This is the preprint version of the following article:
Jiayao Yan, Ru Xiang, David Kamensky, Michael T. Tolley, and John T. Hwang. Topology optimization with automated derivative computation for multidisciplinary design problems. Structural and Multidisciplinary Optimization, 2022.

Published article: https://doi.org/10.1007/s00158-022-03168-2
Preprint pdf: https://github.com/LSDOlab/lsdo_bib/blob/main/pdf/yan2022topology.pdf
Bibtex: https://github.com/LSDOlab/lsdo_bib/blob/main/bib/yan2022topology.bib

Topology Optimization with Automated Derivative Computation for Multidisciplinary Design Problems

Jiayao Yan*, Ru Xiang†, David Kamensky‡, Michael T. Tolley§, John T. Hwang¶

University of California San Diego, La Jolla, CA, 92093

Abstract

Topology optimization has drawn increasing attention as a method to aid the design of engineering systems. Gradient-based optimization is typically used to solve these problems because of its efficiency in dealing with a large number of design variables. However, there is a large amount of implementation effort required for both the forward model formulation and the derivative computation of the model, especially for multiphysics problems. This paper addresses these challenges by presenting a general-purpose topology optimization platform called ATOmiCS, built in a modular framework to facilitate the formulation of complex, multiphysics problems with fully automated derivative computation. ATOmiCS automates the derivative computation by coupling FEniCS—a multiphysics partial-differential-equation solver with accessible symbolic partial derivatives—with OpenMDAO, a modular framework for multidisciplinary design optimization that can automatically solve the adjoint equation for the total derivatives. ATOmiCS is implemented as an open-source toolbox with online documentation and has been used in a graduate-level class. The features of ATOmiCS are demonstrated using three case studies: compliance minimization of cantilever beams with linear and nonlinear elasticity models, compliance minimization of a battery pack with the thermoelastic equation and an unstructured mesh, and optimization of liquid crystal elastomer using a modified elastic equation for shape matching. The results demonstrate the characteristics that, as a whole, make ATOmiCS a unique topology optimization toolbox: modularity and flexibility with respect to operations such as filtering and penalization; ease of implementation of governing equations, type of elements, and solvers for systems of equations; and fully automated derivative computation for gradient-based optimization.

*PhD Student, Department of Mechanical and Aerospace Engineering
†PhD Student, Department of Mechanical and Aerospace Engineering
‡Assistant Professor, Department of Mechanical and Aerospace Engineering
§Associate Professor, Department of Mechanical and Aerospace Engineering
¶Assistant Professor, Department of Mechanical and Aerospace Engineering

1 Introduction

Topology optimization is a numerical method that computes optimal material or structural layouts that maximize the performance of engineering systems [1]. Emerging in the structural engineering field, topology optimization was first applied to structural design problems, considering only solid mechanics [2, 3]. In recent years, the application domains for topology optimization have widened, leading to more complex models across more disciplines, such as problems involving thermoelasticity [4–9], fluids [10–13], and electromagnetism [14, 15]. However, these implementations are tailored to one specific type of multiphysics problem due to the complexity of both the problem formulation and the derivative evaluation, as gradient-based optimization is the main solution approach. Thus, there is a need to facilitate the widespread use of topology optimization for multiphysics problems by developing a general-purpose toolbox with interchangeable governing equations and with minimum effort required for derivative computation.

On the optimization side, the implementation of efficient derivative computation for complex models is a bottleneck. Recent advances in derivative computation for large-scale optimization have reduced the implementation cost of gradient-based optimization by automating different parts of the derivative computation process. Laurain [16] has proposed a level-set topology optimization implementation that automates the computation of partial derivatives using FEniCS [17], a library for solving multiphysics partial differential equations (PDEs). Chung et al. [18] have implemented topology optimization in OpenMDAO [19], a multidisciplinary design optimization (MDO) framework that automatically assembles the total derivatives using the adjoint method. However, there has not been an educational tool that can fully automate the entire derivative computation process, which includes the computation of both partial and total derivatives. Additionally, for a topology optimization problem, the governing partial differential equation (PDE), solvers (one for solving the states and the other for solving the derivatives), penalization, constraints, and the objective are often problem-specific. Thus, for a general-purpose toolbox capable of solving various topology optimization problems, those components need to be wrapped by a consistent interface and made interchangeable. The above challenges require the framework to be implemented in a modular form with fully automated derivative computation.

