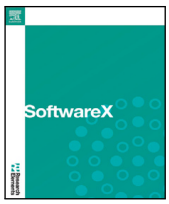
SoftwareX 26 (2024) 101754

Contents lists available at ScienceDirect

SoftwareX

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



Original software publication

MOOSE Optimization Module: Physics-constrained optimization

Zachary M. Prince [a], Lynn Munday [a,*], Dewen Yushu [a], Max Nezdyur [a,b], Murthy Guddati [c]

[a] Idaho National Laboratory, Idaho Falls, ID, 83415, United States
[b] Duke University, Durham, NC, 27708, United States
[c] North Carolina State University, Raleigh, NC, 27695, United States

ARTICLE INFO

Keywords:
Optimization
Automatic adjoint
Finite element method

ABSTRACT

The MOOSE Optimization Module integrates optimization capabilities within the MOOSE framework, enabling efficient and accurate physics-constrained optimization. This module leverages automatic differentiation to compute Jacobians and employs an automatic adjoint formulation for gradient computation, significantly simplifying the implementation of optimization algorithms. The primary goal of this software is to provide a platform where analysts and researchers can rapidly prototype and explore new optimization algorithms tailored to their complex multiphysics problems without requiring them to be computational experts. By handling the aspects of adjoint problem formulation and gradient computation, the module allows users to focus on the optimization problem itself, thereby accelerating the development of more efficient designs and solutions.

Code metadata

Current code version	Date: 2024–03–16 SHA: bc5a2fe
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00146
Legal Code License	LGPL 2.1
Code versioning system used	Continuous stable branch with git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	C++17 compiler (GCC or Clang) Memory: 16GB+ Disk: 30GB+ OS: Mac OS 10.13+, Linux (POSIX) Deps: MPI, PETSc/TAO, libMesh
If available Link to developer documentation/manual	https://mooseframework.inl.gov/modules/optimization/index.html
Support email for questions	https://github.com/idaholab/moose/discussions

1. Motivation and significance

Computational modeling has progressed significantly over the past few decades to facilitate routine prediction of the response of complex multiphysics systems. An important bottleneck however is that the simulation accuracy hinges on accurate inputs, i.e. material properties and load parameters, which cannot often be directly measured. Fortunately, there often exists a possibility of indirectly estimating these parameters from the system's response. Such a process is often referred to as inverse modeling, where the approach is to iteratively modify the parameters by minimizing the mismatch between predicted and measured observations. Such an approach, which can be formulated as a Partial Differential Equation (PDE) constrained optimization problem, can also be utilized for design optimization where the goal is to minimize the mismatch between predicted and desired responses. Inverse modeling and design optimization have been active areas of research in a wide range of disciplines, with specialized software packages developed in the context of single-physics problems [1,2], with routine multiphysics inversion receiving attention only recently [3]. To fill this gap, we have developed a PDE constrained optimization module in the multiphysics object-oriented simulation environment (MOOSE), which

* Corresponding author.
E-mail address: lynn.munday@inl.gov (Lynn Munday).

https://doi.org/10.1016/j.softx.2024.101754
Received 7 March 2024; Received in revised form 16 April 2024; Accepted 26 April 2024
Available online 3 May 2024

has an expanding user base for multiphysics simulation. Named MOOSE `optimization`, this module utilizes efficient adjoint-based gradient optimization approach coupled with the flexible constructs of MOOSE that naturally facilitate coupled multiphysics simulations.

MOOSE is a C++ based open-source framework for performing simulations on complex systems that involve multiphysics phenomena [4] that follows NQA-1 software quality standard for verification procedures [5]. Apart from the framework, MOOSE includes a variety of modules that implement specific solution strategies for partial differential equations that govern the physics of fluid dynamics [6], porous flow [7,8], structural mechanics [9], electromagnetics [10], microstructural evolution [11], among others. Additionally, these modules include analysis tools, like that of the stochastic tools module, which is used to perform uncertainty quantification, sensitivity analysis, and reduced-order modeling of multiphysics models [12]. MOOSE-based applications are created by dynamically linking to the framework any desired modules to solve specific engineering problems (i.e Bison [13] for nuclear fuel performance or Griffin [14] for neutronics). This provides the modules ready access to simulation data and solver methods, which enables efficient data transfers and manipulation of the solve. These aspects are vital for a generalized optimization platform since relevant algorithms rely on modifying input parameters, running models, gathering simulation results, and often gathering derivatives of the results with respect to parameters.

2. Software description

MOOSE `optimization` solves PDE constrained optimization problems formulated as

$$\min_{\mathbf{p}} f(\mathbf{u},\mathbf{p}); \quad \text{subject to } \mathcal{R}(\mathbf{u},\mathbf{p}) = \mathbf{0}, \tag{1}$$

where the design variable (e.g. material or load parameters) are represented by the vector $\mathbf{p}$. $f(\mathbf{u},\mathbf{p})$ is the objective function providing a scalar measure being minimized, e.g. a norm of the mismatch between observations and measurements. The constraint $\mathcal{R}(\mathbf{u},\mathbf{p}) = \mathbf{0}$, is the residual vector for the forward model, i.e. PDE system governing the multiphysics phenomena simulated by MOOSE (e.g. coupled heat and elasticity equations) whose solution field (state variable) is given by the vector $\mathbf{u}$. Eq. (1) appears simple on the outset but is extremely difficult to solve. The solution space, $\mathbf{u}$, can span millions of degrees of freedom and the parameter space, $\mathbf{p}$, can also be very large. Finally, the forward PDEs can be highly nonlinear, time-dependent and tightly coupling complex phenomena across multiple physics.

