$) are shown in black. Dual basis functions (denoted by $\psi_j(\xi)$) are shown in blue. (a) Linear basis functions and (b) quadratic basis functions.



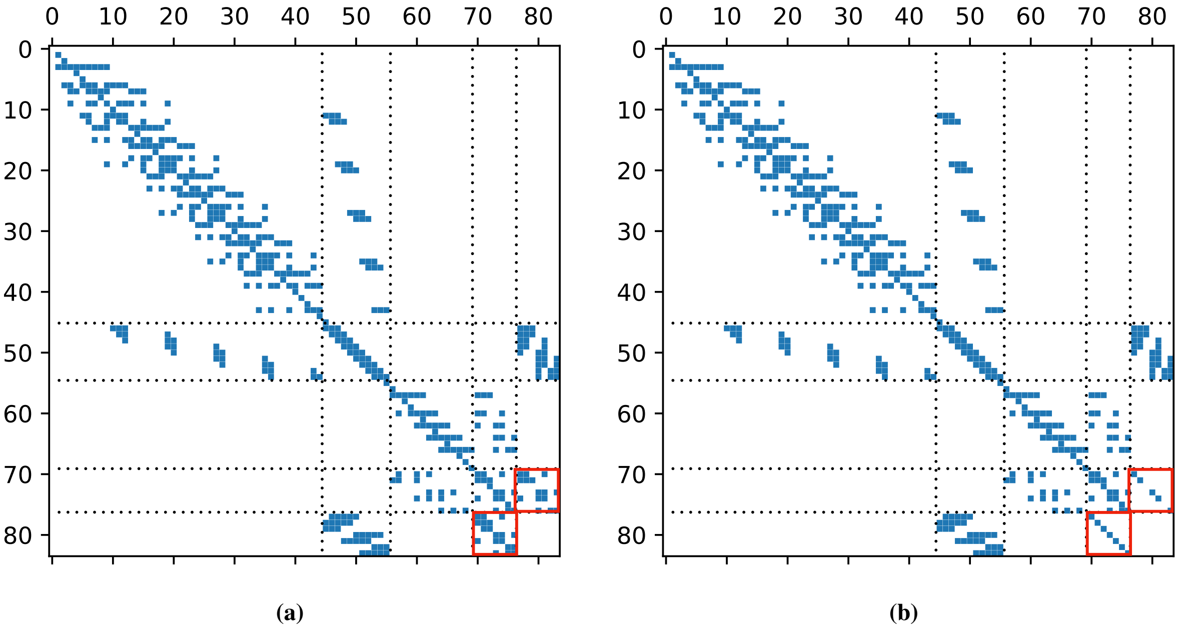
**(a)**

**(b)**

Figure 3.7: Standard and dual 2D Lagrange basis function at node $(\xi,\eta) = (-1,-1)$. Standard basis functions (denoted by $\phi_i(\xi,\eta)$) are shown in gray. Dual basis functions (denoted by $\psi_j(\xi,\eta)$) are shown in blue. (a) Linear basis function and (b) quadratic basis function.

#### 3.3.3.2 System simplification

As described in Section 3.3.2, part of the global system matrix can be simplified by using the dual basis functions for the Lagrange multipliers. As a verification, we show the system matrix pattern that uses the dual basis functions (see Figure 3.8(b)) and compare it with the matrix pattern that uses the standard basis functions (see Figure 3.8(a)). Note here, we permute the rows and columns of the global matrix *a priori* in order to transform it into a block system that matches the matrix system described earlier in Equation (3.11).



**(a)** **(b)**

Figure 3.8: Pattern of system matrix using (a) standard basis function and (b) dual basis function for the Lagrange multipliers. The submatrix marked by the red rectangle corresponds to the $D$ matrix in Equation (3.11). Note that $D$ is diagonalized after using the dual basis function.

From Figure 3.8, note that the submatrix $D$ and $D^\intercal$ (marked by the red rectangles) become strict diagonals when the dual basis functions are utilized for the Lagrange multipliers. Therefore, inverting $D$ is trivial and condensation of the Lagrange multipliers following Equations (3.12) and (3.13) does not add to the overall computation load. This is particularly advantageous when the system becomes large.

#### 3.3.3.3 Performance improvement

As an initial implementation, we designed a new preconditioner interface to realize the condensation step as described in Section 3.3.2. This interface act on the original system (see Equation (3.11)), computes the condensed system matrix and right-hand-side (see Equation (3.13)), and pass the condensed system to the solver package (i.e., PETSc). Owing to the positive definiteness property of the condensed system, a wide range of robust solvers/preconditioners can be utilized.

In Table 3.2, we compare the convergence behavior in terms of the $L^2$-norm of the total residual for one time step. Here, we utilize BoomerAMG as a preconditioner. It can be seen from Table 3.2 that the solver converges within 12 linear iterations with condensation. However, without condensation, the solver significantly stagnates and convergence is unlikely.

| Iteration Number | With Condensation | Without Condensation |
|---|---|---|
| 0 | 5.776114e+00 | 5.776114e+00 |
| 1 | 9.919965e-01 | 1.063330e+00 |
| 2 | 7.040158e-01 | 1.061189e+00 |
| 3 | 6.224583e-01 | 1.055026e+00 |
| 4 | 8.487218e-02 | 1.052544e+00 |
| 5 | 5.599553e-02 | 1.040641e+00 |
| 6 | 5.444969e-02 | 1.039412e+00 |
| 7 | 2.375990e-03 | 1.038976e+00 |
| 8 | 1.423721e-03 | 1.038760e+00 |
| 9 | 6.599375e-04 | 1.038540e+00 |
| 10 | 3.948003e-06 | 1.038321e+00 |
| 11 | 9.718431e-11 | 1.038101e+00 |
| 12 | 6.567258e-14 | 1.037882e+00 |

Table 3.2: Convergence behavior in terms of the $L^2$-norm of the total residual for one time step using BoomerAMG as a preconditioner. Results are shown for system matrices with and without condensation.

### 3.3.4 Conclusions and Future Work

A dual mortar method has been implemented in the MOOSE framework. Improvements have been demonstrated via a diffusion problem with equal value constraint. Specifically, the submatrix $D$, which couples the Lagrange multipliers with the primal variable, is diagonalized. This enables static condensation of the Lagrange multipliers (see Section 3.3.3.2) and results in a positive definite system which can be efficiently solved using a wider range of robust solvers/preconditioners (see Section 3.3.3.3). Future work in this topic includes extending current solver interface to mortar-based mechanical contact problems and conduct verification and scaling studies of the approach.

# Bibliography

[1] Alexander Popp and WA Wall. Dual mortar methods for computational contact mechanics–overview and recent developments. *GAMM-Mitteilungen*, 37(1):66–84, 2014.

[2] I Temizer, P Wriggers, and TJR Hughes. Three-dimensional mortar-based frictional contact treatment in isogeometric analysis with nurbs. *Computer Methods in Applied Mechanics and Engineering*, 209:115–128, 2012.

[3] Barbara I Wohlmuth. A mortar finite element method using dual spaces for the lagrange multiplier. *SIAM journal on numerical analysis*, 38(3):989–1012, 2000.

[4] Barbara Wohlmuth. Variationally consistent discretization schemes and numerical algorithms for contact problems. *Acta Numerica*, 20:569–734, 2011.

[5] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

# 4 BISON Ease-of-Use Improvements

The BISON fuel performance code is being developed by INL to analyze a range of fuel forms, from LWRs to metallic fuel and TRISO fuel, among others, while retaining the flexibility to conduct simulations in 1D through 3D geometries. These capabilities continue to create the potential for new BISON users. Originally developed as a research and development tool, BISON presents usage challenges for potential new users who may be more accustomed to commercially developed finite element analysis (FEA) codes. The efforts completed for this milestone report demonstrate concrete steps to reduce the complexity of the text-based BISON input file. Our development efforts completed under this milestone include:

1. Improvement of existing mechanics quantity output input file `action` capabilities
2. Incorporation of best practice settings for common nuclear material models into predefined `action` blocks
3. Expansion of existing fuel rod specific output `action` capabilities
4. Increased transparency for these and other existing `action` input file blocks within the editor Atom.

These objectives build on the goals identified in previous FY-19 milestone reports. During FY-19 issues with expanding user support documentation were addressed and the need for improved ease-of-use within BISON input files was identified, with a focus on new and beginning BISON users. These needs are addressed by the development efforts listed above. In the following report section each of these focus areas are discussed, and this report concludes with a summary of potential future work to continue the accomplishments during this year.

## 4.1 Tensor Mechanics Master Action

Significant improvements were made to the *Tensor Mechanics Master Action* (TM-MA) to aid in user ease-of-use input file generation. The TM-MA is a set of actions that combine the effects of several other classes to provide a user-friendly, quick, and concise method of creating input files. These actions are created in the "background" to reduce the length and complexity of input files. Specifically, major changes in the TM-MA were made with the objective of creating outputs for tensor stress/strain variables.

### 4.1.1 Creation of New Output Quantity Classes

Prior to this improvement, mechanical stress/strain outputs were created via the process of an aux-variable being passed through an aux-kernel for use in a post-processor. This process had several disadvantages, foremost among which was the averaging of variables instead of getting exact values at quadrature points. In switching to obtaining material properties, values at individual quadrature points could be utilized, greatly increasing the accuracy. Additionally, the procedure significantly reduces the length and complexity of input files while retaining the previous functionality. In the refinement of material outputs for the TM-MA, four new material classes were created in MOOSE. In this section, the new classes are discussed below.

#### 4.1.1.1 RankTwoCartesianComponent

RankTwoCartesianComponent, as shown in Table 4.1, is a material model used to extract components of a Rank-2 tensor within a Cartesian coordinate system. RankTwoCartesianComponent takes as arguments the values of the indices $i$ and $j$ for the single tensor component that is saved in a material property.

| Name | Output | Function |
|---|---|---|
| Strain | Total_Strain | Sum of elastic and plastic strains |
| Stress | Stress | Force applied to a material, divided by cross-sectional area |
| Elastic_Strain | Elastic_Strain | Measure of the deformations which are recoverable |
| Plastic_Strain | Plastic_Strain | Measure of the deformations which are permanent |
| Creep_Strain | Creep_Strain | Measure of the deformations which occur due to creep stress |
| Creep_Stress | Creep_Stress | Application of persistent mechanical stress |

Table 4.1: Material based stresses/strains created by the RankTwoCartesianComponent class

This model can be used regardless of the coordinate system used in the simulation, as shown by Equation 4.1. The user's selection of the indices returns only a component of a Rank-2 tensor, regardless of the coordinate system. Therefore this operation is applicable to any coordinate system currently accepted by MOOSE.

$$\sigma_{ij} = \begin{bmatrix} \sigma_{00} & \sigma_{01} & \sigma_{02} \\ \sigma_{10} & \sigma_{11} & \sigma_{12} \\ \sigma_{20} & \sigma_{21} & \sigma_{22} \end{bmatrix} \tag{4.1}$$

#### 4.1.1.2 RankTwoCyclindricalComponent

RankTwoCyclindricalComponent is only applicable for cylindrical coordinates. For returned output quantity values to retain a physical meaning, these calculations should include an element of rotational symmetry. While it is possible to calculate these output quantities in any coordinate system, the traditional choice of cylindrical coordinates (RZ in MOOSE) has been adopted as a requirement for this class. As seen in Table 4.2, this class returns the scalar value of a Rank-2 tensor in the direction of the axis of rotation. By default the axis of rotation is assumed to be the y-axis. Redefining the axis of rotation is possible for the user via input of two points which define the vector on which rotational symmetry can be applied. These two points are referred to as cylindrical_axis_point_1 and cylindrical_axis_point_2 respectively. Following the convention of the right-hand rule, the direction of the axis starts at "point 1" and ends at "point 2".

| Name | Output | Function |
|---|---|---|
| Axial | Stress<br>Strain<br>Plastic_Strain<br>Creep_Strain<br>Elastic_Strain | Calculates the scalar value of a Rank-2 tensor in the direction of the axis specified by the user |
| Hoop | Stress<br>Strain<br>Plastic_Strain<br>Creep_Strain<br>Elastic_Strain | Calculates the value of a Rank-2 tensor along the hoop direction of a cylinder |
| Radial | Stress<br>Strain | Calculates the scalar component for a Rank-2 tensor in the direction of the normal vector from the user defined axis of rotation |

Table 4.2: Material classes created by the class `RankTwoCylindricalComponent`. *Note: These outputs are only valid and recognized when used with the MOOSE RZ coordinate system.*

#### 4.1.1.3 RankTwoDirectionalComponent

For an extremely customizable output quantity option, the class `RankTwoDirectionalComponent` allows the determination of stresses and strains in a user specified direction. As shown Table 4.3, this class returns a scalar value of a Rank-2 tensor in the direction specified by the user. The input file parameter `direction` is a point in space that, when taken with respect to the origin, gives the direction in which the calculation will be computed. Following standard conventions, the direction vector starts at the origin and ends at the specified point given by the user.

| Name | Output | Function |
|---|---|---|
| Directional | Stress<br>Strain | Calculates the scalar value of a Rank-2 tensor in the direction selected by the user |

Table 4.3: Material classes created by the `RankTwoDirectionalComponent`

#### 4.1.1.4 RankTwoInvariant

Stresses and Strains which do not change under coordinate transformations are known as invariant quantities. The class `RankTwoInvariant` involves the calculation of such invariant quantities as described in Table 4.4.

| Name | Output | Function |
|---|---|---|
| VonMises | Stress | Calculates the von Mises measure for a Rank-2 tensor |
| Effective | Plastic_Strain<br>Creep_Strain | Calculates an effective scalar measure of a Rank-2 tensor |
| Hydrostatic | Stress | Calculates the hydrostatic scalar of a Rank-2 tensor |
| L2norm | Stress<br>Strain<br>Elastic_Strain<br>Plastic_Strain<br>Creep_Strain | Calculates the L2 normal of a Rank-2 tensor |
| Volumetric | Strain | Computes the volumetric strain |
| FirstInv | Stress<br>Strain | Calculates the first invariant of the specified Rank-2 tensor |
| SecondInv | Stress<br>Strain | Calculates the second invariant of the specified Rank-2 tensor |
| ThirdInv | Stress<br>Strain | Calculates the third invariant of the specified Rank-2 tensor |
| Triaxiality | Stress | Finds the ratio of the hydrostatic measurement to the von Mises measurement |
| MaxShear | Stress | Calculates the maximum shear stress for a Rank-2 tensor |
| Intensity | Stress | Calculates the stress intensity for a Rank-2 tensor |
| Max Principal | Stress<br>Strain | Calculates the largest principal value for symmetric tensor |
| Mid Principal | Stress<br>Strain | Calculates the second largest principal value for a symmetric tensor |
| Min Principal | Stress<br>Strain | Calculates the smallest principal value for a symmetric tensor |

Table 4.4: Material based stresses/strains created by the `RankTwoInvariant` class. *Note: The effective strain measure,*`Effecitve`*, is different than* `effective_plastic_strain` *or* `effective_plastic_strain` *calculated elsewhere in BISON. These inelastic strains are determined by calculating strain integrated over time.*

These quantities depend on symmetric matrices when expressed in tensor notation. As shown in Equation 4.1.1.4, the properties of a symmetric matrix enable these calculations.

$$\mathbf{A}' = \mathbf{Q} \cdot \mathbf{A} \cdot \mathbf{Q}^T \tag{4.2}$$

where $\mathbf{A}$ is any symmetric tensor with $\mathbf{Q}$ and $\mathbf{Q}^T$ are the transformation and conjugate transpose transformation tensors respectively. In the situation where $\mathbf{A}' = \mathbf{A}$, the transformation is invariant.

## 4.2 Nuclear Materials

Additional BISON ease-of-use improvements were created through `NuclearMaterials` actions. These actions help to significantly reduce the overall length of input files by creating the commonly used material blocks in the background via the action mechanism. Additionally these actions aid new and beginning BISON users by directly incorporating best practice settings that are routinely utilized by more experienced BISON users and developers.