Here, we present a topology optimization toolbox called ATOmiCS (**A**utomated **T**opology **O**ptimization for **m**ultidisciplinary problems using FEn**iCS**). ATOmiCS leverages the capabilities of automatic code generation from FEniCS to simplify the solution of finite element problems and the computation of partial derivatives. In ATOmiCS, the different components of the model are integrated within the OpenMDAO modeling framework, which is capable of assembling total derivatives given partial derivatives of the components of the model. Thus, ATOmiCS is able to automate both steps (computing partial derivatives and total derivatives) of the derivative computation. In addition, ATOmiCS further explores the flexibility of a modular framework to provide users with multiple options for the governing equations, solvers, penalization methods, the objective, and constraints.

The main contributions of this paper are:

1. The first topology optimization method that fully automates the derivative computation for gradient-based topology optimization by combining two ideas: the symbolic specification of variational problems and a generalized adjoint method (Eq. (7));

2. Implementation in a high-quality software library that leverages version control on GitHub[1], online tutorials[2], and automatically generated documentation that dynamically embeds tested tutorial scripts.

The remainder of this paper is organized as follows. Section 2 reviews existing topology optimization implementations. Section 3 describes the density-based topology optimization methods used by ATOmiCS. Section 4 summarizes the use of FEniCS and OpenMDAO for derivative computation, while Sec. 5 illustrates how this approach is put into practice by ATOmiCS. Section 6 then applies ATOmiCS to several case studies involving different physics to demonstrate its flexibility. Lastly, Sec. 7 draws conclusions and outlines future research directions.

2 Literature survey

This section reviews the current landscape of educational topology optimization implementations, emphasizing developments from the past five years (summarized in Table 1). Many educational topology optimization contributions have emerged recently. These educational papers have presented solutions for not only structural, but also multiphysics topology optimization problems.

The most common approaches for topology optimization are the level-set method (LSM) [16, 18, 20, 21] and density-based methods [18, 22–25]. Two of the most commonly used density-based methods are the Solid Isotropic Material with Penalization (SIMP) method and the Rational Approximation of Material Properties (RAMP) method [2]. Researchers have implemented educational topology optimization tools in multiple environments such as MATLAB, C++, and Python. Some work has combined topology optimization algorithms with commercial simulation software, such as ANSYS [26].

Topology optimization tools for multiphysics problems: As mentioned in Sec. 1, prior work in topology optimization has considered multiphysics problems that span a number of domains, including thermoelasticity, fluids, and electromagnetism. Recent work contributes to educational topology optimization implementations in domains beyond structural design; for instance, Homayouni-Amlashi et al. [27] presented two separate educational topology optimization implementations for the actuation and energy harvesting of piezoelectric plates. Most existing multiphysics implementations are problem-specific; they are developed in a way that requires significant re-implementation by an expert developer when considering a new type of physics. Therefore, there is a need for an extensible educational framework for topology optimization across a variety of disciplines. In order to handle a variety of (potentially coupled) problems, the governing equations, along with solvers, filters, penalization schemes, constraints, and objective functions, need to be easily interchangeable.

Recent advances in automated derivative computation for topology optimization problems: To reduce the implementation cost, researchers have proposed various ways of automating derivative computation for topology optimization problems. To automate partial derivatives, FreeFEM++ [22] has used a scripting language based on C++ to generate the partial derivatives. Laurain [16] has presented an implementation to obtain the symbolic representation of partial derivatives from FEniCS Unified Form Language (UFL). However, using this approach, one still needs to implement the adjoint method to assemble total derivatives. Chung et al. [18] have proposed an implementation in OpenMDAO to automatically compute the adjoint total derivatives by solving the unified

[1] `http://github.com/LSDOlab/atomics`
[2] `http://lsdolab.github.io/atomics`

derivatives equation. To implement a general toolbox capable of solving a variety of topology optimization problems, the computations of both the partial and total derivatives need to be automated, as automating only one of them would still result in a significant amount of ad hoc user effort.