Due to the large number of parameters and the computational burden of forward model evaluations, we have developed MOOSE `optimization` for gradient-based optimization using the adjoint method. By using the adjoint method, the gradient is computed with a single adjoint solution, which has the same complexity as the linearized forward model and is independent on the number of parameters [15]. In the adjoint method, the gradient, i.e. the total derivative $\mathrm{d}f/\mathrm{d}\mathbf{p}$, is computed as,

$$\frac{\mathrm{d}f}{\mathrm{d}\mathbf{p}} = \lambda^{\top}\frac{\partial\mathcal{R}}{\partial\mathbf{p}}, \tag{2}$$

where λ is the adjoint variable solved for from the adjoint equation

$$\left(\frac{\partial\mathcal{R}}{\partial\mathbf{u}}\right)^{\top}\lambda = -\frac{\partial f}{\partial\mathbf{u}}, \tag{3}$$

where $(\partial\mathcal{R}/\partial\mathbf{u})^{\top}$ is the transpose of the Jacobian of the forward PDE and $\partial f/\partial\mathbf{u}$ is a forcing-like term. The forward problem Jacobian can be automatically computed using forward mode automatic differentiation [16]. The source term for the adjoint Eq. (3), $\partial f/\partial\mathbf{u}$, is straightforward to compute for common objective functions. The final term that must be computed in Eq. (2) is $\partial\mathcal{R}/\partial\mathbf{p}$ which is currently left to the user to implement.

2.1. Software architecture

MOOSE `optimization` relies on the `MultiApps` and `Transfers` systems within the MOOSE framework [17]. As illustrated in Fig. 1, a main application defines the optimization problem and one or more sub-applications define the physics. The main application is responsible for interfacing with the optimization solver (provided from PETSc/TAO [18]), interpreting and sending parameter and other required data, and running the sub-application(s). The sub-application solves the PDE constraining the optimization, computes the objective, and computes the optimization gradient based on an adjoint solve. Each time the optimization solver requests an objective and/or gradient calculation, the main application sends updated parameter values to the sub-application, runs the sub-application, then gathers the calculated objective and gradients. This usage of the `MultiApps` is similar to that described in Ref. [12], except there is no parallel execution of the sub-applications, since most gradient-based optimization algorithms are serial. Instead, parallelism is solely implemented within the forward (physics) solve.

MOOSE `optimization` follows the same design principles as the underlying MOOSE framework, including the requirement to be extensible and modular. As such, the module contain systems of objects, outlined in Fig. 1, that tackle the various aspects of inverse optimization.

2.1.1. Defining optimization problem

`OptimizationReporter` object sets up the optimization problem by linking the optimization app and the physics app to define the problem's parameters, gradients, and constraints. As an example of inverse optimization, the class `GeneralOptimization` gathers the objective from the physics application. For the gradient, the base class converts the gradient data from the physics application to an appropriate format for the optimization solver. The application programming interface (API) for these evaluations are public and virtual. So other objects, like the optimization solver, have access to these evaluations; and inherited objects can override them to provide a custom evaluation of objective and gradient.

The `OptimizationReporter` also instantiates global data that can be accessed and modified by other objects. The most convenient way to explain these data is to show an input snippet:

```
[OptimizationReporter]
  type = GeneralOptimization
  objective_name = objective_value
  parameter_names = 'param1 param2'
  num_values = '4 3'
  initial_condition = '0.01 0.01 0.01 0.01
                       0.2 0.2 0.2'
[]
```

The object allows the creation of multiple parameter "groups", which allows separation of different parameter types (e.g. optimizing two different material properties). A vector is instantiated for each group with the size and initial condition set from the input; a corresponding gradient vector is also created. `Transfers` are used to send the parameters to the physics application, as well as fill the objective value and gradient vectors.

The `OptimizationData` object is designed to compute the objective value in the physics application. It computes the misfit between the measurement data and the simulation data in the physics application. The object also initializes global data, which can be accessed by other objects. A clear way to demonstrate these data elements is through presenting an example of an input snippet:

```
[Reporters]
  [OptData]
    type = OptimizationData
    measurement_file = measurement_data.csv
```