Currently there are four `NuclearMaterials` used within BISON, with plans for at least two more in the near future. External BISON users have adopted the success of this formalism and created additional `NuclearMaterials` which do not fall within the scope of this report. The existing Nuclear Material actions are discussed in detail below.

### 4.2.1 Light Water Reactor Material Actions

LWR simulations within BISON can be broken into two simple components, fuel and cladding. The initial versions of the `NuclearMaterials` actions included simplifying assumptions, which are discussed below.

#### 4.2.1.1 UO$_2$ Fuel

The action `NuclearMaterialUO2` deals with the commonly used material blocks for LWR Uranium Oxide fuels. The current implementation reduces input file length by creating classes internally which only deal with elastic stress calculations. Expansion of the action capabilities to accommodate inelastic models are planned for a future development versions. All of the material model input file blocks, generated with the default parameter settings, are shown in Table 4.5.

| **Created Classes** | **Pre-Set Parameters** | **Block Name** |
|---|---|---|
| ComputeIsotropicElasticityTensor | poissons_ratio = 0.345<br>youngs_modulus = 2.0e11 | fuel_elasticity_tensor |
| ComputeFiniteStrainElasticStress | | fuel_elastic_stress |
| ComputeThermalExpansionEigenstrain | thermal_expansion_coeff = 10.0e-6<br>eigenstrain_name = fuel_thermal_strain | fuel_thermal_expansion |
| UO2VolumetricSwellingEigenstrain | burnup_function = burnup<br>initial_fuel_density = 10431.0<br>eigenstrain_name = fuel_volumetric_strain | fuel_volumetric_swelling |
| UO2RelocationEigenstrain | burnup_function = burnup<br>linear_heat_rate_function = q<br>gap = 160.0e-6<br>eigenstrain_name = fuel_relocation_strain | fuel_relocation |
| Sifgrs | gbs_model = false<br>burnup_function = burnup | fuel_fission_gas_release |
| Density | initial_fuel_density = 10431.0 | fuel_density |
| ThermalFuel | | fuel_thermal |

Table 4.5: Material model classes created by the `NuclearMaterialUO2` action

#### 4.2.1.2 Zirconium Alloy Cladding

The action `ZirconiumAlloy` generates the necessary material classes for common elastic LWR simulations, specifically Zirconium alloy. This action is designed for use with only LWR simulations which use creep mechanics models. Simulations with plasticity are not currently accommodated by this action but will be implemented in future development efforts. All of the material model input file blocks, generated with the default parameter settings, are shown in Table 4.6.

| **Created Classes** | **Pre-Set Parameters** | **Block Name** |
|---|---|---|
| ComputeIsotropicElasticityTensor | poissons_ratio = 0.3<br>youngs_modulus = 7.5e10 | clad_elasticity_tensor |
| ComputeMultipleInelasticStress | tangent_operator = elastic<br>inelastic_models = clad_zrycreep | clad_stress |
| ZryCreepLimbackHoppeUpdate | absolute_tolerance = 1e-10<br>max_iterations = 50<br>fast_neutron_flux = fast_neutron_flux<br>fast_neutron_fluence = fast_neutron_fluence | clad_zrycreep |
| ZryThermalExpansionMATPROEigenstrain | burnup_function = burnup<br>eigenstrain_name = fuel_irradiation_strain | clad_thermal_expansion |
| Density | density = 6551.0 | clad_density |
| HeatConductionMaterial | thermal_conductivity = 16.0<br>specific_heat = 330.0 | clad_thermal |

Table 4.6: Material classes created by the `NuclearMaterialZirconiumAlloy` action

#### 4.2.1.3 Combined LWR NuclearMaterials

As seen in Examples 4.1 and 4.2, LWR simulation input file lengths are greatly reduced while maintaining the same functionality. Future development efforts to add creep, plasticity, and inelastic material models will further enhance the user ease-of-use in effectively and quickly creating input files. The meta-action which creates the Tensor-Mechanics Master Action, as discussed in Section 4.3.2, is utilized in the example 4.2, thus eliminating the need for additional lines the user needs to include within an input file.

Example 4.1: Example of the input file length before reduction via NuclearMaterialsUO2 and NuclearMaterialZirconiumAlloy

```
[Materials]
  [fuel_volumetric_swelling]
    type = UO2VolumetricSwellingEigenstrain
    block = pellet_type_1
    burnup_function = burnup
    temperature = temp
    eigenstrain_name = fuel_volumetric_swelling_eigenstrain
    initial_fuel_density = 10233
  []
  [fuel_thermal]
    type = ThermalFuel
    block = pellet_type_1
    temp = temp
    burnup_function = burnup
    thermal_conductivity_model = NFIR
  []
  [fuel_elastic_stress]
    type = ComputeFiniteStrainElasticStress
    block = pellet_type_1
  []
  [fuel_thermal_expansion]
    type = ComputeThermalExpansionEigenstrain
    block = pellet_type_1
    temperature = temp
    stress_free_temperature = 300
    thermal_expansion_coeff = 10e-6
    eigenstrain_name = fuel_thermal_eigenstrain
  []
  [fuel_elasticity_tensor]
    type = UO2ElasticityTensor
    block = pellet_type_1
    temperature = temp
  []
  [fuel_relocation]
    type = UO2RelocationEigenstrain
    block = pellet_type_1
    burnup_function = burnup
    diameter = .00819
    linear_heat_rate_function = q
    gap = 1.7e-4 #diameteral gap
    relocation_activation1 = 5000
    burnup_relocation_stop = .035
    eigenstrain_name = fuel_relocation_eigenstrain
  []
  [clad_thermal]
    type = HeatConductionMaterial
    block = 1
    thermal_conductivity = 16.0
    specific_heat = 330.0
  []
  [clad_creep_stress]
    type = ZryCreepLimbackHoppeUpdate
    block = 1
    temperature = temp
    fast_neutron_flux = fast_neutron_flux
    fast_neutron_fluence = fast_neutron_fluence
  []
  [clad_inelastic_stress]
    type = ComputeMultipleInelasticStress
    block = 1
    tangent_operator = elastic
    inelastic_models =’clad_creep_stress’
  []
  [clad_elasticity_tensor]
    type = ZryElasticityTensor
    block = 1
  []
  [clad_irradiation_growth]
      type = ZryIrradiationGrowthEigenstrain
      block = 1
      fast_neutron_fluence = fast_neutron_fluence
      eigenstrain_name = clad_irradiation_growth_eigenstrain
  []
  [clad_thermal_expansion]
    type = ZryThermalExpansionMATPROEigenstrain
    block = 1
    stress_free_temperature = 300
    temperature = temp
    eigenstrain_name = ’clad_thermal_eigenstrain’
  []
  [fission_gas_release]
    type = Sifgrs
    diff_coeff_option = 2
    transient_option = 2
    block = pellet_type_1
    temp = temp
    burnup_function = burnup
    grain_radius = grain_radius
    gbs_model = true
  []
  [clad_density]
    type = Density
    block = clad
    density = 6551.0
  []
  [fuel_density]
    type = Density
    block = pellet_type_1
  []
[]
```

Example 4.2: Example of the input file length reduction achieved with the use of the `NuclearMaterialsUO2` and `NuclearMaterialZirconiumAlloy` actions

```
[NuclearMaterials]
  [UO2]
    [fuel]
      block = pellet_type_1
      stress_free_temperature = 300
      diameter = 0.00819
      gap = 1.7e-4
      burnup_relocation_stop = 0.035
      thermal_model = NFIR
      grain_radius = grain_radius
      gbs_model = true
      initial_fuel_density = 10233
      burnup_function = burnup
      diff_coeff_option = 2
      transient_option = 2
      generate_output = ’hydrostatic_stress stress_xx stress_yy stress_zz
         vonmises_stress’
    []
  []
  [ZirconiumAlloy]
    [clad]
      block = 1
      stress_free_temperature = 300
      generate_output = ’stress_xx stress_yy stress_zz vonmises_stress
         creep_strain_xx
        creep_strain_yy creep_strain_xy’
    []
  []
[]
```

### 4.2.2 Metallic Fuel

#### 4.2.2.1 UPuZr Fuel

The action `NuclearMaterialUPuZr` deals with the commonly used material blocks for metallic fuels when these actions are utilized in the input file. All of the material blocks, generated with the default parameter settings, are shown in Table 4.7.

| **Created Classes** | **Pre-Set Parameters** | **Block Name** |
|---|---|---|
| UPuZrFissionRate | | fission_rate |
| UPuZrBurnup | burnup_name = burnup | burnup |
| UPuZrElasticityTensor | | fuel_elasticity_tensor |
| ComputeMultipleInelasticStress | | fuel_inelastic_stress |
| UPuZrCreepUpdate | max_inelastic_increment = 1e-2 | fuel_upuzrcreep |
| ComputeThermalExpansionEigenstrain | thermal_expansion_coeff = 1.18e-5<br>eigenstrain_name = fuel_thermal_strain | fuel_thermal_expansion |
| UPuZrGaseousEigenstrain | bubble_number_density = N_bubbles<br>interconnection_initiating_porosity = 0.23<br>interconnection_terminating_porosity = 0.25<br>bubble_number_density = N_bubbles | gas_swelling |
| BurnupDependentEigenstrain | eigenstrain_name = solid_swelling_strain | solid_swelling |
| ThermalUPuZr | | metal_fuel_thermal |
| Density | initial_fuel_density = 15800 | fuel_density |
| FgrUPuZr | fission_rate = fission_rate<br>critical_porosity = 0.24<br>fractional_fgr_initial = 0.8<br>fractional_fgr_post = 1.0 | fission_gas_release |

Table 4.7: Material classes created by the `NuclearMaterialUPuZr` action

#### 4.2.2.2 HT9 Cladding

The action `NuclearMaterialHT9` deals with the commonly used material blocks for high-Cr martensitic steel (HT9) metallic fuel cladding. All of the material blocks, generated with the default parameter settings, are shown in Table 4.8.

| **Created Classes** | **Pre-Set Parameters** | **Block Name** |
|---|---|---|
| ComputeIsotropicElasticityTensor | poissons_ratio = 0.236<br>youngs_modulus = 1.88e11 | clad_elasticity_tensor |
| ComputeMultipleInelasticStress | tangent_operator = nonlinear<br>inelastic_models = clad_ht9creep | clad_stress |
| FastNeutronFlux | factor = 1.0 | fast_flux |
| HT9CreepUpdate | | clad_ht9creep |
| ComputeThermalExpansionEigenstrain | thermal_expansion_coeff = 1.2e-5<br>eigenstrain_name = clad_thermal_strain | clad_thermal_expansion |
| ThermalHT9 | | clad_thermal |
| Density | density = 7874.0 | clad_density |

Table 4.8: Material classes created by the `NuclearMaterialHT9` action

#### 4.2.2.3 Combined Metallic Fuel NuclearMaterials

As seen in Examples 4.3 and 4.4, metallic fuel simulation input file lengths are greatly reduced. The meta-action utilized in the combined input reductions outlined in Example 4.2, and discussed in 4.3.2, is not used in the Example 4.4.

Example 4.3: Example of the input file before reduction via NuclearMaterialsUPuZr and NuclearMaterialHT9

```
[Materials]
  [./fission_rate]
    type = UPuZrFissionRate
    rod_linear_power = power_history
    axial_power_profile = axial_peaking_factors
    pellet_radius =  2.195e-03
    X_Zr = 0.225
    X_Pu_function = 0.163
    block = pellet
  [../]
  [./burnup]
    type = UPuZrBurnup
    initial_X_Pu = 0.163
    density = 15800
    block = pellet
  [../]
  [./fuel_elasticity_tensor]
    type = UPuZrElasticityTensor
    X_Zr = 0.225
    X_Pu = 0.163
    block = pellet
  [../]
  [./fuel_inlastic_stress]
    type = ComputeMultipleInelasticStress
    inelastic_models = ’fuel_upuzrcreep’
    block = pellet
  [../]
  [./fuel_upuzrcreep]
    type = UPuZrCreepUpdate
    block = pellet
    max_inelastic_increment = 1e-2
  [../]
  [./fuel_thermal_expansion]
    type = ComputeThermalExpansionEigenstrain
    block = pellet
    thermal_expansion_coeff = 1.18e-5
    stress_free_temperature = 295.0
    eigenstrain_name = fuel_thermal_strain
  [../]
  [./gas_swelling]
    type = UPuZrGaseousEigenstrain
    eigenstrain_name = gas_swelling_eigenstrain
    bubble_number_density = 5e17
    interconnection_initiating_porosity = 0.29
    interconnection_terminating_porosity = 0.31
    output_properties = ’porosity gaseous_porosity’
    block = pellet
  [../]
  [./solid_swelling]
    type = BurnupDependentEigenstrain
    eigenstrain_name = solid_swelling_eigenstrain
    block = pellet
    swelling_name = ’solid_swelling’
  [../]
  [./metal_fuel_thermal]
    type = ThermalUPuZr
    block = pellet
    X_Zr = 0.225
    X_Pu = 0.163
    spheat_model = savage
    thcond_model = lanl
  [../]
  [./fuel_density]
    type = Density
    block = pellet
  [../]
  [./Fission_Gas_Release]
    type = FgrUPuZr
    block = pellet
    critical_porosity = 0.30
    fractional_fgr_initial = 0.4
    fractional_fgr_post = 0.8
  [../]
  [./clad_elasticity_tensor]
    type = ComputeIsotropicElasticityTensor
    youngs_modulus = 1.88e11
    poissons_ratio = 0.236
    block = clad
  [../]
  [./clad_stress]
    type = ComputeMultipleInelasticStress
    inelastic_models = ’clad_ht9creep’
    block = clad
  [../]
  [./fast_flux]
    type = FastNeutronFlux
    block = clad
    factor = 2.47e19
  [../]
  [./clad_ht9creep]
    type = HT9CreepUpdate
    block = clad
  [../]
  [./thermal_expansion]
    type = ComputeThermalExpansionEigenstrain
    block = clad
    thermal_expansion_coeff = 1.2e-5
    stress_free_temperature = 295.0
    eigenstrain_name = clad_thermal_strain
  [../]
  [./clad_thermal]
    type = ThermalHT9
    block = clad
  [../]
  [./clad_density]
    type = Density
    block = clad
    density = 7874.0
  [../]
[]
```

Example 4.4: Example of the input file length reduction via `NuclearMaterialsUPuZr` and `NuclearMaterialHT9`

```
[NuclearMaterials]
  [./UPuZr]
    [./fuel]
      block = pellet
      rod_linear_power = power_history
      axial_power_profile = axial_peaking_factors
      pellet_radius =  2.195e-03
      stress_free_temperature = 295.0
      bubble_number_density = 5e17
      interconnection_initiating_porosity = 0.29
      interconnection_terminating_porosity = 0.31
      X_Zr = 0.225
      X_Pu = 0.163
      initial_X_Pu = 0.225
      initial_X_Zr = 0.163
      spheat_model = savage
      thcond_model = lanl
      critical_porosity = 0.30
      fractional_fgr_initial = 0.4
      fractional_fgr_post = 0.8
    [../]
  [../]
  [./HT9]
    [./clad]
      block = clad
      factor = 2.47e19
      stress_free_temperature = 295.0
    [../]
  [../]
[]
```

## 4.3 Additional Refinements Within NuclearMaterials

### 4.3.1 Stress Free Temperature

In updating the Nuclear material classes, several classes need to be altered so that they inherited from `ComputeThermalExpansionEigenstrainBase`. Prior to this update, these classes calculated the stress free temperature as an average. With the change stress free temperature became a material property and could be calculated at individual quadrature points. In addition, some material classes which were still performing Fortran class were updated to modern C++ implementations.

### 4.3.2 Tensor Mechanics Master Action Creation via NuclearMaterials

A common feature integrated into all Nuclear Materials is the ability to create the `TensorMechanicsMasterAction`. This feature is only activated in a Nuclear Material block if the user specifies the "generate_output" parameter. When TM-MA is constructed within the NuclearMaterials, in addition to the stress/strain outputs listed in the "generate_output," the eigenstrain names are created and used by their respective material blocks. This behind the scenes designation allows for a seamless implementation of the tensor mechanics eigenstrain without any naming errors.