Table 1: Recent educational topology optimization papers

Year	Author	Physics problem	Approach	Deriv.	Environment
2021	Zhu et al. [22]	elasticity (NL)	SIMP	PA	C++ (FreeFEM)
2021	Homayouni-Amlashi et al. [27]	piezoelectricity (L)	PEMAP-P	M	MATLAB
2020	Lin et al. [26]	elasticity (L)	DER-BESO	*	ANSYS APDL
2019	Liang et al. [28]	elasticity (L)	SAP	*	MATLAB
2019	Chung et al. [18]	elasticity (L)	LSM; SIMP	TA	Python; C++
2019	Chen et al. [23]	elasticity (NL)	SIMP	M	MATLAB; ANSYS
2018	Loyola et al. [29]	elasticity (L)	SERA	*	MATLAB
2018	Laurain [16]	elasticity (L)	LSM	PA	FEniCS
2018	Wei et al. [20]	elasticity (L)	LSM	M	MATLAB
2016	Pereira et al. [13]	fluid (L)	density method	M	MATLAB
2015	Aage et al. [24]	elasticity (L)	SIMP	M	C++
2015	Nobel-Jørgensen et al. [25]	elasticity (L)	density method	M	C# (GUI); C++

L: linear; NL: nonlinear; PA: partial derivatives automated; TA: total derivatives automated; M: manual.
*: the derivative computation method for the work is not reported.

3 Review of density-based topology optimization

3.1 Topology optimization problem overview

The general form of density-based topology optimization is

$$\begin{aligned} \textbf{minimize} \quad & f \\ \textbf{with respect to} \quad & \boldsymbol{\rho}, \\ \textbf{subject to} \quad & \boldsymbol{R}(\boldsymbol{u}, \tilde{\boldsymbol{\rho}}) = 0 \\ & \int_\Omega \rho \, d\Omega = \text{constant} \\ & \rho_{\min} < \boldsymbol{\rho} \leq 1, \end{aligned} \tag{1}$$

where f is the objective function; $\boldsymbol{\rho}$ is the vector of density degrees of freedom; ρ is a density function on domain Ω, parameterized by the degrees of freedom in $\boldsymbol{\rho}$ (the density vector); $\boldsymbol{R}$ is the assembled residual vector for the weak form of the PDEs (e.g., Eq. (8)), which depends on state variables $\boldsymbol{u}$ and the filtered density; and $\rho_{min} > 0$ is a small constant to prevent singularities in the Jacobian of $\boldsymbol{R}$. The residual vector corresponds to the discretization of a PDE system, whose coefficients are modulated by the filtered density $\tilde{\boldsymbol{\rho}}$. The map from $\boldsymbol{\rho}$ to $\tilde{\boldsymbol{\rho}}$ is introduced to suppress numerical instabilities and checkerboard patterns, with various options discussed in Sec. 3.3. The integral constraint on ρ fixes the total volume of material distributed within the domain Ω. The mapping from filtered density to PDE coefficients of the effective Young's modulus also includes a penalization to discourage intermediate values, as discussed further in Sec. 3.2. For gradient-based topology optimization, the derivatives of the outputs (the objective and constraints) with respect to the inputs (design variables) of the optimization problem need to be computed.

3.2 Penalization methods

We implement two commonly used topology optimization methods, SIMP and RAMP, as options. SIMP is the most commonly used penalization method. SIMP assumes a mapping between an element-wise density property on the mesh, $\rho \in [0, 1]$, and material properties. For example, the effective Young's modulus, E, of an elastic structure would be given by

$$E = E_0 + \rho^{p_0}(E_{\max} - E_0) \ , \tag{2}$$

where $E_{\max}$ is the value at $\rho = 1$, E_0 is a minimum value to avoid singularities, and $p_0 \in (1, \infty)$ controls variation. We use a default value of $p_0 = 3$ [30]. The default value of E_0 is selected to be $E_0 = \epsilon_0 E_{\max}$, where ϵ_0 is taken to be 1×10^{-5}.