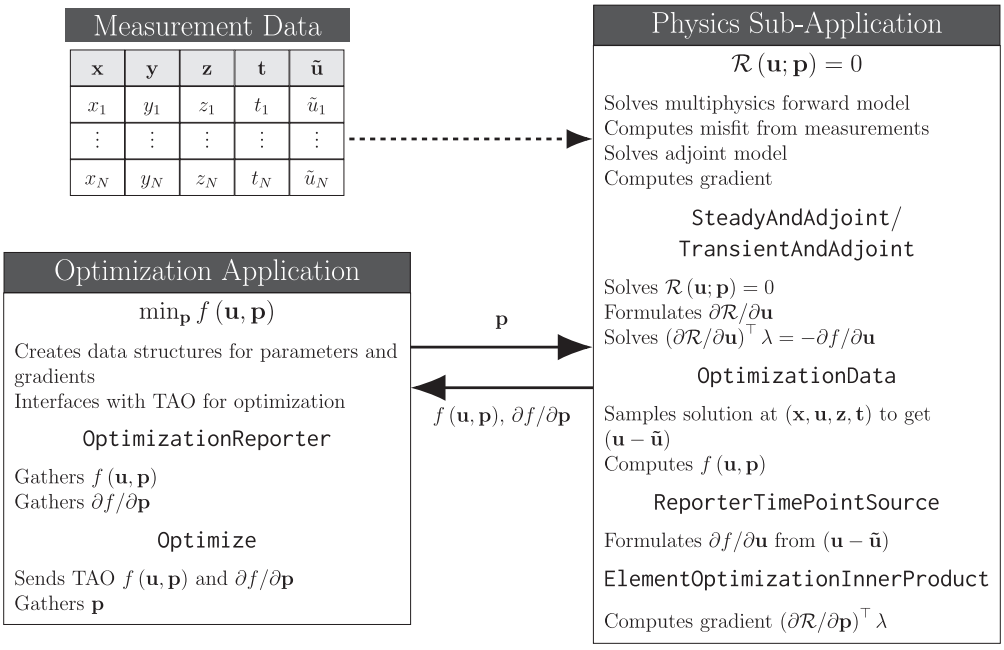


Fig. 1. `MultiApp` workflow for performing inversion with MOOSE `optimization`. This is a demonstrative workflow for parameter estimation with inverse optimization, where the objective is to minimize the difference, or misfit, between observed/measurement data ($\tilde{\mathbf{u}}$) and simulation results ($\mathbf{u}$).

```
    file_xcoord = measurement_xcoord
    file_ycoord = measurement_ycoord
    file_zcoord = measurement_zcoord
    file_time = measurement_time
    file_value = simulation_values
  []
[]
```

For the `OptimizationData` object, the measurement data is taken from a comma-separated values (CSV) file. Vectors are created specifying the spatial and temporal location of the measurement, along with a vector for the measured values of the state variable.

2.1.2. Optimization solver

The optimization solvers are implemented as MOOSE `Executioners`, which are responsible for manipulating the parameter vector and running relevant objects during the solve. `Optimize` executioner provides an interface with a variety of algorithms available through PETSc/TAO [18]. These algorithms include both gradient-free—e.g. Nelder Mead [19]—and gradient-based—e.g. quasi-Newton line search [20] and nonlinear conjugate gradient [21]—methods with options to bound parameter values. `Optimize` uses a flagging system to execute objects at the appropriate time. Objects registered with the `FORWARD` flag are executed when an objective evaluation is requested, and `ADJOINT` when a gradient evaluation is requested. PETSc/TAO requests the objective and gradient in a single call, so any object that is relevant to the gradient, like the physics sub-application, can also be executed on `FORWARD`. We also note that MOOSE `optimization` contains preliminary matrix-free implementation of Hessian-based inversion that uses Newton methods for optimization, which is not detailed here.

2.1.3. Forward and adjoint solver

MOOSE `optimization` enables an automatic adjoint evaluation within the physics sub-application using the `SteadyAndAdjoint` and `TransientAndAdjoint` executioner, which are used for steady-state and transient simulations, respectively. These executioners work by first solving the multiphysics forward problem, linearizing the forward operator around the converged solution(s), then formulating and solving the adjoint problem. The forward problem is then solved using Newton's method, which for steady-state problems takes the form:

$$\frac{\partial\mathcal{R}}{\partial\mathbf{u}}(\mathbf{u}^{i-1})\delta\mathbf{u}^i = \mathcal{R}(\mathbf{u}^{i-1}), \quad \mathbf{u}^i = \mathbf{u}^{i-1} + \delta\mathbf{u}^i, \quad i = 1,\dots,I, \tag{4}$$

where i is the iteration index and I is the number of iterations it takes to converge. The executioner then linearizes the Jacobian around the converged solution to solve the adjoint problem:

$$\left(\frac{\partial\mathcal{R}}{\partial\mathbf{u}}\right)^\top_{\mathbf{u}=\mathbf{u}^I}\lambda = -\frac{\partial f}{\partial\mathbf{u}}. \tag{5}$$

Note that the linearization of Eq. (5) must be exact, which is achieved by exact computation of the Jacobian matrix in MOOSE using automatic differentiation [16].

Formulating and solving a transient adjoint is a bit more complicated since it involves backwards timestepping [22,23]. To summarize the approach we start with the linearized form of a general time-dependent PDE:

$$\mathbf{M}(\mathbf{u}(t))\dot{\mathbf{u}}(t) + \mathbf{K}(\mathbf{u}(t))\mathbf{u}(t) = \mathbf{q}(t). \tag{6}$$

Currently, MOOSE `optimization` supports using implicit Euler time discretization, which can be applied to Eq. (6) to (formally) express the full matrix operator:

$$\begin{bmatrix} \mathbf{I} & & & & \\ \mathbf{A_{1,0}} & \mathbf{A_{1,1}} & & & \\ & \mathbf{A_{2,1}} & \mathbf{A_{2,2}} & & \\ & & \ddots & \ddots & \\ & & & \mathbf{A_{N-1,N}} & \mathbf{A_{N,N}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} = \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_N \end{bmatrix}, \tag{7}$$

where $\mathbf{u}_n$ and $\mathbf{q}_n$ are the field variable and right hand side at time t_n, respectively, and,

$$\mathbf{A_{n,n}} \equiv \frac{1}{\Delta t_n}\mathbf{M}(\mathbf{u}_n) + \mathbf{K}(\mathbf{u}_n) \quad \text{and} \quad \mathbf{A_{n,n-1}} \equiv \frac{1}{\Delta t_n}\mathbf{M}(\mathbf{u}_n). \tag{8}$$