### 4.3.3 Thermal Components

A final `NuclearMaterials` refinement is the ability to create the thermal kernels and temperature variable in the background. This class is a standalone feature within `NuclearMaterials` in that it doesn't require a specific block. The "fuel_block" is requested since the class `NeutronHeatSource` is only applied to fuel components. Future refinements will allow this to be created within the respective fuel blocks within `NuclearMaterials`. Further clarification on the classes and default settings are shown in Table 4.9.

| **Created Classes** | **Pre-Set Parameters** | **Name** |
|---|---|---|
| MooseVariable | family = LAGRANGE<br>control_tags = Variables | temperature |
| ConstantIC | variable = temperature | initial_temperature |
| HeatConduction | variable = temperature<br>extra_vector_tags = ref | heat |
| HeatConductionTimeDerivative | variable = temperature<br>extra_vector_tags = ref | heat_ie |
| NeutronHeatSource | variable = temperature<br>extra_vector_tags = ref | heat_source |

Table 4.9: Material classes created by the `NuclearMaterialThermal action`

## 4.4 Standard LWR Fuel Rod Output Action

Building on the work started in the previous year [1], the Standard Outputs Action was expanded during this physical year to reflect best practices for analyzing results and to expand the usage options. As with the NuclearMaterials actions and TensorMechanics action described above, the StandardLWRFuelRodOutputs action is intended to simplify a BISON input file by reducing the number of input file lines devoted to output quantity creation. This action creates up to 13 scalar and vector postprocessor quantities, and the user may elect to generate output quantities associated with only the fuel pellets, only the cladding, or, in the default case, both the fuel and cladding rod components. Equally importantly, the StandardLWRFuelRodOutputs action helps to enforce best practices in fuel rod performance analyses by ensuring the creation of the set of output quantities deemed commonly required for these analyses. As the name suggests, this action is intended only for use with LWR simulations. Development efforts in other project focus areas, however, have involved the creation of a standard outputs action specific to metallic fuel applications.

### 4.4.1 Rod Average Burnup Calculation

Due to the period over which LWR assessment cases have been added to the BISON assessment repository, significant variations in the method used to calculate the rod average burnup can be found. Many of the BISON LWR assessments simulations relied on an element averaging post-processors while others queried the burnup function directly. A third group of simulations neglected to include the average burnup as an output quantity. Despite these shortcomings, the average burnup of the fuel rod is a useful quantity in comparing BISON simulation results to experimental data. Rod average burnup is also used to compare the results of different BISON fuel rod simulations. In order to ensure consistent comparisons, a common method of calculating the fuel rod average burnup quantity for BISON simulations is necessary.

The specifically designed RodAverageBurnup post-processor was selected as the most appropriate output quantity, and was added to the existing StandardLWRFuelRodOutput action to ensure consistent, best-practices usage throughout the BISON LWR assessment repository. Since this quantity is directly associated with the fuel, this quantity is only created if the user elects to generate fuel pellet associated output quantities with the StandardLWRFuelRodOutput action.

The RodAverageBurnup post-processor directly queries the BurnupFunction input file block and thus ensures consistent calculation of the average rod burnup value, regardless of the simulation mesh order and the input file Burnup block settings. The other prevalent method of calculating the average burnup, the ElementAverage post-processor, was sensitive to the order and family settings in the Burnup input file block. This sensitivity could have resulted in different calculated output values of the rod average burnup for the same BurnupFunction settings. The expansion of the StandardLWRFuelRodOutput action to use the RodAverageBurnup post-processor eliminates these concerns.

### 4.4.2 Expanded Plenum Temperature Calculation Options

A second development focus area within the expansion of the StandardLWRFuelRodOutput action is the plenum temperature value calculation and output generation. The plenum temperature value is used to calculate the quantity of fission gas released throughout the BISON simulation; therefore, this value plays a crucial role in the proper simulation of the fuel rod performance behavior.

Similar to the rod average burnup calculation, multiple methods for calculating the plenum temperature were used within the BISON LWR assessment repository. The development efforts here focused on expanding the StandardLWRFuelRodOutput action to allow for the calculation of the plenum temperature with one of two different post-processor types, replacing the previous simple post-processor option used with the StandardLWRFuelRodOutput action. An additional benefit of this development work was the standardization of the code and terminology associated with the plenum temperature calculation methods.

Many of the BISON LWR assessment cases use the SideAverageValue post-processor, which computes a simple average of all the interior cladding and and exterior pellet surfaces exposed to the plenum. This method can lead to an overestimation of the plenum temperature because the cladding surfaces which are above the fuel pellet stack more strongly influence the temperature of the plenum gases by nature of being exposed to more of these gases.

The PlenumTemperature action was developed to correct the potential overestimation by introducing weighting factors to the plenum temperature calculation. These weighting factors are proportional to the amount of plenum gas volume each surface touches: the cladding surfaces above the fuel pellet stack will be in direct contact with more of the plenum gas volume than will the lower cladding surfaces and external pellet surfaces in the fuel stack.

To enable the user to calculate and generate the plenum temperature value with either of these two options, the StandardLWRFuelRodOutput action was expanded to include so-called meta action capabilities. These additional capabilities enable the standard outputs action to generate the secondary PlenumTemperature action. A new input file enumeration option was added to require the user to select between these different methods of calculating the plenum temperature value.

#### 4.4.2.1 Improved Nomenclature Consistency

The expansion of the StandardLWRFuelRodOutput action into a meta action uncovered inconsistencies in the nomenclature used to calculate the plenum temperature value with the two post-processor approaches describe above. Due to the weighting factors employed in the PlenumTemperature action approach, variations in the surface designations as 'inner' and 'outer' produced significantly different plenum temperature values in even a simplified regression test simulation. The significant variation in the calculated temperature value is the result of a mesh node search algorithm in the code: if the search does not return an opposing node within a gap region, the first surface is assumed to be above the pellet stack and is assigned a higher weighting factor. Incorrect settings of the surface designations will result in incorrect applications of the weighting factor.

As a result of this work, updates were made to both the in-code documentation strings, the user-facing web-based documentation, and the regression tests. All three of these updates will improve the user experience by clarifying the best practices use of the PlenumTemperature action, both with and without the standard LWR outputs action. Assessments are ongoing to determine if code updates to calculation of the plenum temperature in the case of discrete pellet meshes are compatible with the BISON validation base.

#### 4.4.2.2 Extension of the Standard Outputs Action to Discrete Meshes

Originally designed for only smeared pellet meshes, the StandardLWRFuelRodOutput action was expanded to discrete pellet meshes in this FY. In discrete pellet meshes, pellet top and bottom surfaces, in addition to previously utilized pellet exterior and cladding interior surfaces, influence the calculation of the plenum temperature values. Through the extension of the standard LWR outputs action to include the PlenumTemperature action, the ability to handle discrete pellet meshes is also being added to the StandardLWRFuelRodOutput action, widening the range of LWR fuel performance simulations that can utilize this action.

## 4.5 Action Syntax Expansion

The introduction of additional MOOSE/BISON actions enable users to choose a more concise input file syntax. The action system was designed to cover common use cases for BISON, as described in the previous sections. The simplicity does, however, come with the cost of a reduced flexibility. To prevent the simplified action syntax from being a dead end for users who need to set up complex models which go beyond the scope of the action system, a method was needed to convert the action syntax to the *low level* MOOSE object input syntax.

Actions within the MOOSE framework and its derived applications are a programmatic way of setting up objects from any of the many MOOSE systems. These can include elements such as Variables, Kernels, Materials, and Boundary Conditions, to name just a few. By design the action system is opaque to the end user. Which objects are created through each action can only be determined through the documentation or by investigating the source code. Knowing exactly what parameters are passed to the created objects is even more difficult, as the parameters passed to the low level MOOSE objects can be a complex function of the parameters passed into the simplified action syntax.

The goal of this development effort was to remove some of the opacity of the MOOSE action system and to provide an automated way to expand the simplified action syntax into the verbose low level MOOSE object syntax. The basis of this desired capability comes from a custom MOOSE problem class, `DumpObjectsProblem`, that intercepts the creation of each object performed by a selected action. Object type and parameter set as created by the action are recorded and serialized into the MOOSE input file format. In order to make this functionality more accessible a plugin[2] for the Atom text editor[3] was developed. The `action-explode-moose` defines a hotkey that transforms the action block under the cursor into the verbose low level MOOSE object syntax. The expanded syntax should result in a simulation behavior that is indistinguishable from the original input using the action syntax. Once expanded, the verbose input file offers all the flexibility and customization of the underlying MOOSE objects. Any object in the expanded syntax can be replaced with a customized version and parameters can be tweaked that are not available in the simplified syntax.

The action expansion was achieved by parsing the input file into a tree structure from which we extract the input file block path (e.g. `Modules/TensorMechanics/Master/all`) and the character range the identified block occupies in the input text. A valid MOOSE executable was detected in the current path (or above) from which MOOSE is launched with command line options to activate the `DumpObjectsProblem` and supply the input file block path of the action to be expanded. The MOOSE executable will return the verbose input text which is then inserted at top level near the original simplified action, which is removed from the input text. An example of an input file snippet before and after the action expansion is shown in 4.5 and 4.6 respectively.

Example 4.5: Example of an input file excerpt with the simplified action syntax before expanding the Modules/TensorMechanics/Master/block2 action subblock.

```
[Modules/TensorMechanics/Master]
  [./block1]
    strain = FINITE
    add_variables = true
    #block = 1
  [../]
  [./block2]
    strain = SMALL
    add_variables = true
    block = 2
  [../]
```

```
[]
```

Example 4.6: Example of an input file excerpt after expanding one of the `Modules/TensorMechanics/Master` subblocks.

```
[Modules/TensorMechanics/Master]
  [./block1]
    strain = FINITE
    add_variables = true
    block = 1
  [../]
[]

[Kernels]
  [./TM_block20]
    type = StressDivergenceTensors
    block = 2
    component = 0
    displacements = 'disp_x disp_y'
    variable = disp_x
  [../]
  [./TM_block21]
    type = StressDivergenceTensors
    block = 2
    component = 1
    displacements = 'disp_x disp_y'
    variable = disp_y
  [../]
[]

[Materials]
  [./block2_strain]
    type = ComputeSmallStrain
    block = 2
    displacements = 'disp_x disp_y'
  [../]
[]

[Variables]
  [./disp_x]
    type = MooseVariable
  [../]
  [./disp_y]
    type = MooseVariable
  [../]
[]
```

## 4.6 Summary and Future Work

The majority of the efforts for FY20 were focused on improvements to the BISON ease-of-use. These improvements included a significant effort to simplify input file creation for users through `action` blocks. These `action` blocks maintain full functionality of the BISON code while greatly reducing the complexity and length of input files created by users. As a result of these BISON input file creation refinements, code was refined to allow for more accurate calculations of items such as stresses, strains, burnup and plenum temperature. These improvements help to reduce some of the barriers to BISON use for both new and existing users.

Additionally, work utilizing the editor Atom allowed input file creation to become more transparent with respect to the parameters which users have control over. When combined with the newly implemented `action` blocks, user creation of BISON input files was simplified and improved.

Future developments and improvements for BISON's ease-of-use will continue to focus on ensuring the input files represent the current capabilities of BISON and on assisting end users with harnessing these capabilities. To achieve these goals we have identified the following tasks for future work:

1. Continued improvement of existing and new mechanics quantity output input file helper `action` capabilities
2. Expansion of common nuclear material model sets into and within predefined `action` blocks
3. Refinement of existing and new fuel rod specific output capabilities
4. Further utilization and simplification of `action` input file blocks within the editor Atom.

All of these future development goals will build on the work completed under this milestone to update, expand, and improve the BISON ease-of-use.

# Bibliography

[1] Stephanie A. Pitts, Russell J. Gardner, Al Casagranda, Benjamin W. Spencer, Daniel J. VanWasshenova, Dylan J. McDowell, Giuseppe Rota, and Richard L. Williamson. Improvements to bison validation base for lwr fuel. Technical Report CASL-U-2019-1895-000, Idaho National Laboratory, 2019.

[2] Daniel Schwen. action-explode-moose. `https://github.com/dschwen/action-explode-moose/`, 2020.

[3] GitHub.com. Atom - A hackable text editor for the 21st Century. `https://atom.io`, 2020.

# 5 Frictional contact with multiple inelastic materials

## 5.1 Background

During the development of a new LWR Validation case for the IFA-629.4 high burnup fuel case it was found that thermomechanical frictional contact in combination with smeared cracking inelastic materials led to small timesteps and convergence issues [1]. In this work we attempt to address these issues using Automatic Differentiation (AD) to improve Jacobian calculations for the fuel material models. This work includes converting all of the smeared cracking models over to AD as well as the $UO_2$ relocation eigenstrain model. We then use the new AD models to perform a comparison of an AD versus non-AD simulation of the IFA-629.4 high-burnup fuel case. The main accomplishment of this work is to begin using AD to run a complex nuclear fuel problem in BISON. The full benefits of AD cannot be realized in this work until the entire simulation is fully AD, which would require the conversion of the following models to AD: thermomechanical frictional contact, fission gas release, fuel burnup, and cladding material.

## 5.2 Class Conversion to Automatic Differentiation

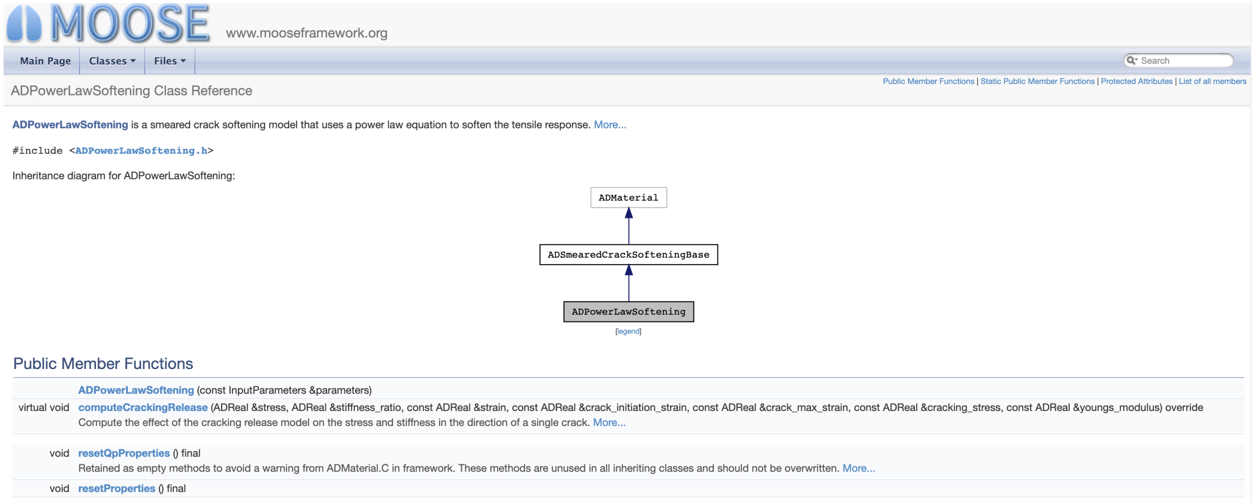
Traditionally, the physical model's Jacobian or its required approximation for preconditioning have been obtained by manually computing on- and off-diagonal Jacobian entries. This can often result in an adequate Jacobian approximation but it forces the developer to employ significant effort in making sure the often cumbersome Jacobian expressions are correct. In addition, obtaining an accurate system Jacobian by means of analytical expressions can be very challenging for non-smooth physics (e.g., frictional contact and material models with abrupt stiffness changes).