We also implement the RAMP method, which is given by

$$E = \frac{\rho E_{\max}}{1 + p_1(1 - \rho)} \ , \tag{3}$$

where p_1 (with 8 being a reasonable default value [31]) controls the variation. With the RAMP method, we use a nonzero lower bound $0 < \rho_{\min} \ll 1$ for the densities to prevent singularities. Based on Eq. (2), the SIMP method suffers from numerical difficulties as it has zero sensitivity when the densities are zero, while the sensitivities are always positive for the RAMP method (Eq. (3)). Thus, the RAMP method shows better performance under design-dependent loads, including thermal loads [32–34].

3.3 Density filters

For the filtering with unstructured meshes, we implement two types of density filters: a distance-based direct filter [35] and a variational filter.

The distance-based direct filter assumes the meshes to be quasi-uniform and computes the filtered density $\tilde{\rho}$ using a weighted average of densities in neighboring elements. Specifically, the filtered density $\tilde{\rho}_i$ in the i^{th} element is

$$\tilde{\rho}_i = \sum_j w_{ij}\rho_j \ , \tag{4}$$

where ρ_j is the unfiltered density in the j^{th} element. The weight is given by

$$w_{ij} = \frac{\langle r - d(i,j)\rangle}{\sum_k \langle r - d(i,k)\rangle} \ , \tag{5}$$

where $\langle\cdot\rangle$ are Macaulay brackets, which means that $w_{ij} = 0$ if $d(i,j) > r$; r is the filter radius selected to be $2h$; h is the global average quasi-uniform mesh size; and $d(i,j)$ is the distance between the centroids of the i^{th} and j^{th} elements.

The variational filter determines the filtered density by solving the following problem: find $\tilde{\rho} \in V_{\text{dens}}$ such that $\forall v \in V_{\text{dens}}$,

$$\int_\Omega (\tilde{\rho} - \rho) v \, d\Omega + \int_{\partial K} C h_{\partial K} [\![\tilde{\rho}]\!][\![v]\!] \, d\partial K = 0 \ , \tag{6}$$

where V_{dens} is the space of element-wise constant functions for the densities, ∂K is the union of interior facets of the mesh, $[\![\cdot]\!]$ is the jump of a quantity over an interior facet, $h_{\partial K}$ is the averaged diameter of adjacent elements, and C is a dimensionless scaling factor. On a uniform, structured interval, quadrilateral, or hexahedron mesh, the interior penalty term of our variational filter is, in fact, algebraically equivalent to a centered-difference scheme for the Laplace operator. In addition, the first term in (6) corresponds to a reaction term. This exact equivalence breaks down for general unstructured meshes, but the qualitative smoothing effect remains similar to a reaction-diffusion-based filter such as the filters from [36] and [37]. With this analogy, the length scale is a dimensionless

constant (proportional to $\sqrt{C}$) times $h_{\partial K}$. Larger values of C increase the strength of the smoothing effect.

4 Methodology

This section summarizes the methodology in ATOmiCS as a toolbox with modular architecture and fully automated derivative computation. We first introduce the background and implementations in OpenMDAO for constructing the modular toolbox and the automatic assembly of total derivatives using the unified derivatives equation (UDE) (Sec. 4.1). Then, we introduce the specification of the variational forms and the computation of partial derivatives using FEniCS UFL (Sec. 4.2). Finally, in Sec. 4.3, we summarize the overall method of ATOmiCS, including modularizing the computation of states and outputs, and automating the derivative computation.

4.1 Modular architecture and unified derivatives equation in OpenMDAO

To achieve a general-purpose topology optimization toolbox dealing with a large number of design variables inherent in multiphysics problems, we need to implement our tool such that 1) its components are flexible to change in order to adjust to different problems, and 2) it efficiently calculates the derivatives with minimal effort from the user. Here, the OpenMDAO framework is used to build this tool. We provide an introduction to OpenMDAO and how it allows ATOmiCS to achieve those two features, as well as the structure of ATOmiCS in terms of its *components*.