The adjoint equation is then the transpose of this operator, with source term $(-\partial f/\partial\mathbf{u}_n)$. Note that the forward time-stepping operator is a lower triangular matrix representing forward time stepping, while the adjoint (transpose) is upper triangular matrix, which can be solved using backward time stepping:

$$\left(\frac{1}{\Delta t_n}\mathbf{M}(\mathbf{u}_n) + \mathbf{K}(\mathbf{u}_n)\right)^\top \lambda_n = -\frac{\partial f}{\partial\mathbf{u}_n} + \frac{1}{\Delta t_{n+1}}\mathbf{M}^\top(\mathbf{u}_{n+1})\lambda_{n+1}, \quad n = N,\dots,1. \tag{9}$$

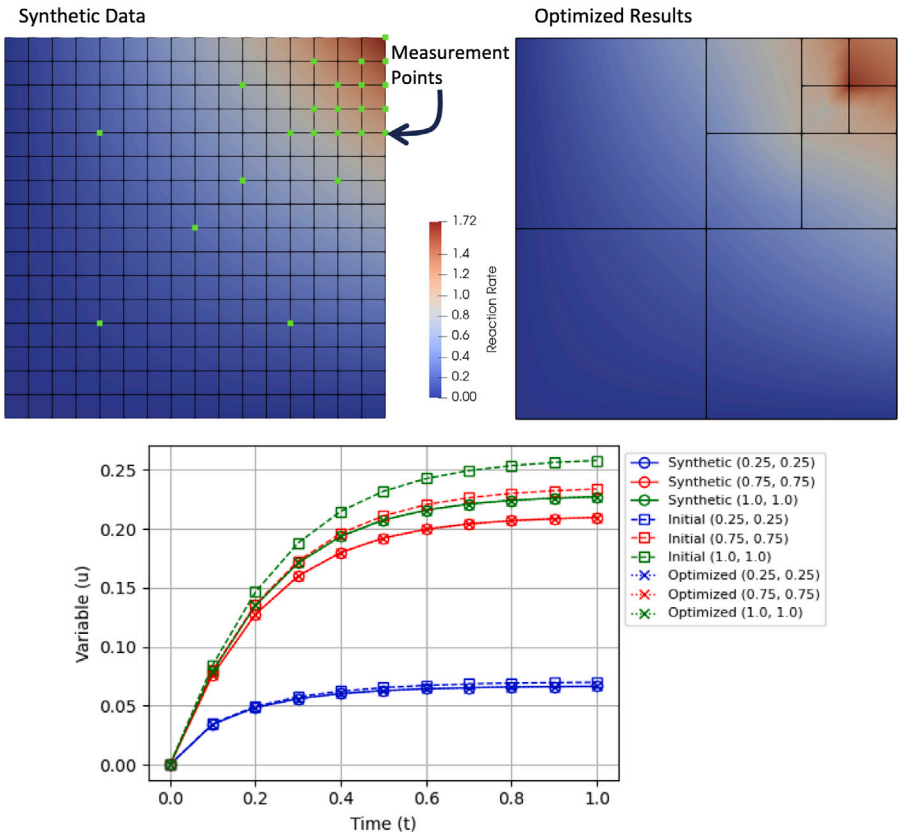


Fig. 2. Top Left: Exact reaction rate, simulation mesh, and measurement locations shown by green dots. Top Right: Optimized reaction rate and parameter mesh. Bottom: Comparing simulated transient solution with exact, initial optimization guess, and optimized reaction rate at several measurement locations (measurement and optimized data are visually identical). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The full details are presented in Algorithm 1. We note that while the implementation is currently limited to implicit Euler scheme for diffusion problems, it can easily be extended to generalize trapezoidal and other time stepping schemes, as well as to second order (dynamical) systems.

Algorithm 1 Transient and adjoint executioner algorithm

1: Set forward initial condition: $\mathbf{u}_0$
2: Cache forward solution and time: $\mathbf{u}_0$, t_0
3: **for** $n \leftarrow 1, .., N$ **do**
4: Solve forward time step: $\mathbf{u}_n \leftarrow {\mathbf{A_{n,n}}}^{-1} \left(\mathbf{q}_n + \mathbf{A_{n,n-1}}\mathbf{u}_{n-1}\right)$
5: Cache forward solution and time: $\mathbf{u}_n$, t_n
6: **end for**
7: Set previous time residual: $\mathbf{R}^{\text{old}} \leftarrow 0$
8: **for** $n \leftarrow N, .., 1$ **do**
9: Set forward solution: $\mathbf{u}_n$
10: Compute forward Jacobian: $\mathbf{A_{n,n}}$
11: Compute adjoint source: $\partial f/\partial \mathbf{u}_n$
12: Solve adjoint system: $\lambda_n \leftarrow \left({\mathbf{A_{n,n}}}^{\top}\right)^{-1} \left(\partial f/\partial \mathbf{u}_n + R^{\text{old}}\right)$
13: Evaluate time residual: $\mathbf{R}^{\text{old}} \leftarrow \frac{1}{\Delta t_n}\mathbf{M}^{\top}(\mathbf{u}_n)\lambda_n$
14: **end for**

2.1.4. Computing gradients

There are two derivatives the physics sub-application is responsible for computing. The first is the derivative of the objective function with respect to the simulation solution: $\partial f/\partial \mathbf{u}$. For inverse optimization, where the objective function is the misfit between simulation and measurement data, this derivative is computed using the `OptimizationData` object. This object allocates storage for the measurement data and samples the simulation solution at the measurement locations and times to compute the misfit. This misfit is then used by a MOOSE `DiracKernel`, called `ReporterTimePointSource`, to apply as a source term in the adjoint problem, $-\partial f/\partial \mathbf{u}$ in Eq. (5).