### 5.2.1 Automatic Differentiation Classes

In MOOSE, automatic differentiation, or *AD*, classes compute the Jacobian of the system automatically by taking advantage of the MetaPhysicL library features. By defining scalars and tensors with the correct *AD* C++ type, the Jacobian is computed numerically without developer intervention. The advent of automatic differentiation in the MOOSE ecosystem has caused some same-purpose classes to have duplicated documentation, header, and source files. This causes undesirable code duplication. AD and non-AD classes that are used by the analyst for the same purpose can easily get out of sync and, additionally, will require duplicated maintenance. For these reasons, we have recently started to give preference to templated *AD-non-AD* classes which, avoiding code duplication, resolve to the right types by instantiating a template parameter *is_ad* to *true*, if automatic differentiation is used, and *false*, if the traditional hand-coded version is used. In this case, and to have a quicker development turnaround, we first converted a hierarchy to separate *AD* classes.

### 5.2.2 MOOSE Changes

A typical parent class for materials in MOOSE is *Material*. This class has an *AD* counterpart named *ADMaterial*. Starting from *ADMaterial*, we created a new hierarchy of automatic differentiation-enabled classes, which are depicted in Fig. 5.1. Parent class *ADSmearedCrackSofteningBase* and child classes *ADAbruptSoftening*, *ADExponentialSoftening*, and *ADPowerLawSoftening* inherit from *ADMaterial* and compute the


MOOSE
www.mooseframework.org
Main Page
Classes
Files
Search
Public Member Functions | Static Public Member Functions | Protected Attributes | List of all members
ADMaterial
ADSmearedCrackSofteningBase
ADPowerLawSoftening
[legend]

ADPowerLawSoftening Class Reference

ADPowerLawSoftening is a smeared crack softening model that uses a power law equation to soften the tensile response. More...

```
#include <ADPowerLawSoftening.h>
```

Inheritance diagram for ADPowerLawSoftening:

Public Member Functions

```
              ADPowerLawSoftening (const InputParameters &parameters)
virtual void  computeCrackingRelease (ADReal &stress, ADReal &stiffness_ratio, const ADReal &strain, const ADReal &crack_initiation_strain, const ADReal &crack_max_strain, const ADReal &cracking_stress, const ADReal &youngs_modulus) override
              Compute the effect of the cracking release model on the stress and stiffness in the direction of a single crack. More...

        void  resetQpProperties () final
              Retained as empty methods to avoid a warning from ADMaterial.C in framework. These methods are unused in all inheriting classes and should not be overwritten. More...
        void  resetProperties () final
```

Figure 5.1: Doxygen documentation depicting hierarchy of ADPowerLawSoftening

cracking release stress that is used by *ADComputeSmearedCrackingStress* to calculate the finite-strain-based stress tensor for a smeared cracking material. Instead of the non-AD types *Real*, *RealVectorValue*, *RankTwoTensor*, and *RankFourTensor*, the converted classes use *ADReal*, *ADRealVectorValue*, *ADRankTwoTensor*, and *ADRankFourTensor*, which incorporate Jacobian information. Failure to choose the right type for a contributor to the residual results in Jacobian information loss (i.e., inaccuracies in the computation of on- and off-diagonal terms).

Partial details of C++ inheritance are given in the excerpt below.

```
/**
 * ADSmearedCrackSofteningBase is the base class for a set of models that define the
 * softening behavior of a crack under loading in a given direction.
 * These models are called by ADComputeSmearedCrackingStress, so they
 * must have the compute=false flag set in the parameter list.
 */
class ADSmearedCrackSofteningBase : public ADMaterial
{
}

/**
 * ADAbruptSoftening is a smeared crack softening model that abruptly
 * drops the stress upon crack initiation and relies on automatic
 * differentiation. It is for use with ADComputeSmearedCrackingStress.
 */
class ADAbruptSoftening : public ADSmearedCrackSofteningBase
{
}

/**
 * ADExponentialSoftening is a smeared crack softening model that
 * uses an exponential softening curve. It is for use with
 * ADComputeSmearedCrackingStress and relies on automatic
 * differentiation.
 */
```

```
class ADExponentialSoftening : public ADSmearedCrackSofteningBase
{
}

/**
 * ADPowerLawSoftening is a smeared crack softening model that
 * uses a power law equation to soften the tensile response.
 * It is for use with ADComputeSmearedCrackingStress and uses
 * automatic differentiation.
 */
class ADPowerLawSoftening : public ADSmearedCrackSofteningBase
{
}

/**
 * ADComputeSmearedCrackingStress computes the stress for a finite strain
 * material with smeared cracking
 */
class ADComputeSmearedCrackingStress : public ADComputeMultipleInelasticStress
{
}
```

For all the cases tested, the use of automatic differentiation produces a Jacobian of the same quality or better than its hand-coded counterpart.

## 5.3 BISON Computational Model

A brief summary of the finite-element model and base irradiation power history from [1] will be given here. The simulation model consists of a 2D axisymmetric finite element mesh containing dished fuel pellets initially separated from the cladding. Only the base irradiation is studied in this work approximated by constant power operation at 20, 25, 24, 21, and 17 kW/m for cycles of length 6088, 7327, 6237, 6383, and 6176 hours. The fuel material model includes creep and smeared cracking while the cladding material only includes creep. The thermal contact model captures conductance through the gap as well as increased conductance as a function of contact pressure. Mechanical frictional contact between fuel and cladding is modeled with kinematic constraints normal to the contact surface and penalty contact for sliding tangential to the contact surface.

Only the fuel models are converted to AD. Initially, only the tensor mechanics smeared cracking material model and its associated softening models were going to be converted to AD. However, AD and non-AD variables such as stress and eigenstrain cannot be easily shared between tightly-coupled material models. This required fuel materials associated with eigenstrains, such as relocation and volumetric swelling, to also be converted to AD. Material converters were used to convert regular material properties produced by non-AD material models, such as the SIFGRS fission gas release model, to AD material properties required by the AD versions of the relocation and volumetric swelling models. With these changes, we were able to run simulations using AD material models for the fuel.

In order to evaluate AD, four different simulations were set up. Two different material models for the fuel were used for the simulations, the original model and a simplified elastic model. An AD version of the original model was also created and tested with two different nonlinear solver typed, Preconditioned Jacobian Free Newton Krylov (PJFNK) and Newton. The original model is described in [1] and contains smeared cracking with exponential softening. A simplified model was also created that includes only elastic materials and frictionless contact. Results are also given for two AD versions of the original fuel model that use different nonlinear solvers. The first version uses a Jacobian Free Newton Krylov Precondtioner (PJFNK) which only requires an approximate Jacobian as a preconditioner. The second version uses a full Newton method to solve the nonlinear system which assumes the Jacobian supplied by the material models is correct. There are pros and cons to both methods' nonlinear solution methods. For a fully AD version of the entire model, the Newton solver will provide the fastest solve time by using an exact Jacobian to the linear solves. For a partially converted simulation with only some models being AD, PJNFK will provide a more robust solution method because Jacobian is assumed to only be approximate. However, the PJNFK should benefit from a few of the material models producing exact Jacobians.

### 5.3.1 Results and Comparisons

The timing results are shown in Figure 5.2 where the top plot gives the simulation wall time for timing and 5.3 for iteration count per step. The light gray lines mark the power ramps for each of the four power cycles of the base irradiation. The original model from [1] is shown by the orange line and labeled *Original*. As expected, the simplified model shown by the blue line and labeled *Simplified* solves the base irradiation faster than the original model and with fewer nonlinear iterations. The AD versions of the original model do no perform as well and fail due to convergence issues before the base irradiation is complete. The AD fuel model using PJFNK shown by the green line and labeled *AD PJFNK* makes it to the fourth power cycle before running into convergence issues and failing due to the timestep falling below 1 second. Each of the power cycles lead to a large drop in the timestep for all of the simulations but the *AD PJFNK* takes several more time steps to recover from these. At about 400 days, the iteration count for the *AD PJFNK* simulation increases to around 8 nonlinear solves per timestep and never drops back down. This could be due to Jacobian inaccuracies caused by one of the non-AD fuel models — such as fission gas release or burnup — and needs to be investigated further. The AD fuel model using Newton shown by the red line and labeled *AD Newton* makes it slightly past the first power cycle before its timestep drops below one second. The nonlinear iteration count for the *AD Newton* simulation remains high from the start of the simulation. However, the *AD Newton* only requires a single linear solve per nonlinear iteration because the system size is small enough to use a direct linear solver.

As more models and functions are converted over to AD, it is expected that AD will provide a more robust solution strategy compared to its non-AD counterparts. This, in effect, should increase the timestep limits and reduce the overall solve time. AD should also provide a more robust solution strategy allowing us to run simulations that are currently not possible to run.

## 5.4 Creep/Plasticity Model Systems

To further investigate the effect of converting models over to automatic differentiation we set up two small model systems. The first one (Figure 5.4) consists of a single material block that is being compressed by prescribing a sinusoidal displacement curve with linearly increasing amplitude to its right surface.

The second system (Figure 5.5) consists of a two-block setup, with a smaller block repeatedly impacting a larger block with the aforementioned prescribed sinosoidal displacement. This second setup repeatedly goes in and out of contact with increasing forces exhibited by the two blocks. The advantages of small model systems are the rapid turnaround between successive simulations with different parameters and model implementations as well as the reduction to the smallest set of model components.

### 5.4.1 Results and Comparisons

We set up both models with either a set of conventional model classes with hand-coded approximate Jacobians and a set of fully AD-enabled model classes with perfect Jacobians. The single block model system converges

Figure 5.2: Simulation time in days versus (top) total compute time and (bottom) time-step size. The light gray lines mark the power ramps for each of the four power cycles of the base irradiation.

Figure 5.3: Simulation time in days versus (top) total linear iterations per step and (bottom) number of nonlinear iterations per step. The light gray lines mark the power ramps for each of the four power cycles of the base irradiation.

Figure 5.4: Final state of the single-block model system after four compression cycles.

Figure 5.5: Visualization of the creep (left) and plastic (right) strains of the impacting block in our model system, during the fourth impact cycle.

with both the AD as well as the non-AD models. The most striking observation in Figure 5.6 is how much better the AD version (blue curve) of the model performs compared to the non-AD version (red curve). Despite the added overhead of the dual number computation, the AD calculation surpasses the non-AD calculation as soon as the deformation gets sufficiently large. Notably the AD convergence behavior is much more consistent and does not show jumps in wall time due to difficult to converge steps. We also analyzed the non-AD model behavior when only one of the inelastic models is enabled, either creep or plasticity. We observe that the computational cost of the combined model is substantially larger than the sum of the individual inelastic models (green curves).

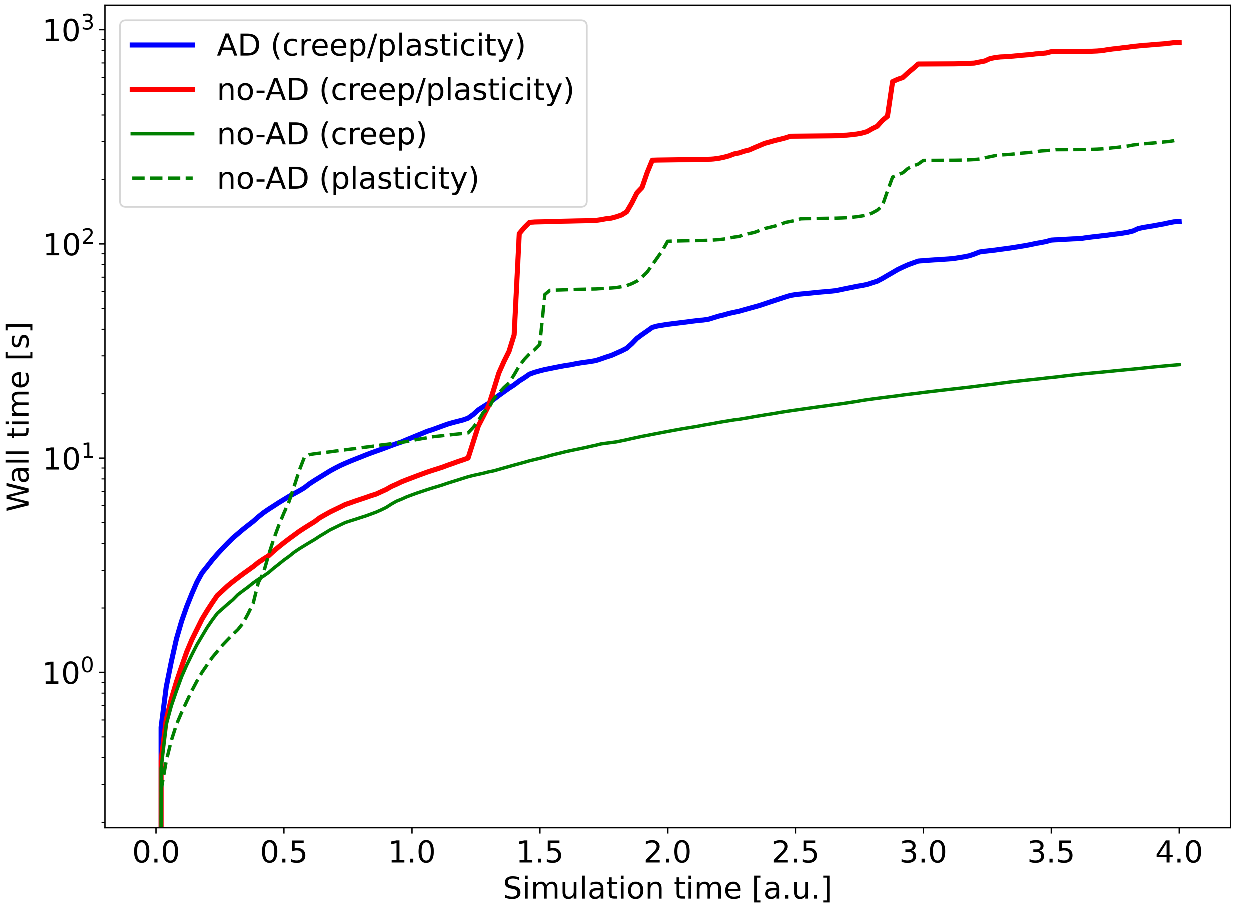


Figure 5.6: Wall time vs. simulation time in the single-block model system, showing a substantial performance advantage of the AD implementation over the non-AD implementation of the mechanics models.

The two-block impact configuration with contact only converges when using the AD versions of the mechanics models. Neither with the penalty nor the mortar contact formulation the non-AD version of the model converges beyond the first time the two blocks come into contact.

For all AD model systems, we were able to successfully solve using the Newton method, which realizes the performance gains provided by the perfect AD Jacobian. All non-AD models only ran as far as they did using PJFNK.