OpenMDAO is an open-source software framework implemented in Python for large-scale gradient-based optimization [19]. It allows the construction of a complex physical model involving multiple disciplines by decomposing the model into small units. The smallest unit in OpenMDAO is called *component*. A *component* expresses the computation for part of a disciplinary analysis or the analysis of a whole discipline. There are two types of *components* in OpenMDAO—the *explicit component* and the *implicit component*. The *explicit component* is used for variables that can be explicitly computed, while the *implicit component* is used to compute variables that are implicit functions of other variables. For ATOmiCS, the density filters, computations of outputs, and other calculations such as the preprocessing and postprocessing (shown in Fig. 1), are implemented using *explicit components*. The solution of the state variables is performed using the *implicit components*. The partial derivatives of the outputs with respect to the inputs of each *component* need to be provided to OpenMDAO.

The *components* are partitioned into a higher hierarchy unit, called the *group*. In ATOmiCS, we combine the main *components* for the topology optimization into a *group* called ATOmiCS *group* (Fig. 1). In OpenMDAO, a *group* can be combined together with *components* or other *groups* to form a larger *group*. The whole calculation combined by the *groups* and *components* is called a *model*. The *model* is then connected with an optimizer (called *driver*) to form an OpenMDAO *problem*. Thus, the units of code in an OpenMDAO model are in a hierarchy of *component*, *group*, *model*, and *problem*. This modular architecture in OpenMDAO allows the *components* as well as the options within a *component* in ATOmiCS to be flexible to change. This enables ATOmiCS to solve various topology optimization problems with different governing equations, penalizations, solvers, constraints, and objectives.

Another key feature of OpenMDAO is its capability of automatically computing total derivatives

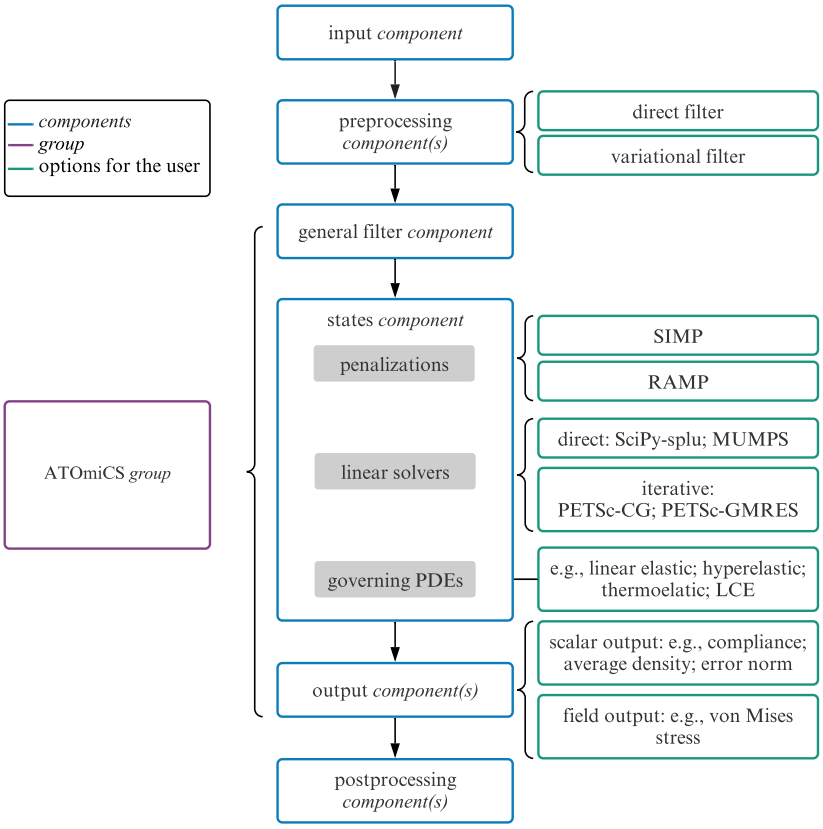


Figure 1: Structure of ATOmiCS: the purple block (left column) represents the *ATOmiCS group* composed of *components* that are automated in ATOmiCS; blue blocks (middle column) represent the *components*; grey blocks are changeable options in *states component*, and green blocks (right column) are options that user can choose from.