The second derivative is the gradient of the objective function with respect to the parameters, which, using adjoint-based gradient computation, is defined as the inner product of the adjoint solution and the derivative of the forward residual with respect to the parameters:

$$\frac{df}{dp_j} = \lambda^{\top}\frac{\partial \mathcal{R}}{\partial p_j} = \sum_{n=1}^{N}\int_{\Omega}\lambda_n(\vec{x})\left.\frac{\partial \mathcal{R}_n}{\partial p_j}\right|_{\vec{x}} d\vec{x}, \quad j = 1, \ldots, J. \tag{10}$$

This inner product is computed in MOOSE `optimization` using objects inherited from `ElementOptimizationInnerProduct`. The $\partial \mathcal{R}_n/\partial p_j$ term is currently explicitly coded for specific ways in which the parameters are applied. For instance, `ElementOptimizationDiffusionCoefFunctionInnerProduct` utilizes a MOOSE `Function` defining a diffusion coefficient ($D(\vec{x}, t; \mathbf{p})$) to compute the inner product:

$$\int_{\Omega}\lambda_n(\vec{x})\left.\frac{\partial \mathcal{R}_n}{\partial p_j}\right|_{\vec{x}} d\vec{x} \rightarrow \int_{\Omega}\vec{\nabla}\lambda_n(\vec{x})\cdot\left.\frac{\partial D}{\partial p_j}\right|_{\vec{x},t_n}\vec{\nabla}u_n(\vec{x})d\vec{x}, \tag{11}$$

where the derivative of the diffusion coefficient with respect to parameters is evaluated automatically.

2.2. Parameterization

The definition of parameters in MOOSE `optimization` is extremely flexible, as their values are declared as global data within the physics sub-application and can be accessed by any MOOSE object. However, to make applying the parameters and computing gradients simpler, MOOSE `optimization` includes various MOOSE `Functions` that use the parameter values to define a function of space and time, which can subsequently be used to parameterize unknown mate-

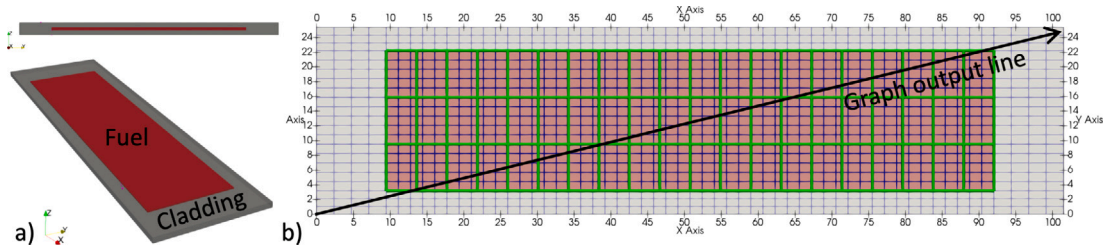


Fig. 3. Left: Fuel plate showing the thin layer of fuel sandwiched between two plates of aluminum cladding. Cladding layer is semi-transparent to show fuel extent. Right: Finite element mesh used in the thermo-mechanical plate model. Parameter mesh used to parameterize the fuel heat source is shown in green. Dimensions are in millimeters. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

rial properties or sources. The two most commonly used `Functions` for optimization are `ParsedOptimizationFunction` and `ParameterMeshFunction`. `ParsedOptimizationFunction` gives the user the ability to define a algebraic function of space and time that depends on the parameters transferred from the optimization application. `ParameterMeshFunction` takes in a pre-defined mesh to represent the parameter field and uses finite-element shape functions to perform spatial interpolation.

3. Illustrative examples

Two representative examples are presented here to illustrate the main functionalities of MOOSE `optimization` as applied to inverse optimization problems. These examples are contained in the MOOSE `optimization` test suite.

3.1. Example 1 — transient material inversion

The following illustrates the capability of MOOSE `optimization` by applying the module to a nonlinear, material-inversion problem constrained by a reaction–diffusion PDE:

$$\min_p f(u,\sigma) = \frac{1}{2}\sum_{i=1}^{N}\left(u_i - \tilde{u}_i\right)^2;$$

$$\text{subject to}\begin{cases} \dot{u} - \vec{\nabla}\cdot(1+u)\vec{\nabla}u + \sigma u = 1 & \text{in } t=[0,1], \Omega=(0,1)^2 \\ u(t=0)=0 \\ u(x=0)=u(y=0)=0 \\ \left.\frac{\partial u}{\partial x}\right|_{x=1} = \left.\frac{\partial u}{\partial y}\right|_{y=1} = 0 \\ \sigma \geq 0 \end{cases}, \tag{12}$$

where N is the number of measurement locations and u_i and $\tilde{u}_i$ are the simulated and measured state variables (e.g. concentration fields) at location i. The parameter being optimized is the spatially dependent reaction rate (σ). The PDE domain is meshed using a 16 × 16 grid of quadrilateral elements shown on the left of Fig. 2, and time is discretized using implicit Euler over ten uniform time steps. The synthetic measurement data is generated by evaluating the PDE with $\sigma = e^{xy} - 1$ and sampling the resulting solution at 22 locations—shown by the green dots on the left mesh of Fig. 2—at every time step, resulting in $N = 220$.