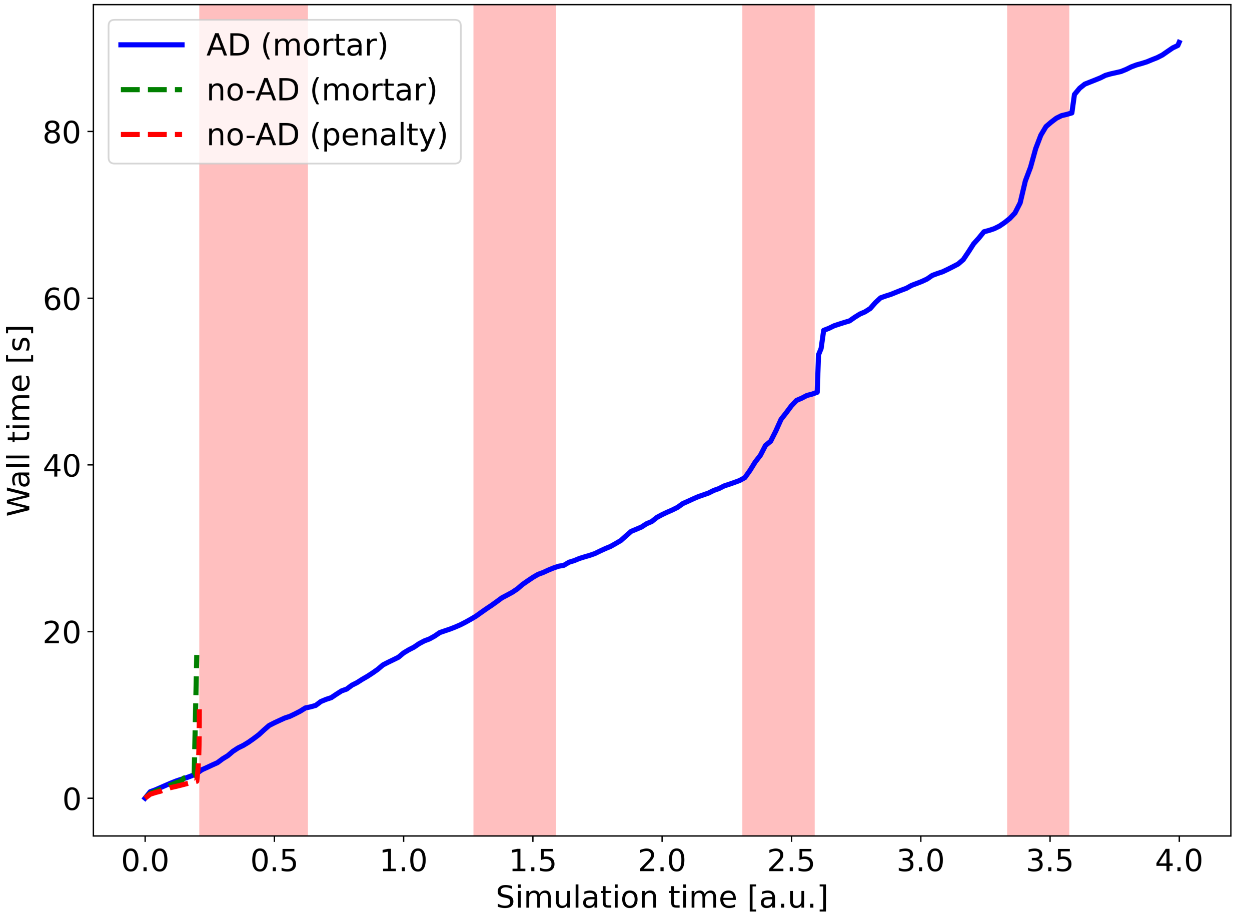


Figure 5.7: Wall time vs. simulation time in the repeated impact creep/plasticity model system. Times the two blocks are in contact are shaded light red. The fully AD-converted system shows good convergence properties, whereas the conventional models fail as soon as contact is established for the first time.

## 5.5 Conclusions

Converting models to use automatic differentiation improves convergence and overall computation speed in simulations with large deformation. Simulations with multiple inelastic models in particular benefit from the perfect Jacobian provided by AD. Going forward we will use the templated `is_ad` approach to combine the codebases of the non-AD and AD implementations of our models removing potential code duplication and reducing the maintenance overhead for MOOSE/BISON.

# Bibliography

[1] S.A. Pitts, R.J. Gardner, A. Casagranda, B.W. Spencer, D.J. VanWasshenova, D.J. McDowell, G. Rota, and R.L. Williamson. Improvements to bison validation base for lwr fuel l3:fmc.fuel.p1909. Technical Report CASL-U-2019-1895-000, Idaho National Laboratory, August 2020.

# 6 BISON Windows Executables

To lower the barrier for novice BISON users who are running the Microsoft Windows operating system, we investigated the possibility of providing a native Windows binary distribution of the BISON code. The underlying MOOSE framework is officially only supported on macOS and Linux operating systems (OSs). Both of those OSs follow the so-called Posix standard and offer a set of largely compatible APIs as well as easily available toolchains for compiling and linking the C++source code files. In particular the availability of the GNU Compiler Collection or the compatible LLVM based Clang compiler as well as GNU make as the build tool are key features of these OSs.

## 6.1 Native Windows Executables

Several compatibility layers centered around the *Minimalist GNU for Windows* (MinGW) project exist on Windows to provide a build environment that should be largely compatible with a genuine Posix OS. We chose the *MSYS2* project [1], which is up to date, well maintained, and features the pacman package manager for easy installation and updates. The MSYS2 repositories provide several patched and precompiled dependencies of MOOSE and libmesh that greatly simplify the effort of porting to Windows.

While some successes had been reported in the past for compiling PETSc and libMesh on Windows, we found the process to be less than straightforward. Building PETSc required a custom configuration, which has now been added to the MOOSE repository. A file path-mangling issue needed to be resolved by patching build configuration files using a Python script, also checked in to the MOOSE repository. Among the several patches submitted to libMesh and MOOSE were:

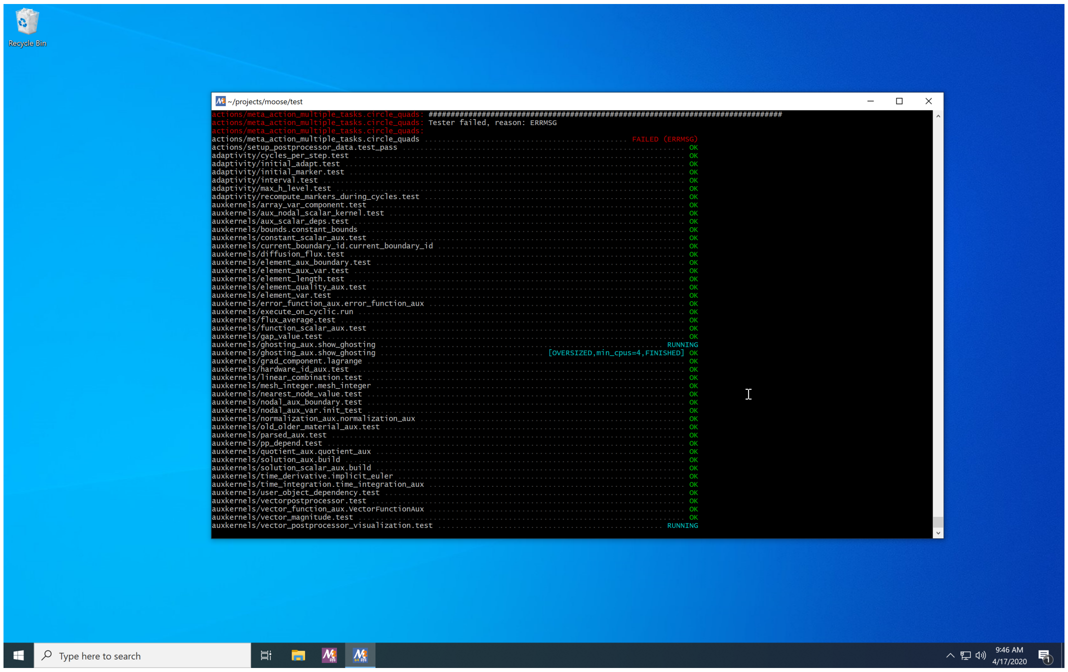
- Use of Windows APIs for several filesystem functions, such as file deletion, path resolution, and file meta data retrieval. These calls were wrapped in MOOSE utility functions, minimizing the amount of platform-specific code.
- Detect and work around unimplemented features, such as just-in-time compilation for parsed functions
- Tweaks to the build system to ensure correct directory path formats

We have thoroughly documented the required setup steps for MSYS2 and the build process for PETSc, libmesh, and MOOSE on the mooseframework.org website at `https://mooseframework.org/getting_started/installation/msys2.html`.

We have been able to produce native Windows MOOSE executable that pass a large fraction of the MOOSE test suite and can be executed from a vanilla Windows command prompt. Further work will be needed to automate the remaining manual steps required to produce a linked executable. Issues remain linking MOOSE-derived applications

### 6.1.1 Regular expression library

In an effort to improve portability of the MOOSE code base, we have worked on replacing the external *pcre* (Perl Compiled Regular Expressions) library with built-in C++11 regular epressions.



```
actions/meta_action_multiple_tasks.circle_quads: ######################################################################
actions/meta_action_multiple_tasks.circle_quads: Tester failed, reason: ERRMSG
actions/meta_action_multiple_tasks.circle_quads:
actions/meta_action_multiple_tasks.circle_quads ........................................ FAILED (ERRMSG)
actions/setup_postprocessor_data.test_pass .......................................................... OK
adaptivity/cycles_per_step.test ..................................................................... OK
adaptivity/initial_adapt.test ....................................................................... OK
adaptivity/initial_marker.test ...................................................................... OK
adaptivity/interval.test ............................................................................ OK
adaptivity/max_h_level.test ......................................................................... OK
adaptivity/recompute_markers_during_cycles.test ..................................................... OK
auxkernels/array_var_component.test ................................................................. OK
auxkernels/aux_nodal_scalar_kernel.test ............................................................. OK
auxkernels/aux_scalar_deps.test ..................................................................... OK
auxkernels/bounds.constant_bounds ................................................................... OK
auxkernels/constant_scalar_aux.test ................................................................. OK
auxkernels/current_boundary_id.current_boundary_id .................................................. OK
auxkernels/diffusion_flux.test ...................................................................... OK
auxkernels/element_aux_boundary.test ................................................................ OK
auxkernels/element_aux_var.test ..................................................................... OK
auxkernels/element_length.test ...................................................................... OK
auxkernels/element_quality_aux.test ................................................................. OK
auxkernels/element_var.test ......................................................................... OK
auxkernels/error_function_aux.error_function_aux .................................................... OK
auxkernels/execute_on_cyclic.run .................................................................... OK
auxkernels/flux_average.test ........................................................................ OK
auxkernels/function_scalar_aux.test ................................................................. OK
auxkernels/gap_value.test ........................................................................... OK
auxkernels/ghosting_aux.show_ghosting ...................................................... RUNNING
auxkernels/ghosting_aux.show_ghosting ................................. [OVERSIZED,min_cpus=4,FINISHED] OK
auxkernels/grad_component.lagrange .................................................................. OK
auxkernels/hardware_id_aux.test ..................................................................... OK
auxkernels/linear_combination.test .................................................................. OK
auxkernels/mesh_integer.mesh_integer ................................................................ OK
auxkernels/nearest_node_value.test .................................................................. OK
auxkernels/nodal_aux_boundary.test .................................................................. OK
auxkernels/nodal_aux_var.init_test .................................................................. OK
auxkernels/normalization_aux.normalization_aux ...................................................... OK
auxkernels/old_older_material_aux.test .............................................................. OK
auxkernels/parsed_aux.test .......................................................................... OK
auxkernels/pp_depend.test ........................................................................... OK
auxkernels/quotient_aux.quotient_aux ................................................................ OK
auxkernels/solution_aux.build ....................................................................... OK
auxkernels/solution_scalar_aux.build ................................................................ OK
auxkernels/time_derivative.implicit_euler ........................................................... OK
auxkernels/time_integration.time_integration_aux .................................................... OK
auxkernels/user_object_dependency.test .............................................................. OK
auxkernels/vectorpostprocessor.test ................................................................. OK
auxkernels/vector_function_aux.VectorFunctionAux .................................................... OK
auxkernels/vector_magnitude.test .................................................................... OK
auxkernels/vector_postprocessor_visualization.test ......................................... RUNNING
```

Figure 6.1: An initial build of a native Windows MOOSE executable is shown to pass a large number of integration tests.

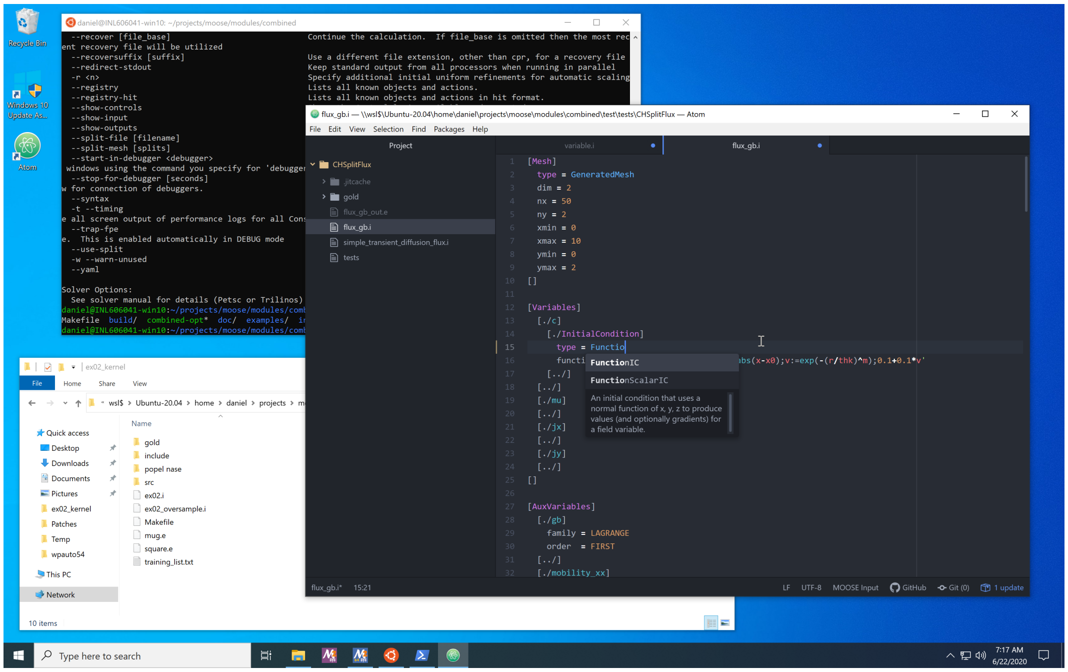
### 6.1.2 Windows Subsystem for Linux

During the work on the *BISON on Windows* milestone, Microsoft unveiled the next generation of its Windows Subsystem for Linux (WSL) 2. WSL is a compatibility layer for Windows that allows the installation of Linux distribution packages, providing a full-featured Linux terminal. The initial version WSL1 allowed to install prerequisites to build and execute PETSc, libMesh, and MOOSE. However, it exhibited severely degraded filesystem performance, making the installation and build process unwieldy and slow. Furthermore, working with files inside the WSL system required the use of Linux tools, such as editors and visualization software, rather than familiar Windows tools.

WSL2 promised better performance and we investigated building MOOSE and MOOSE-based applications on this improved system. Besides having to build the MPI *mpich2* from scratch, we were able to follow the Linux build instructions very closely and ended up with fully featured MOOSE app executables.

To improve the user experience, the MOOSE integration for autocompletion[2] in the *Atom* editor[3] was improved to work on Windows allowing the native Windows version of Atom to be used to edit files inside the WSL2 container. WSL2 files are made available in Windows through a virtual network share located at `\\wsl$\distribution\path`. The *moose-autocomplete* plugin for Atom obtains a description of valid application syntax by running a MOOSE app using the `-json` commandline switch. This causes the app to output JSON data describing all valid parameters for all available objects for the given app — rather than statically supplying a syntax description which would quickly go out of date as soon as objects in a MOOSE-based application are added or modified.