by solving the unified derivatives equation (UDE) [38]

$$\frac{\partial \tilde{\boldsymbol{R}}}{\partial \tilde{\boldsymbol{u}}}\frac{d\tilde{\boldsymbol{u}}}{d\tilde{\boldsymbol{r}}} = \mathcal{I} = \frac{\partial \tilde{\boldsymbol{R}}}{\partial \tilde{\boldsymbol{u}}}^T \frac{d\tilde{\boldsymbol{u}}}{d\tilde{\boldsymbol{r}}}^T , \tag{7}$$

where $\tilde{\boldsymbol{u}}$ is the concatenated vector of inputs, outputs, and states, $\tilde{\boldsymbol{r}}$ is the vector of residuals, and $\tilde{\boldsymbol{R}}$ is a residual function of $\tilde{\boldsymbol{u}}$ in the UDE that satisfies $\tilde{\boldsymbol{R}}(\tilde{\boldsymbol{u}}) = 0$. These vectors are all automatically generated by OpenMDAO, with the inputs and outputs of individual components and their connections specified by the user. In the present work, for example, $\tilde{\boldsymbol{u}}$ corresponds to a concatenated vector of the inputs, outputs, and states in all the *components* in Fig. 1.

The significance of the use of the UDE is that it automates the assembly of total derivatives. For instance, solving the right-hand side of Eq. (7) is equivalent to applying the adjoint method, which can be derived by decomposing $\tilde{\boldsymbol{u}}$ into design variables, states, and the output of interest [39]. In this case, the term $\partial\tilde{\boldsymbol{R}}/\partial\tilde{\boldsymbol{u}}^T$ is a block matrix, that when block-solved with the right column of the identify matrix in Eq. (7), yields precisely the adjoint equations.

In multiphysics topology optimization problems, there can be multiple sets of governing equations with one-way or two-way coupling, which would necessitate the solutions of two decoupled

adjoint systems in sequence or a single coupled adjoint system [40], respectively. Because ATOmiCS uses OpenMDAO as the modeling framework, the developer of the topology optimization algorithm does not have to concern themselves with identifying and implementing the right method; OpenMDAO would do so automatically by solving the UDE.

4.2 Use of FEniCS for solving variational problems and computing partial derivatives

The computation of the partial derivatives is automated using FEniCS. The FEniCS Project [17] is a toolchain for developing finite element-based PDE solvers, which can be used for solving multiphysics problems in various fields [41]. The canonical FEniCS workflow is to specify a variational formulation in the Python-based Unified Form Language (UFL) [42], which can then be automatically compiled [43] into high-performance C++ code. Computer algebra can be performed on the UFL symbolic representation of the formulation to automatically obtain partial derivatives needed for gradient-based optimization. UFL consists of sublanguages for finite elements, expressions, and forms [44]. There are operators for each sublanguage. One of the most important operators is a form operator called the *derivative* operator, which computes the symbolic form of the Gâteaux derivatives, a generalization of the directional derivative in differential calculus [45]. Within ATOmiCS, we use FEniCS to compute residual vectors of discretized PDE systems, along with matrices of their partial derivatives with respect to state and design variables. The general procedure is shown in Fig. 2, where the variational form is converted to discrete vectors or matrices using the *assemble* operator, and the symbolic derivatives are obtained using the *derivative* operator. The same calculation is applied to the computation of the outputs as well.

The main advantage of using an automated system like FEniCS to generate code for residuals and outputs (and their derivatives) is that the physics of the system being optimized can be easily changed through the high-level symbolic abstractions provided by UFL. This versatility is achieved while maintaining high performance. Because of the generality of FEniCS, changes can include adding and removing arbitrarily many coupled fields.