The reaction rate is parameterized using the mesh shown in the top right plot of Fig. 2, where parameter values set the reaction rate at the 19 nodes and are linearly interpolated among them. The initial estimates for the optimization parameters are set to $\sigma = 0$, emulating a diffusion-only system. PETSc/TAO's bounded quasi-Newton line search (`TAOBQNLS`) was used as the optimization algorithm. The fields plotted on the left and right meshes in Fig. 2 compares the exact reaction rate to the rate found through optimization. The plot in Fig. 2 compares the solution from the optimized reaction to the synthetic measurement data.

3.2. Example 2 — multiphysics force inversion

In the next example, we solve a steady state thermo-mechanical inversion problem applied to heat source generation and deformation of the nuclear plate fuel shown in Fig. 3. Plate fuel consists of nuclear fuel sandwiched between two layers of cladding material and is used in several university and national laboratory research nuclear reactors; including the Advanced Test Reactor at Idaho National Laboratory. In this problem, we parameterize a spatially varying heat source in the plate fuel based on the out-of-plane deformation measurements taken on the top surface of the cladding. This PDE constrained optimization problem is formulated as

$$\min_p f(u,q) = \frac{1}{2}\sum_{i=1}^{N}\left(u_i - \tilde{u}_i\right)^2;$$

$$\text{subject to}\begin{cases} -\nabla_s\cdot\left(\mathbf{C}:\nabla_s\mathbf{u}\right) + \nabla(\alpha\Lambda T) = \mathbf{f} & \\ -\nabla\cdot(K\nabla T) = q & \\ \mathbf{u} = 0 & \text{on } \Omega_{\mathrm{D}} \\ q_o = H\left(T - T_\infty\right) & \text{on } \Omega_{\mathrm{M}} \end{cases}. \tag{13}$$

The degrees of freedom for the mechanics problem given by the first PDE, where $\mathbf{u}$ is the (solid) displacement vector, $\mathbf{f}$ is the force vector, $\mathbf{C}$ is the modulus tensor, ∇_s is the symmetric gradient that relates the displacement to strain, Λ is the bulk modulus, which is linked to $\mathbf{C}$, T is the temperature field and α is the coefficient of thermal expansion that couples the temperature field to the mechanical response. The temperature field is governed by the equation for steady-state heat conduction given by the second PDE where K is the thermal conductivity and q is the volumetric heat flux in the fuel. The material properties used for the mechanics and heat conduction problems are linearized versions of those used in the Bison example [24,25]. A minimal number of displacement boundary conditions are contained in Ω_{D} to remove rigid body motion. Convective boundary conditions are placed on the top and bottom faces. Different values for T_∞ and H are used on each side of the plate to create a temperature gradient through the thickness to cause the plate to bow. Synthetic out-of-plane displacement data, $\tilde{u}_z$, from the top surface of the cladding is created using a spatially varying known heat source given by $q(x,y) = 80,000xy$ Watts/mm^3, producing $N = 1,298$ measurement points. This heat source and the convective boundary conditions are modified from the original Bison plate fuel assessment in order to produce a nontrivial deformation profile.

The plate fuel volume shown in Fig. 3(a) is meshed with 15,834 3D linear hexahedral elements with a mesh density on the top surface shown by the blue elements in (b). The thermo-mechanical problem solves for temperature, T, and three displacement variables at each node, $u_{x,y,z}$, resulting in 72,688 degrees of freedom. A 20 × 3 × 1 parameter mesh of constant monomial hexahedral elements are used to parameterize the heat source shown by the green elements in Fig. 3(b). Note that the FEM model of the fuel uses 50 × 15 × 3 mesh, resulting in 2.5 × 5 × 3 FEM elements per parameter mesh element where data