The MOOSE-autocomplete plugin detects whenever a file is located within a WSL2 container and uses the Windows wsl command to discover and execute a MOOSE app executable within the WSL2 container. The entire process is seamless to the user. Figure 6.2 shows a screenshot of a native Atom editor being used to edit a MOOSE inputfile within a WSL2 container with working autocompletion. We believe WSL2 to be a suitable avenue for making MOOSE available on Windows systems as it very closely maps to our current



```
  --recover [file_base]                 Continue the calculation.  If file_base is omitted then the most rec
ent recovery file will be utilized
  --recoversuffix [suffix]              Use a different file extension, other than cpr, for a recovery file
  --redirect-stdout                     Keep standard output from all processors when running in parallel
  -r <n>                                Specify additional initial uniform refinements for automatic scaling
  --registry                            Lists all known objects and actions.
  --registry-hit                        Lists all known objects and actions in hit format.
  --show-controls
  --show-input
  --show-outputs
  --split-file [filename]
  --split-mesh [splits]
  --start-in-debugger <debugger>
 windows using the command you specify for 'debugger
  --stop-for-debugger [seconds]
w for connection of debuggers.
  --syntax
  -t --timing
e all screen output of performance logs for all Cons
  --trap-fpe
e.  This is enabled automatically in DEBUG mode
  --use-split
  -w --warn-unused
  --yaml

Solver Options:
  See solver manual for details (Petsc or Trilinos)
daniel@INL606041-win10:~/projects/moose/modules/comb
Makefile  build/  combined-opt*  doc/  examples/  in
daniel@INL606041-win10:~/projects/moose/modules/comb
```

flux_gb.i — \\wsl$\Ubuntu-20.04\home\daniel\projects\moose\modules\combined\test\tests\CHSplitFlux — Atom
File
Edit
View
Selection
Find
Packages
Help
Project
variable.i
flux_gb.i
CHSplitFlux
.jitcache
gold
flux_gb_out.e
flux_gb.i
simple_transient_diffusion_flux.i
tests

```
[Mesh]
  type = GeneratedMesh
  dim = 2
  nx = 50
  ny = 2
  xmin = 0
  xmax = 10
  ymin = 0
  ymax = 2
[]

[Variables]
  [./c]
    [./InitialCondition]
      type = Functio
      functi                                    abs(x-x0);v:=exp(-(r/thk)^m);0.1+0.1*v'
    [../]
  [../]
  [./mu]
  [../]
  [./jx]
  [../]
  [./jy]
  [../]
[]

[AuxVariables]
  [./gb]
    family = LAGRANGE
    order  = FIRST
  [../]
  [./mobility_xx]
```

An initial condition that uses a normal function of x, y, z to produce values (and optionally gradients) for a field variable.

Figure 6.2: MOOSE input file editing with autocompletion on Windows. A native Atom editor with the moose-autocomplete plugin is used to edit an inputfile within a WSL2 container. A MOOSE app compiled within the same WSL2 container is queried to obtain the valid application syntax.

Linux based testing and development infrastructure.

# Bibliography

[1] Alexey Pavlov and Ray Donnelly. MSYS2 - Software Distribution and Building Platform for Windows. `https://www.msys2.org/`, 2020.

[2] Daniel Schwen. autocomplete-moose. `https://github.com/dschwen/autocomplete-moose/`, 2020.

[3] GitHub.com. Atom - A hackable text editor for the 21st Century. `https://atom.io`, 2020.

# 7 BISON User Support, Documentation Improvement, and Quality Assurance

## 7.1 User Support

### 7.1.1 Incorporating End-User Feedback

To aid in incorporating feedback, an ease-of-use survey was created to gauge the users' familiarity and desires with BISON. The questions were developed based on previous informal user inputs as well as a method of assessing current goals in BISON improvement. Input on the survey was a team effort, with numerous revisions of questions and wording considered. An anonymous survey solicitation was submitted to the BISON user group emails with the Qualtrics package utilized by the Idaho National Laboratory. The survey could be completed in under five minutes, and it frequently asked for users' clarifications on questions.
In this section, the user's background and overall satisfaction with BISON is discussed. Other results from the study are integrated throughout this report, in the specific areas which overlap the survey question. Special attention was paid to users' evaluations of documentation as noted in Section 7.2.

#### 7.1.1.1 User Background

As the lead question to the survey, Figure 7.1 shows the manner in which users became aware of BISON. This sampling shows word of mouth as the most frequent method of exposure. The personal recommendation of users for BISON is a positive factor; however, it shows areas of future improvement. As is discussed later in this report, this is often due to users coming from the same workplace/background. Increased scientific publications and conference proceedings are areas of planned exposure improvement in the future which will help develop a more diverse set of new users.

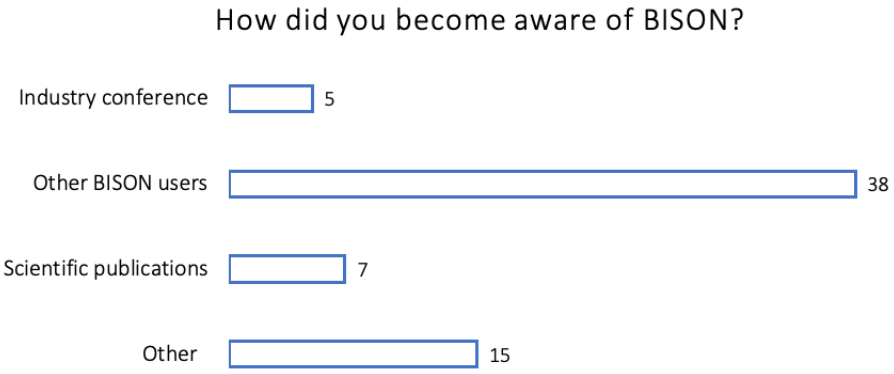


Figure 7.1: Manner in which users became aware of BISON.

With respect to the responses in which “other” was selected as the means of BISON introduction, around 50% of the 15 responses came from self-reported Idaho National Laboratory employees. This fraction of “other,”, as well as a percentage of “word of mouth,” should be taken into consideration to be correlated with the percentage of National Lab researchers currently using BISON discussed further in this report. Examining the user composition more deeply, the aspects of the workplace with respect to BISON usage were requested. Figure 7.2 shows the users’ self- described usage of BISON. Combining this information with the data from the user group email list, Figure 7.3 shows BISON is primarily used in research at the national labs. The second most common user base outside the lab system is academia. With Nuclear Energy University Program (NEUP) and Nuclear Energy Advanced Modeling and Simulation (NEAMS) funding often tied to software such as BISON, this helps in building a user base outside of the laboratory base. Lastly, the number of industry users is relatively low. This could be because BISON is free, making it highly accessible to university users. An area of future growth is to build a larger base of industry users as new technologies such as microreactors and TRISO fuel are further developed within BISON.

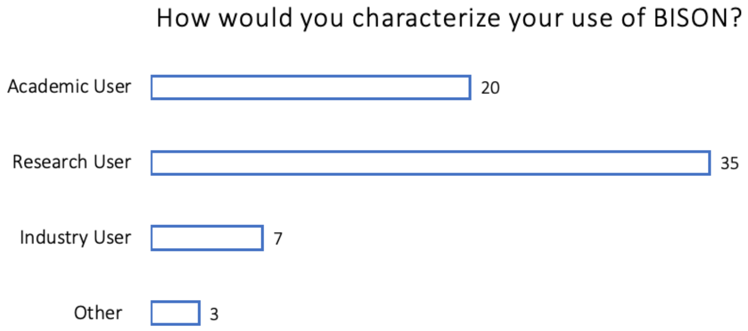


Figure 7.2: BISON user self-described usage.

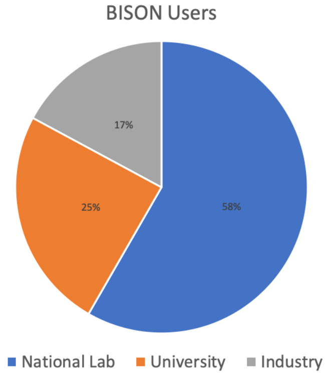


Figure 7.3: Relative percentages of BISON users.

In order of decreasing numbers, the national labs with registered users consist of

- Idaho National Laboratory (INL)
- Argonne National Laboratory (ANL)
- Oak Ridge National Laboratory (ORNL)
- Los Alamos National Laboratory (LANL)
- United States Nuclear Regulatory Commission (NRC)
- Sandia National Laboratory (SNL)
- Pacific Northwest National Laboratory (PNNL)
- Brookhaven National Laboratory (BNL)

as shown in Figure 7.4.

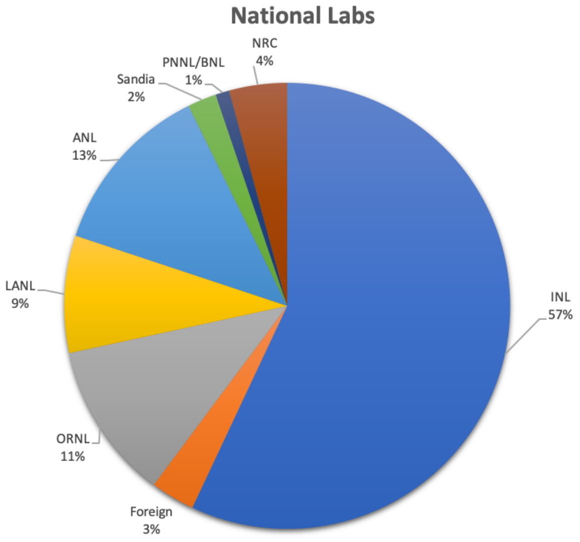


Figure 7.4: Relative percentages of national lab users.

This high INL user number statistic is to be expected since BISON is primarily a MOOSE-based software developed in-house, which aligns with the laboratory mission. Users from other national labs have similar research missions and this is reflected in the relative number of users (i.e., LANL has a similar nuclear energy focus while BNL is centered on fundamental nuclear physics). In addition, national labs from other countries are represented in the BISON user base. Italy, Australia, Canada, Japan, and Finland are just a few of the International Laboratory users which share common research focus. Increasing users in the national labs outside of INL will require additional diversification of the code. One existing method which allows for this diversification is the existence of the MOOSE Multi-App feature. Figure 7.5 shows users are aware of this advanced feature. Understanding how to best leverage this feature to diversify the user base will be investigated in the future.

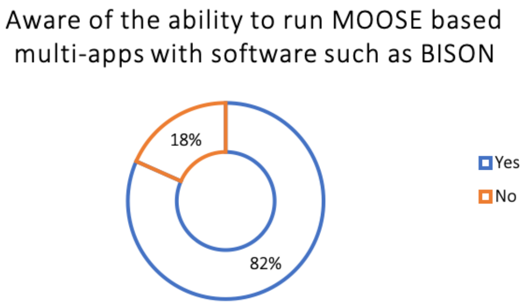
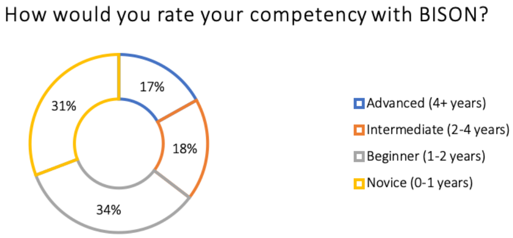


Figure 7.5: BISON user awareness of Multi-App

Related to the usage of BISON as the simulation tool, the user background with BISON varied in familiarity and skill level. Figure 7.6 shows the competency BISON users assign to themselves. Newer users, with experience under 2 years, comprise the majority of respondents, as well as those who did not respond according to the BISON licensing agreements. More experienced users with over 2 years of experience are a mixture of earlier adopters and national lab employees. Future growth will focus on newer users as this group grows.



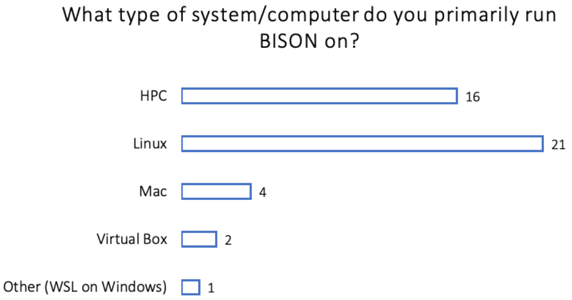


Figure 7.6: BISON user self-assigned competency levels.

Figure 7.7: Computer systems on which users run BISON.

User operating systems were collected as part of the survey as shown in Figure 7.7. The current market predicts between 77% to 87% of the OS market to be Windows based [1]. With the majority of users working on Linux/Mac systems, future growth in this area will be met with the current plan to create a Windows platform.

Figure 7.8 shows the users' familiarity and relative usage of other nuclear simulation codes related to BISON. While the fact that users primarily are working with BISON, the relative lack of usage of other codes denotes users choose BISON over other codes for a number of reasons. First and foremost is the correlation of how users became aware of BISON being primarily driven by introduction from other BISON users. This in conjunction with the fact that most users are national laboratory employees, drives the usage of software created by such employees. In addition, the relative cost of operating BISON when compared to the other software is a driving factor for economically motivated reasons.

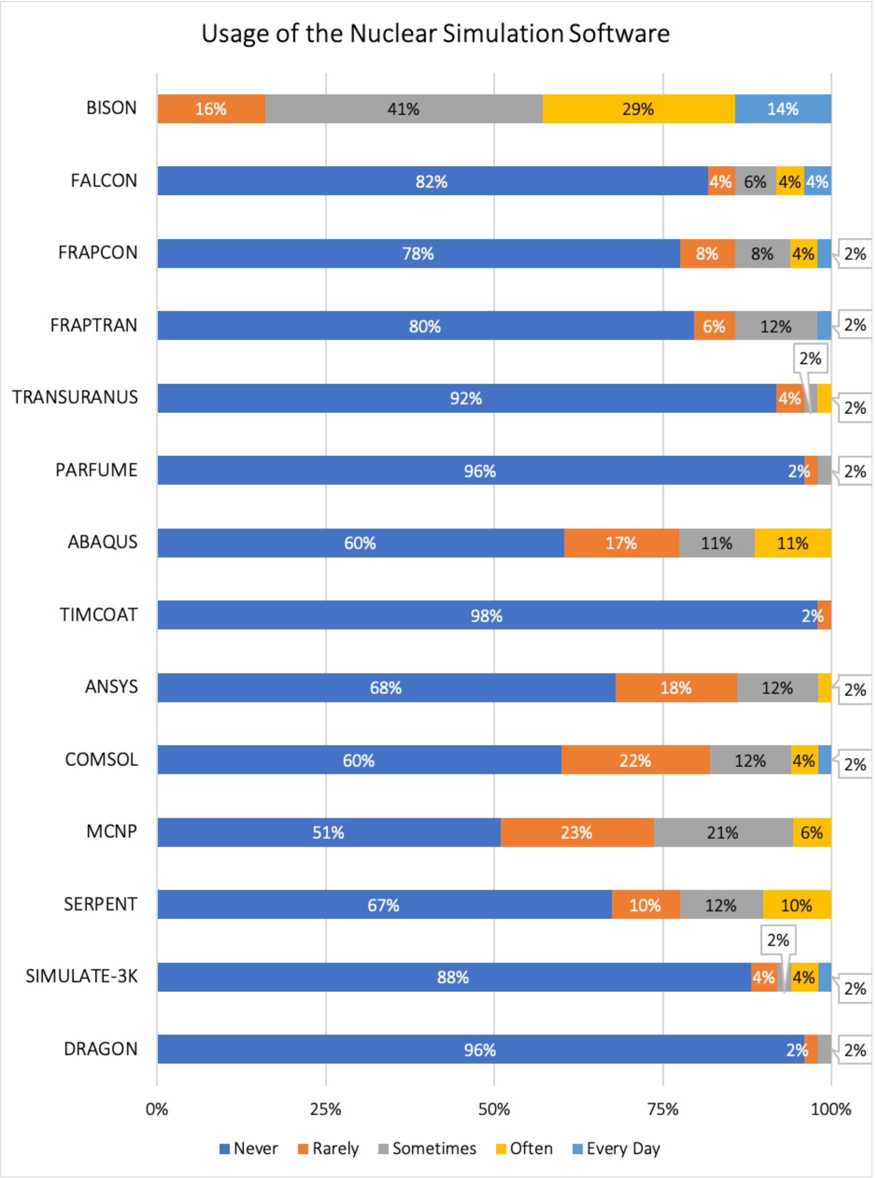


Figure 7.8: Familiarity of users with nuclear simulation codes similar to BISON.

#### 7.1.1.2 User Satisfaction

BISON users' evaluation of the ease-of-use is shown in Figure 7.9. A strong 4 out of 5 for ease of use shows BISON is moving toward a software that is easy for users to work with, but improvements can still be made as shown in Figure 7.10. Documentation remains one of the major areas of concern. There has been considerable effort to resolve this issue in this area. As discussed in the Section 7.2, as well as Section 7.3, this is an area of continued focus in both current and future work.