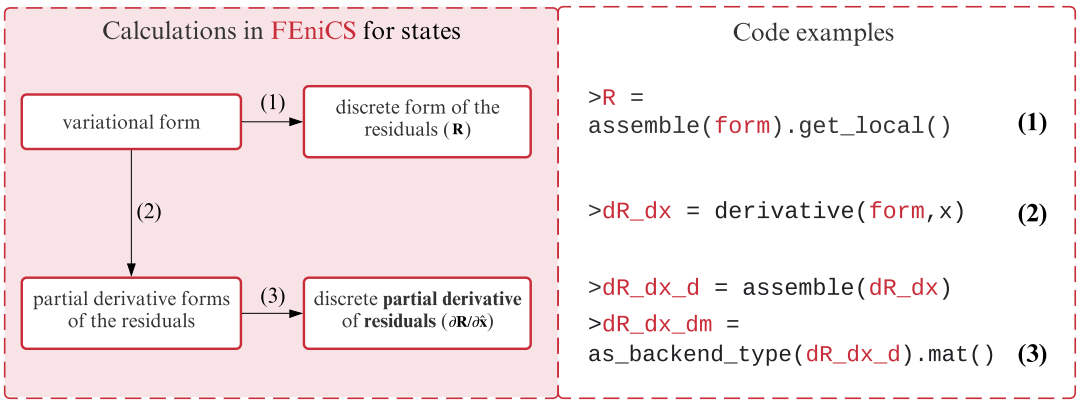


Figure 2: Demonstration of the computations in FEniCS with corresponding code examples.

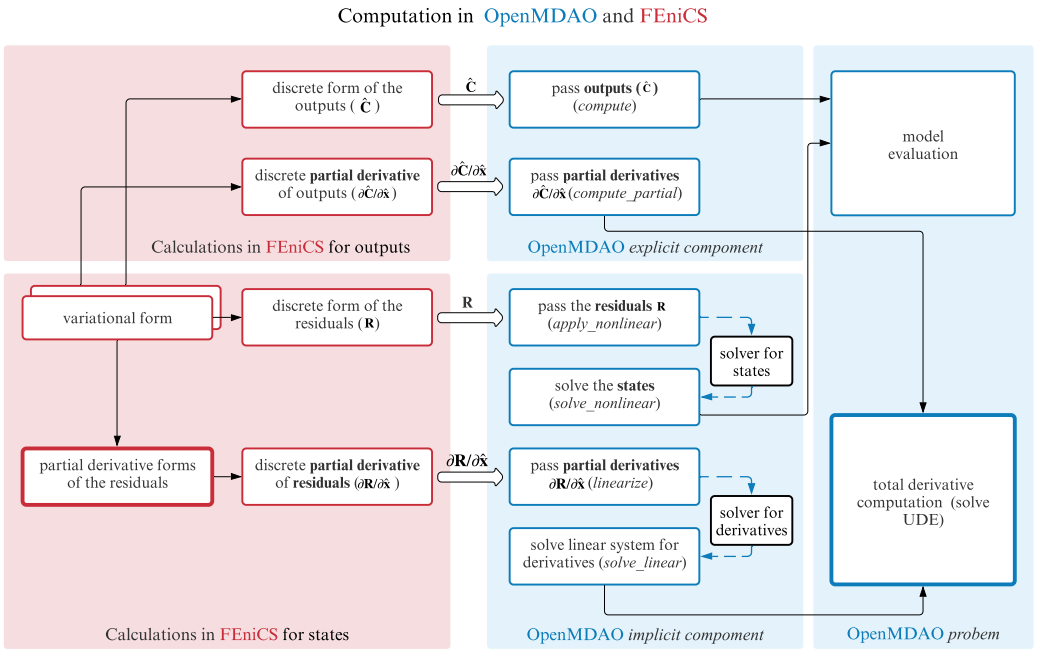


Figure 3: Method for computing the outputs, states, and outputs' derivatives using OpenMDAO and FEniCS. The red blocks (on the left) are computations in FEniCS. The blue blocks (on the right) denote the implementation in OpenMDAO with the function names shown in italics; The solvers (black boxes) can be implemented using OpenMDAO built-in functions, SciPy solvers, or PETSc solvers. The black arrows are used for passing forms to the next calculation; the hollow black arrows are used for passing the vectors or the matrices to OpenMDAO; the blue dashed arrows indicate that the function in the starting block is called by the ending block.

4.3 The overall methodology in ATOmiCS for fully automated derivative computation

This subsection provides a summary of the overall methodology in ATOmiCS that synergize OpenMDAO and FEniCS to automate the total and partial derivative computation (illustrated in Fig. 3). In OpenMDAO, models are implemented as a collection of *explicit components* and *implicit components* (introduced in Sec. 4.1). These two types of *components* have specific attribute functions (i.e., methods) that must be implemented to define how the outputs and partial derivatives of the *component* are computed given the inputs. These functions are listed in italics in the right half of Fig. 3.