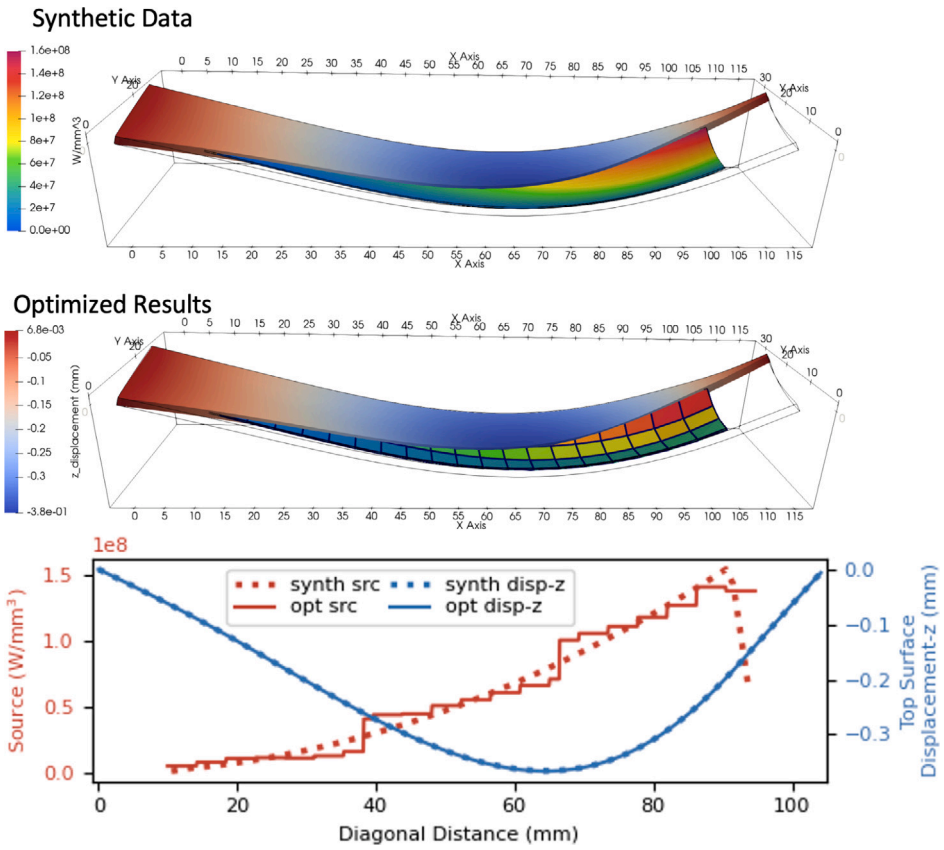


Fig. 4. Top: Synthetic and optimized fuel plate with deformation scaled by 50. Displacement of cladding in z-direction using blue to red color scale and source strength in fuel uses rainbow color scale. Optimized results show the coarse mesh used to parameterize the fuel source strength. Bottom: Comparison of synthetic (solid line) and optimized (dotted line) source (red) and displacement (blue) data along the diagonal cut in the top figure. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

is interpolated between the meshes using the shape functions of the parameter mesh.

The synthetic and optimized results are shown on the top of Fig. 4. The cladding is colored by the out-of-plane displacement, u_z, being matched to $\tilde{u}_z$ by the objective function in Eq. (13). The cladding is cut away in the synthetic and optimized results to reveal the fuel heat source being optimized. The heat source values are constant within each element used to parameterize the heat source shown in the optimized results. At the bottom of Fig. 4, the synthetic and optimized values for q are shown in red where the solid line for the optimized results is piecewise constant across each parameter mesh element. The objective function is reduced by seven orders of magnitude over 100 iterations using the gradient based `TAOLMVM` optimization solver. The maximum error between the optimized and synthetic out-of-plane displacement is 0.37% (1.4 μm).

4. Impact

MOOSE `optimization` provides PDE-constrained optimization within a multiphysics environment. It provides researchers access to state-of-the-art computational optimization algorithms without the requirement that they be computational experts. MOOSE `optimization` leverages the strengths of the MOOSE multiphysics framework, PETSc/TAO optimization library, and automatic differentiation to provide an accessible interface for researchers to prototype or characterize their systems rapidly and efficiently.

5. Conclusions

The MOOSE Optimization Module introduces new capabilities to the MOOSE framework, emphasizing physics-constrained optimization. With the integration of automatic differentiation [16] and adjoint methods, the module facilitates applying optimization algorithms to complex multiphysics problems. Recent work has expanded the module to shape and topology optimization. Additionally, the possibility of integrating reduced-order modeling with optimal control is being considered to improve computational efficiency and broaden the framework's capabilities in design and analysis. The module's evolution will reflect a commitment to addressing the complex needs of the computational modeling community, facilitating advanced optimization in multiphysics simulations.

CRediT authorship contribution statement

Zachary M. Prince: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Lynn Munday:** Writing – review & editing, Writing – original draft, Software, Project administration, Methodology, Funding acquisition, Conceptualization. **Dewen Yushu:** Writing – review & editing, Software, Methodology. **Max Nezdyur:** Writing – review & editing, Software, Methodology. **Murthy Guddati:** Writing – review & editing, Software, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Murthy Guddati reports financial support was provided by Idaho National Laboratory. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research work was supported through Idaho National Laboratory's (INL), United States Laboratory Directed Research & Development (LDRD) Program under the Department of Energy Idaho Operations Office Contract DE-AC07-05ID14517. This manuscript has been authored by Batelle Energy Alliance, LLC, under Contract No. DE-AC07-05ID14517 with the U.S. Department of Energy. This research made use of INL's High Performance Computing systems located at the Collaborative Computing Center and supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517. The United States Government retains, and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] Walsh TF, Aquino W, Ross M. Source identification in acoustics and structural mechanics using Sierra/SD. Tech. Rep. SAND2013-2689 463466, Sandia National Laboratory; 2013, http://dx.doi.org/10.2172/1095940.

[2] Wagman EB, Kurzawski A, Bunting G, Walsh TF, Aquino W, Brunini V. Transient and steady-state inverse problems in Sierra/Aria. Tech. Rep. SAND-2019-15379 681971, Sandia National Laboratory; 2019, http://dx.doi.org/10.2172/1592851.

[3] Logg A, Wells GN. DOLFIN: Automated finite element computing. ACM Trans Math Softw 2010;37(2):1–28. http://dx.doi.org/10.1145/1731022.

[4] Giudicelli G, Lindsay A, Harbour L, Icenhour C, Li M, Hansel JE, et al. 3.0-MOOSE: Enabling massively parallel multiphysics simulations. SoftwareX 2024;26:101690. http://dx.doi.org/10.1016/j.softx.2024.101690.