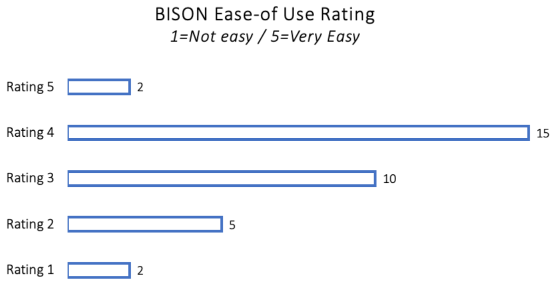


Figure 7.9: BISON user ease-of-use rating.

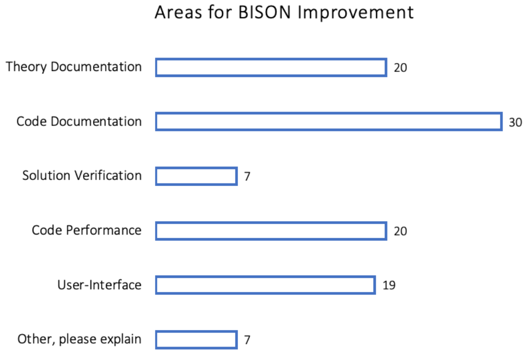


Figure 7.10: BISON user suggestions on areas of improvement.

The second most valuable area of improvements falls under code performance. For the seven user-written responses described as "Other," five of the seven responses would fall under this classification as well. This area can encompass various methods of solution convergence, physical models, as well as relative simulation speed.
The final most valuable area of improvement for BISON is user interface. This area is under development with the deployment of BISON in Windows as well as creating NuclearMaterials and drop-down completion using the Atom editor discussed within this report.

### 7.1.2 Assisting Users

BISON training is, and continues to be, an important method of helping users more efficiently utilize the software. Figure 7.11 shows the interest and value users place on these training sessions. Currently, training classes are run for beginner as well as intermediate skill groups, with advanced user skill groups discussed as a future addition.

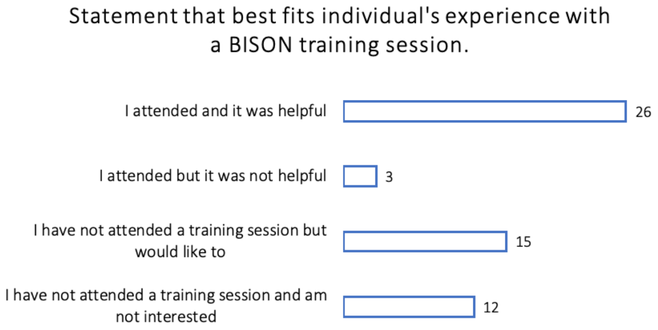


Figure 7.11: BISON user interest in training sessions.

In addition to improving BISON documentation for users, work was focused on creating new user ease-of-use for input file construction. This work built on the previously existing "Actions" used within MOOSE and BISON that users were aware of as seen in Figure 7.12. These actions serve to reduce the complexity and length of input files users create, while retaining the functionality.

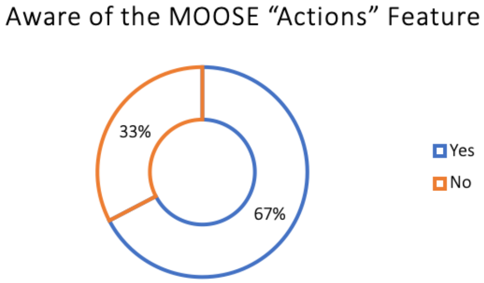
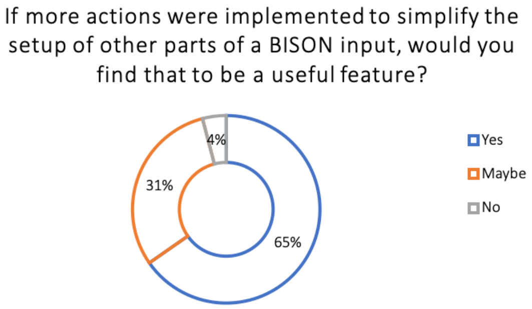


Figure 7.12: User awareness of MOOSE actions.

Users' evaluation of these "Action" tools were generally positive and considered to be useful as in Figures 7.13 and 7.14.



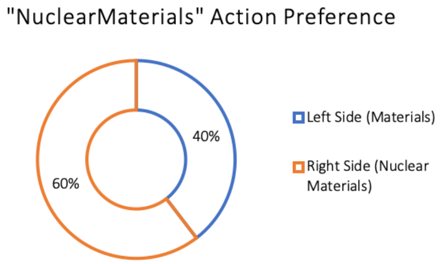


Figure 7.13: User assessment in Actions helping create input files.

Figure 7.14: User interest in NuclearMaterials input file length simplifications.

Current development of "Actions" within Moose and BISON involve the creation of tools to help quickly and efficiently construct input files based on users' needs. The typical simulations run by users, as seen in Figure 7.15, helped to influence the most recent development of `NuclearMaterials` for LWRs. Actions were broken into fuel and cladding components for LWRs as well as metallic fuel simulations. `NuclearMaterials` based on TRISO have not been currently developed, but are planned in the future.

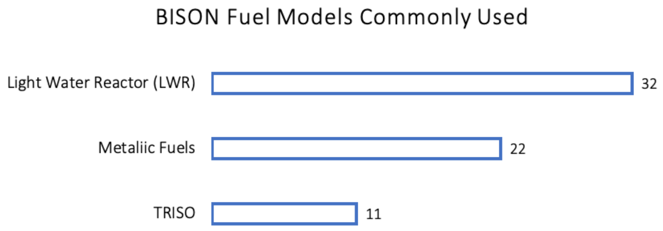


Figure 7.15: Common types of simulations run by BISON users.

In ensuring that the NuclearMaterials for LWRs best met the needs of users, the survey re-verified the most common and essential components for these types of simulations. These commonly used features are shown in Figure 7.16. The specific user responses which fall under "other" expanded on topics which additionally build upon inelastic stress and creep models. These features already exist within BISON; however, there is no implementation via Actions to simplify this for users at this time. Currently, elastic stress was developed as a first pass in creating these classes, with inelastic and creep models being added in future additions. Similarly, as was done for NuclearMaterialsLWR, user input of NuclearMaterialsMetallicFuel commonly used features was verified. Figure 7.17 shows thermal models, creep, and cladding failure as the most often used types of simulations respectively. Again, as was done with the development for NuclearMaterialsLWR, the first pass for NuclearMaterialsMetallicFuel started with thermal models with future additions planned to meet user requests.

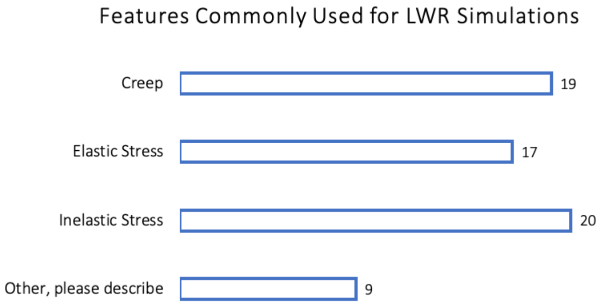


Figure 7.16: Common types of LWR simulations run by BISON users.

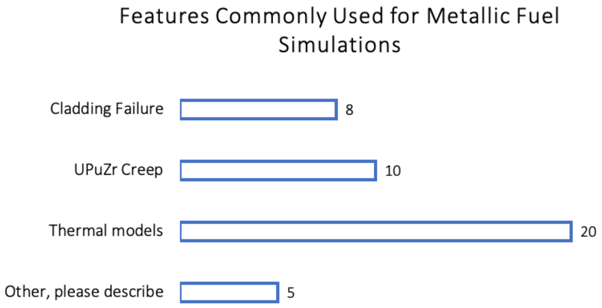


Figure 7.17: Common types of Metallic Fuel simulations run by BISON users.

### 7.1.3 Input File Creation

Most users' interactions with BISON is through input file creation. As such, considerable effort is devoted to creating a seamless experience which allows for customization with a simple and efficient method of input. Figure 7.18 shows that users primarily rework already existing input files to better suit their needs. Debate on creation of "templates" was discussed, but in allowing for greater flexibility, a dynamic method was chosen instead.

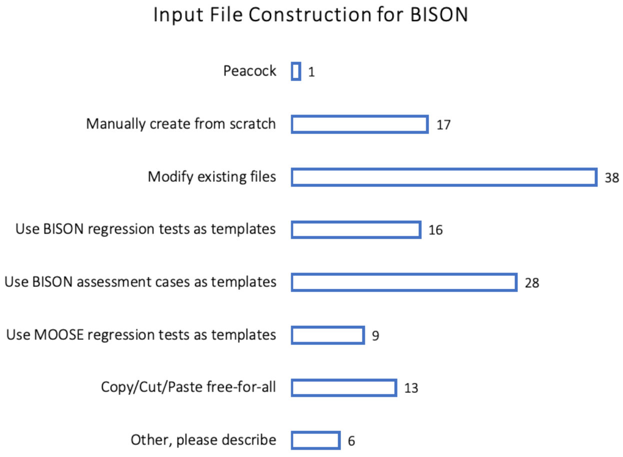


Figure 7.18: Typical BISON user input file construction methods.

In developing a dynamic input file creation technique, it was decided to make use of an editor. Taking advantage of the survey results, as seen in Figure 7.19, Atom was confirmed as the most commonly used code editor within the BISON user group. Atom had already been utilized to develop additional MOOSE-based macros with input files and as such was a logical choice for such a task. User opinions on said dynamic templates via Atom, as seen in Figure 7.20, was extremely positive.

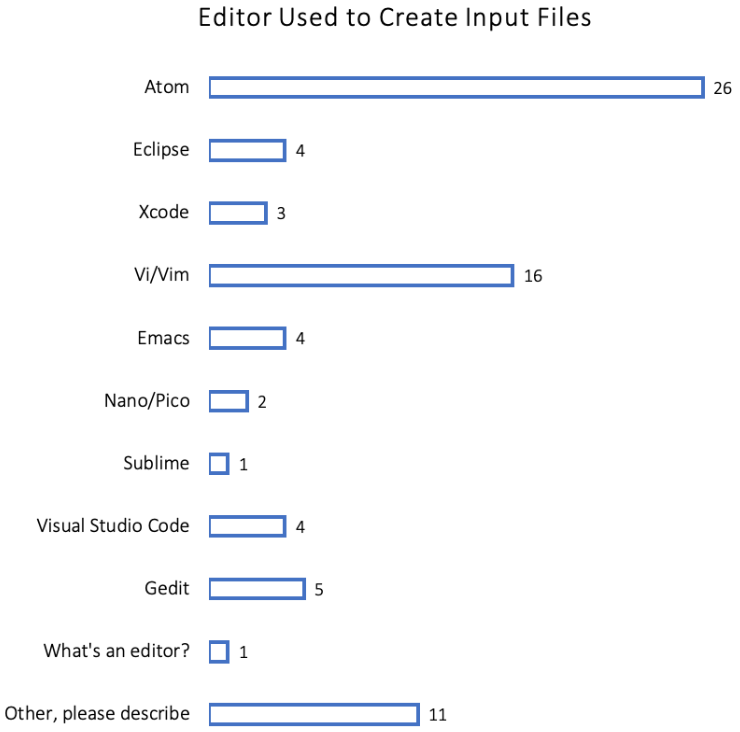


Figure 7.19: Typical BISON user editors used in input file construction.

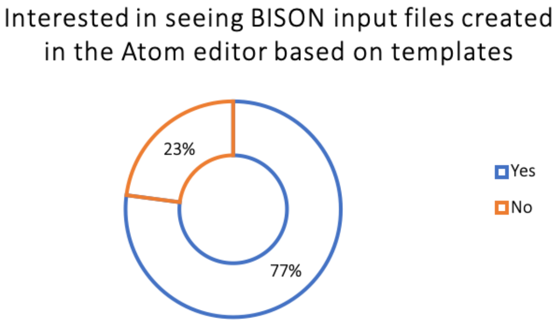


Figure 7.20: User interest in Atom editor input file template construction.

The Atom-based input file template construction method has been deployed and user satisfaction will be examined in the future.

## 7.2 Documentation Improvements

### 7.2.1 End-user Input on BISON Documentation

End-user scores on BISON documentation in Figure 7.21 reveal a mean and a mode score of B+ and A-, respectively. A large number of written complaints came from lack of documentation and the BISON team has spent a considerable amount of time updating documentation for this reason. A problem that persists is that some missing documentation comes from MOOSE, which BISON is built on. Users state that often looking through the source code helps answer their questions; however, this solution is problematic. As the user base grows, users will not be expected to be as adept in computer science. While looking at the source code may be a temporary fix, a better long term solution needs to be developed.

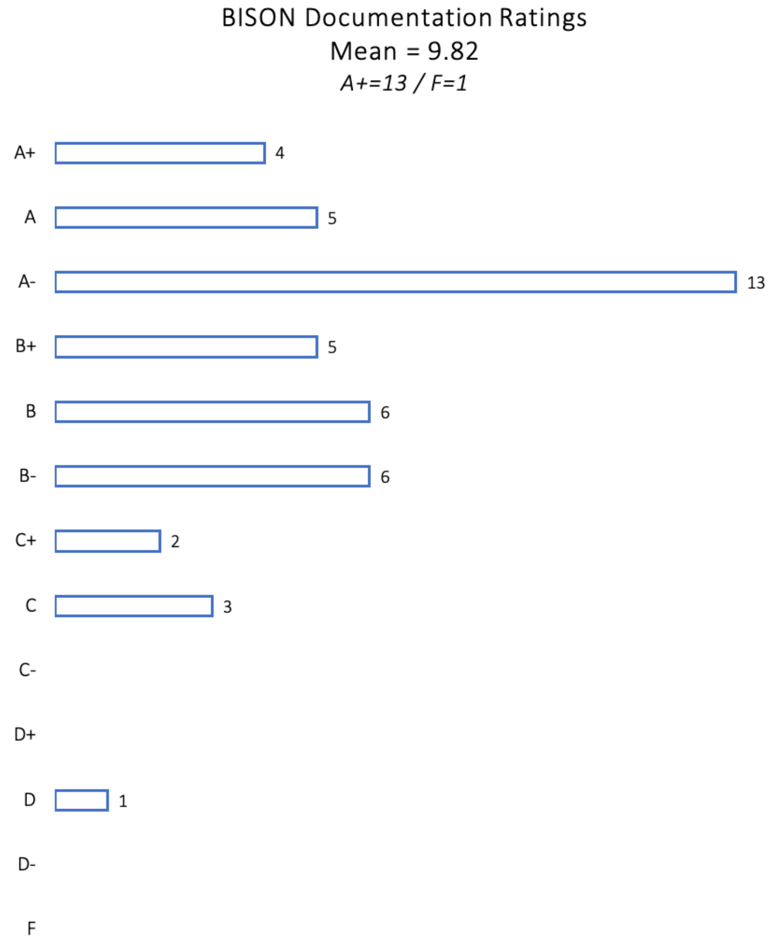


Figure 7.21: User grading of BISON documentation.

Related to the satisfaction with the documentation, Figure 7.22 shows the users' rating of the relative helpfulness of resources in developing input files. Direct interaction is obviously the most popular method of answering user questions; however, this is a non-funded area and extremely time consuming on the developer side. Developing the documentation further along with the examples will help to remedy this situation. Future plans have included a FAQ on running simulations as well as "Best Practices" web document. Discussions about creating future MOOSE and BISON user forums will also help to both reduce confusion on questions and issues.