[5] Slaughter AE, Permann CJ, Miller JM, Alger BK, Novascone SR. Continuous integration, in-code documentation, and automation for nuclear quality assurance conformance. Nucl Technol 2021;207(7):923–30. http://dx.doi.org/10.1080/00295450.2020.1826804.

[6] Lindsay A, Giudicelli G, German P, Peterson J, Wang Y, Freile R, et al. MOOSE Navier–Stokes module. SoftwareX 2023;23:101503. http://dx.doi.org/10.1016/j.softx.2023.101503.

[7] Wilkins A, Green CP, Ennis-King J. PorousFlow: a multiphysics simulation code for coupled problems in porous media. J Open Source Softw 2020;5(55):2176. http://dx.doi.org/10.21105/joss.02176.

[8] Wilkins A, Green CP, Ennis-King J. An open-source multiphysics simulation code for coupled problems in porous media. Comput Geosci 2021;154:104820. http://dx.doi.org/10.1016/j.cageo.2021.104820.

[9] Spencer BW, Hoffman WM, Biswas S, Jiang W, Giorla A, Backman MA. Grizzly and Blackbear: Structural component aging simulation codes. Nucl Technol 2021;207(7):981–1003. http://dx.doi.org/10.1080/00295450.2020.1868278.

[10] Icenhour CT, Lindsay AD, Permann CJ, Martineau RC, Green DL, Shannon SC. The MOOSE electromagnetics module. SoftwareX 2024;25:101621. http://dx.doi.org/10.1016/j.softx.2023.101621.

[11] Schwen D, Aagesen L, Peterson J, Tonks M. Rapid multiphase-field model development using a modular free energy based approach with automatic differentiation in MOOSE/MARMOT. Comput Mater Sci 2017;132:36–45. http://dx.doi.org/10.1016/j.commatsci.2017.02.017.

[12] Slaughter AE, Prince ZM, German P, Halvic I, Jiang W, Spencer BW, et al. MOOSE Stochastic Tools: A module for performing parallel, memory-efficient in situ stochastic simulations. SoftwareX 2023;22:101345.

[13] Williamson RL, Hales JD, Novascone SR, Pastore G, Gamble KA, Spencer BW, et al. BISON: A flexible code for advanced simulation of the performance of multiple nuclear fuel forms. Nucl Technol 2021;207(7):954–80. http://dx.doi.org/10.1080/00295450.2020.1836940.

[14] Lee CH, Jung Y, Park H, Shemon E, Ortensi J, Wang Y, et al. Griffin software development plan. Tech. Rep. INL/EXT-21-63185, ANL/NSE-21/23, Idaho National Laboratory, Argonne National Laboratory; 2021, http://dx.doi.org/10.2172/1845956.

[15] Plessix R-E. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. Geophys J Int 2006;167(2):495–503. http://dx.doi.org/10.1111/j.1365-246X.2006.02978.x.

[16] Lindsay A, Stogner R, Gaston D, Schwen D, Matthews C, Jiang W, et al. Automatic differentiation in MetaPhysicL and its applications in MOOSE. Nucl Technol 2021;207(7):905–22. http://dx.doi.org/10.1080/00295450.2020.1838877.

[17] Gaston DR, Permann CJ, Peterson JW, Slaughter AE, Andrš D, Wang Y, et al. Physics-based multiscale coupling for full core nuclear reactor simulation. Ann Nucl Energy 2015;84:45–54. http://dx.doi.org/10.1016/j.anucene.2014.09.060.

[18] Balay S, Abhyankar S, Adams MF, Benson S, Brown J, Brune P, et al. PETSc/-TAO users manual. Tech. Rep. ANL-21/39 - Revision 3.19, Argonne National Laboratory; 2023, http://dx.doi.org/10.2172/1968587.

[19] Nelder JA, Mead R. A simplex method for function minimization. Comput J 1965;7(4):308–13. http://dx.doi.org/10.1093/comjnl/7.4.308.

[20] Byrd RH, Nocedal J. A tool for the analysis of Quasi-Newton methods with application to unconstrained minimization. SIAM J Numer Anal 1989;26(3):727–39. http://dx.doi.org/10.1137/0726042.

[21] Andrei N. Nonlinear conjugate gradient methods for unconstrained optimization. Berlin: Springer Cham; 2020, http://dx.doi.org/10.1007/978-3-030-42950-8.

[22] Saglietti C, Schlatter P, Monokrousos A, Henningson DS. Adjoint optimization of natural convection problems: differentially heated cavity. Theor Comput Fluid Dyn 2017;31(5):537–53. http://dx.doi.org/10.1007/s00162-016-0398-5.

[23] Marin O, Constantinescu E, Smith B. A scalable matrix-free spectral element approach for unsteady PDE constrained optimization using PETSc/TAO. J Comput Sci 2020;47:101207. http://dx.doi.org/10.1016/j.jocs.2020.101207.

[24] Hales J. BISON plate fuel tutorial. 2022, URL https://mooseframework.inl.gov/bison/tutorials/advanced_fuels/plate_tutorial.html.

[25] Rabin B, Meyer M, Cole J, Glagolenko I, Jones W, Jue J-F, et al. Preliminary report on U-Mo monolithic fuel for research reactors. Tech. Rep. INL EXT-17-40975, Idaho National Laboratory; 2020.