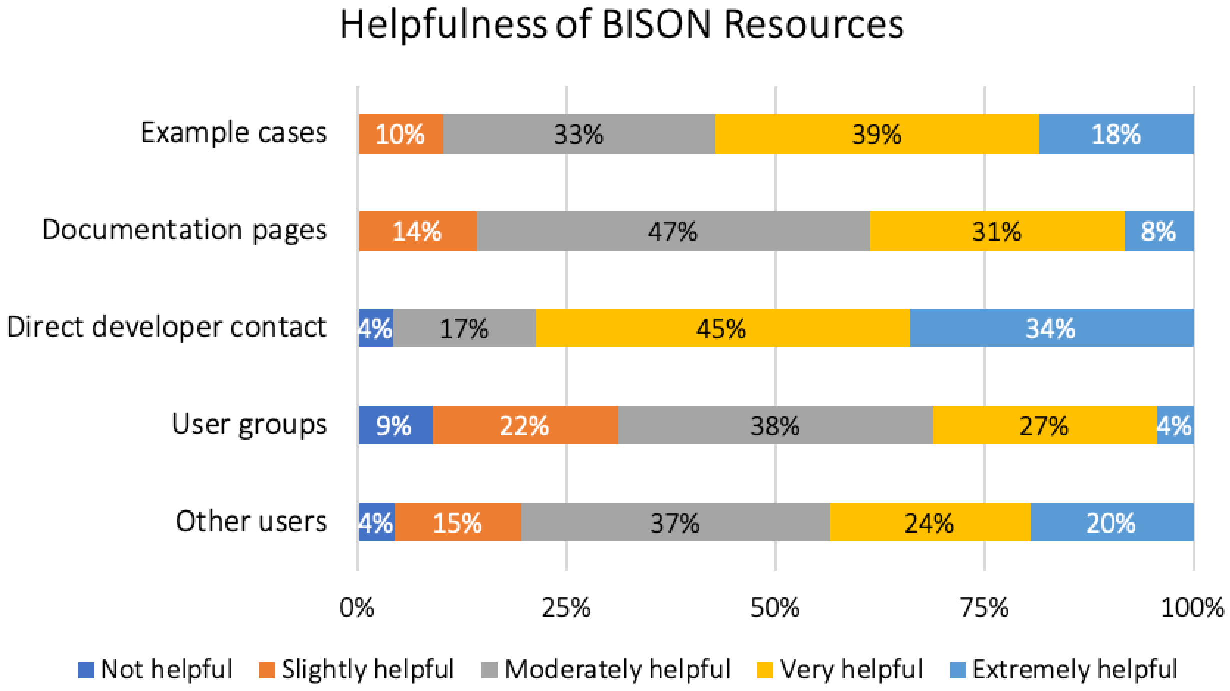


Figure 7.22: Helpfulness of BISON resources.

### 7.2.2 Full Migration to Tensor-Mechanics

Documentation was refined as part of the move away from the previous mechanics system, Solid Mechanics (SM), to the new system, referred to as Tensor Mechanics (TM). This effort has been an ongoing project and documentation outlining the migration of input files for users was developed previously and finalized this year.

As part of this work, all assessment, example, and test input files where verified to be correctly converted over to the TM system. The results were compared with the gold files and, if necessary, new gold files were created.

#### 7.2.2.1 Creation of New Classes

As part of the updates to the system, numerous new classes were created within BISON and MOOSE. Table 7.1 is not a complete list, in that more classes are continually being added.

| Newly Created Classes | |
|---|---|
| **New Class Name** | **Function** |
| NuclearMaterialThermal | Creates common thermal kernels and variables for thermal models |
| NuclearMaterialHT9 | Creates common metallic cladding material classes to reduce input file length |
| NuclearMaterialUPuZr | Creates common metallic fuel material classes to reduce input file length |
| RankTwoInvariant | Computes scalar from Rank-2 tensor |
| RankTwoCylindricalComponent | Computes scalar from Rank-2 tensor in direction of cylindrical axis |
| RankTwoDirectionalComponent | Computes scalar from Rank-2 tensor in user specified direction |
| GenericKernel | Allows inheritance from either Kernel or ADKernel |

Table 7.1: Newly Created Classes in BISON

#### 7.2.2.2 AD Classes

Numerous clean-ups were implemented on BISON assessment input files as seen in Table 7.2. A major focus was spent on further utilizing and creating Automatic Differentiation (AD) classes within input files. AD capabilities in MOOSE underwent a refactoring, which allowed new classes to be constructed quickly and efficiently.

| New AD Classes | |
|---|---|
| **Class Name** | **Method of Conversion** |
| ADUO2CreepUpdate | Original |
| ADBurnupAux | New |
| ADFissionRateAux | New |
| ADNeutronHeatSource | New |
| ADAbruptSoftening | New |
| ADComputeSmearedCrackingStress | New |
| ADExponentialSoftening | New |
| ADPowerLawSoftening | New |
| ADZryOxidation | New |
| ADThermalZry | New |
| ADZryElasticityTensor | New |
| ADZryOxideGrowthRate | New |
| ADZryOxideGrowthRateModels | New |
| ADUO2RelocationEigenstrain | Original |
| ADVolumetricSwellingEigenstrain | Original |
| ADThermalFuel | New |
| ADThermalUO2FissionGas | New |
| ADThermalUO2Meso | New |
| ADRankTwoInvariant | New |
| ADRankTwoCylindricalComponent | New |
| ADRankTwoDirectionalComponent | New |

Table 7.2: Newly created AD classes in BISON

### 7.2.3 Assessment Suite Improvements

BISON currently has a set of assessment cases that are run nightly on INL's high-performance computing (HPC) platform. Each night, after the assessments have completed running, a report is sent out to the development team indicating which cases need direct attention. This suite of cases ensure that the development of BISON remains consistent and accurate with real-world experiments and case studies.

Furthermore, since June 2019, the development team has tracked performance data associated with each assessment case. This data has aided in determining performance impact on new additions to MOOSE and BISON. Also, this data is available to serve as a long-term benchmark of BISON's performance.

In the past, each nightly run was stored simply as version-controlled repositories on HPC. While this structure was simple, it quickly became an overwhelming task for anyone interested in ad-hoc analysis of BISON's performance. This led the development team to rethink how this data was being stored, and what would make the process of fetching, analyzing, and reporting data easier.

An SQL database was created to store and organize the data in a tabular fashion. As a result, assessment case names, tests, and input-files were standardized to create a direct mapping between cases and data. Also, a unique ID was added to all assessment input-files in-order to efficiently track cases through time (i.e., file-name changes, removal of specific cases). An additonal pre-check CIVET test was added to BISON, ensuring all future additions to the assessment suite contain a unique ID.

Currently, the database contains 35 gigabytes of tabular data that tell a story of BISON's continual development over the past year. Future work includes plans to provide a simple Python API to access the data, as well as provide dynamic reporting capabilities of BISON's performance.

## 7.3 NQA-1 Compliance

BISON follows a software quality assurance (SQA) procedure aligned with the American Society of Mechanical Engineers (ASME) Nuclear Quality Assurance Level 1 (NQA-1) requirements, although BISON is intended only for Nuclear Quality Assurance Level 2 (NQA-2) applications. The use of ASME's NQA-1 standard is endorsed by the NRC, and is therefore a key requirement for fuel performance analysis software used in license applications.

### 7.3.1 NQA-1 Audit

BISON development procedures include, among other things, issue tracking, version control, peer review, regression testing, testing, and quantification of code coverage [2, 3]. These procedures are flexible enough to accommodate the modern agile software development process while tracking, through the MOOSEDocs system, the software design requirements and accompanying documentation for each code element. Two separate reviews are completed before a source-code change is accepted into BISON: an independent code review of the individual code change, and a technical lead review when a version of BISON is given an official version tag.

In February 2020 the BISON code, in conjunction with the MOOSE framework code, underwent a Nuclear Quality Assurance (NQA-1) audit, performed by software quality assessors from ASME's NQA-1 committee, and received a rating of 'Effective.' This ensured that the MOOSE framework and BISON meet Department of Energy and ASME NQA-1 requirements. MOOSE and BISON can be used in safety software applications (QL-1 and QL-2), including at ATR, which requires an NQA-1 pedigree. The assessors noted good coordination and solid expertise among staff, few product-related issues, and a high degree of technology use for development (MOOSE tools & methods).

### 7.3.2 MOOSEDOCS: In-Code Documentation System

Doxygen has and continues to be used for documentation within the C++code for MOOSE and BISON. However, this type of documentation is often only helpful in understanding the software development aspects. To help better explain the physics and theory behind the software a more robust style of documentation is needed.

"MOOSEDocs is a stand-alone Python tool within MOOSE that includes the ability to convert markdown to HTML, LaTeX, and presentations" [4]. MOOSEDocs was based on MkDocs, but has independently developed to better meet the capabilities and resiliency requirements for MOOSE and BISON. The flexibility of this system allows documentation to be integrated and required when new code is implemented.

Even with requirements, there existed a significant amount of incomplete or missing documentation. To better meet NQA-1 standards, a significant effort in resolving this missing documentation was undertaken. Multiple group efforts as well as individual efforts helped to reduce the numbers of documentation shortcomings. A partial list of the fixes to the documentation is shown in Table 7.3. During the migration from Solid Mechanics to Tensor Mechanics, certain classes were removed from BISON, and as such the missing documentation associated with said classes was deemed unnecessary. In addition, there is an ORNL milestone effort with a pending merger in BISON dealing with "thermochimica" related classes (M3MS-19OR0201043 *Verify this milestone number?*).

| Documentation Improvements | |
|---|---|
| **Name** | **Method of Improvement** |
| moose/modules/xfem/doc/content/modules/xfem/index.md | Written documentation |
| doc/content/source/actions/CoolantChannelBCAction.md | Removal of class |
| doc/content/source/actions/CoolantChannelUserObjectAction.md | Written documentation |
| doc/content/source/actions/CoolantChannelMaterialAction.md | Written documentation |
| doc/content/source/auxkernels/BurnupMetalAux.md | Written documentation |
| doc/content/source/auxkernels/CoolantAux.md | Written documentation |
| doc/content/source/auxkernels/GapValueDefaultSameValue.md | Written documentation |
| doc/content/source/auxkernels/OxygenThermochimicaAux.md | In Progress |
| doc/content/source/auxkernels/PlenumTemperatureDistance.md | Written documentation |
| doc/content/source/auxkernels/PorosityAuxUO2.md | Written documentation |
| doc/content/source/auxkernels/PowerFunctionAux.md | Removal of class |
| doc/content/source/auxkernels/Radius.md | Written documentation |
| doc/content/source/bcs/coolant/ConvectiveFluxLWRBC.md | Written documentation |
| doc/content/source/bcs/HydrogenFluxBC.md | Removal of class |
| doc/content/source/bcs/HydrogenFluxBC_simplified.md | Removal of class |
| doc/content/source/bcs/PostprocessorBulkCoolant.md | Written documentation |
| doc/content/source/bcs/REBEKADirichletBC.md | Written documentation |
| doc/content/source/bcs/StanNeumannBC.md | Removal of class |
| doc/content/source/ics/UO2PXOxygenic.md | Written documentation |
| doc/content/source/kernels/CompositeHeatConduction.md | Written documentation |
| doc/content/source/kernels/ConstituentDiffusion.md | Written documentation |
| doc/content/source/kernels/HZrHSource.md | In progress |
| doc/content/source/materials/CoolantChannelMaterial.md | Written documentation |
| doc/content/source/materials/CreepUPuZrModel.md | Removal of class |
| doc/content/source/materials/GapConductanceLWR.md | Removal of class |
| doc/content/source/materials/MaterialHZrH.md | In progress |
| doc/content/source/materials/PorosityMOX.md | Written documentation |
| doc/content/source/materials/RadioActiveDecayConstant.md | Removal of class |
| doc/content/source/material/ThermalExpansionUPuZr.md | Removal of class |
| doc/content/source/materials/ThermalNa.md | Written documentation |
| doc/content/source/materials/ThermalUO2PX.md | Written documentation |
| doc/content/source/mesh/SmearedPelletMesh.md | Written documentation |
| doc/content/source/mesh/TRISO1DMesh.md | Written documentation |
| doc/content/source/userobject/CoolantChannelUserObject.md | Written documentation |
| doc/content/source/userobject/PlenumPressureUserObject.md | Written documentation |
| doc/content/syntax/NuclearMaterials/UPuZr/index.md | Written documentation |
| doc/content/syntax/thermochimica_index.md | In progress |
| doc/content/tutorials/advanced_fuels/triso_tutorial/buffer.md | Written documentation |
| doc/content/tutorials/advanced_fuels/triso_tutorial/kernel.md | Written documentation |
| doc/content/tutorials/advanced_fuels/triso_tutorial/matrix.md | Written documentation |
| doc/content/tutorials/advanced_fuels/triso_tutorial/pyc_layer.md | Written documentation |
| doc/content/tutorials/advanced_fuels/triso_tutorial/sic_layer.md | Written documentation |
| doc/content/tutorials/advanced_fuels/triso_tutorial/triso_geometry.md | Written documentation |

Table 7.3: Documentation improvements in BISON

# Bibliography

[1] Statista. Market share statistics. `https://www.statista.com/statistics/218089/global-market-share-of-windows-7/`, April 2020.

[2] Software quality assurance plan for MOOSE and MOOSE-Based applications. Technical Report Document ID: PLN-4005, Rev. 7, Idaho National Laboratory, Idaho Falls, ID (United States), 2020.

[3] J.D. Hales, S.R. Novascone, B.W. Spencer, R.L. Williamson, G. Pastore, and D.M. Perez. Verification of the BISON fuel performance code. *Annals of Nuclear Energy*, 71:81–90, September 2014.

[4] A.E. Slaughter, C.J. Permann, J.M. Miller, B.K. Alger, and S.R Novascone. Continuous integration, in-cod documentation and automation for nuclear quality assurance conformance. 2020.

# 8 Future work

The following sections highlight proposed milestones and activities that logically follow prior work. The actual work, of course, depends on the FY-21 budget.

## 8.1 Anisotropic material model improvements

The focus on new fuel and fuel-related materials such as metallic fuels, uranium nitride, and graphite are drivers for several new capabilities at the MOOSE and BISON levels. Anisotropic materials require an upgrade of our inelastic mechanics capabilities, which currently heavily rely on radial return mapping and the assumption of isotropic stiffness. We plan to add additional code paths to support fully anisotropic materials, without degrading the performance for modeling isotropic materials. Contact remains a challenging component of our modeling toolkit. While important improvements have been made through mortar and automatic differentiation we'd like to explore further approaches, such as Nitsche's method, which yields to better conditioned systems.

## 8.2 Reduced order modeling tools

Develop required foundational tools for performing reduced order modeling (ROMs) which will accelerate models informed form lower length-scale simulations. This entails developing ROMs for BISON simulations based on machine-learning techniques such as recurrent neural networks. Example applications would be surrogate models for individual fuel rods to be used in full-core simulations including temperature and dimensional changes under basic normal operation as well as failure modes such as clad burst under off-normal conditions, and surrogate models for thermal conductivity or metallic fuel swelling and growth trained on mesoscale simulations. Future applications of ROMs will enable the development of inverse UQ capabilities.

## 8.3 Adaptive meshing and stateful material compatibility

The vast majority of BISON models use stateful material properties, due to the use of large deformation mechanics and non-linear material properties. Currently this precludes us from taking advantage of the adaptive meshing capabilities in MOOSE. For pellet clad mechanical interaction with stress concentration hot spots, we anticipate mesh adaptivity to deliver significant performance improvements. Similar locally concentrated phenomena occur in TRISO and metallic fuel. As a stretch goal we would explore the interaction of mesh refinement and XFEM.

## 8.4 Cracking/damage model robustness

Improve the robustness and expand on the capabilities of existing cracking/damage models by generalizing the existing outer solution iteration technique used currently for XFEM to allow its use with other fracture models for managing damage evolution. This would allow for incrementally updating the damage state in a subset of the cracking elements, and can be leveraged to improve the robustness of a variety of fracture modeling techniques, including smeared cracking, peridynamics, cohesive zone models, and phase-field fracture. In addition, during that iteration, failed elements can be deleted, which would be useful for multiple applications such as LWR fuel fragmentation modeling and cladding rupture.

## 8.5 Software maintenance and support

Maintain software quality standard, provide user support and training, improve usability and documentation.

# 9 Acknowledgments

The submitted manuscript has been authored by a contractor of the U.S. Government under Contract DE-AC07-05ID14517. Accordingly, the U.S. Government retains a non-exclusive, royalty free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.