We use OpenMDAO *explicit components* to compute the outputs $\hat{\mathbf{C}}$ (i.e., constraints and the objective of the optimization, as shown in Fig. 1). First, the variational form of the outputs and its derivatives are assembled into discrete vectors and matrices in FEniCS. Then, the output vectors are passed into the *compute* function, while the derivatives ($\partial\hat{\mathbf{C}}/\partial\hat{\mathbf{x}}$) with respect to $\hat{\boldsymbol{x}}$ (states and design variables) are passed into the *compute_partials* function in OpenMDAO's explicit *component* class.

The OpenMDAO *implicit component* (Sec. 4.1) class is used to perform the solution for the states. We send the discrete form of the residual vector $\boldsymbol{R}$ (Eq. (1)) as well as the symbolic derivatives generated by FEniCS to the OpenMDAO framework. First, we express the variational

form using the unified form language (UFL; modeling language of FEniCS) and assemble the residual vector. Next, ATOmiCS passes the residual vector into the *apply_nonlinear* function. The function *solve_nonlinear* then calls the solver for the states, which in turn calls *apply_nonlinear*, typically within the context of applying Newton's method. For general *implicit components*, *apply_nonlinear* represents the computation of the residual vector for an implicitly defined state, and *solve_nonlinear* represents the solution of the residual equations to compute the implicitly defined state.

Finally, the total derivatives are computed by solving the unified derivatives equation (introduced in Sec. 4.1). The whole process shown in Fig. 3 uses the combination of OpenMDAO and FEniCS to form a toolbox that is modular and flexible without manual derivative computation. Its unique implementation results in a general toolbox that solves topology optimization problems with various types of governing equations. In addition, the use of the OpenMDAO environment as the platform allows ATOmiCS to be extensible to complex problems with multiple disciplines.

5 Implementation

This section expands on the structure of ATOmiCS and the workflow for solving topology optimization problems with it. In the last subsection, we also discuss the experience of use ATOmiCS as an educational tool in a graduate-level class on multidisciplinary design optimization.

5.1 Toolbox structure

The structure of ATOmiCS is shown in Fig. 1. The first component, input *component*, contains the design variables (e.g., density of each element). The density vector from the input *component* is then set as the input of a general density filter with two options (a direct filter and a variational filter), as discussed in Sec. 3.3. The output of the general filter *component* is the filtered density vector $\tilde{\boldsymbol{\rho}}$. In the states *component* (an OpenMDAO implicit *component*), we implement SIMP and RAMP methods as options for the penalization (discussed in Sec. 3.2). Then, we solve for the states using FEniCS built-in solvers. For the solution of the derivatives in states *component*, the user can then choose between various direct solvers (e.g., *scipy splu* and *MUMPS*) and iterative solvers (e.g., *PETSc CG*, and *PETSc GMRES*) to solve the UDE for the total derivatives in OpenMDAO's *solve_linear* function. We implemented the variational forms of several governing equations for different problems in a PDE library, e.g., linear elasticity, hyperelasticity (neo-Hookean model); thermoelasticity; a modified elasticity model containing directional strains (for liquid crystal elastomers). The users can also specify their own PDEs to solve.

We define the constraints and the objective of the optimization in output *components*. There are two kinds of output *components*: the *scalar output component* and the *field output component*. The *scalar output component* outputs a scalar value on the entire domain, e.g., compliance, volume percentage. The *field output component* outputs scalar value on each element. For example, the von Mises stress is a field output that is a scalar on each element. The users are also able to add more optional pure OpenMDAO components for preprocessing and postprocessing as well. We then construct *ATOmiCS group* (the purple block in Fig. 1)—an OpenMDAO group instance that consists of the main contents of the topology optimization algorithms.

5.2 Workflow for solving a topology optimization problem using ATOmiCS

The workflow for using ATOmiCS to solve a topology optimization problem is documented online[2]. It requires modifying a single *run file* that contains four steps: 1) import relevant packages and functions (line 1-4 in Appendix A); 2) define the mesh (line 5-8 in Appendix A); 3) set up

the PDE problem including the boundary conditions (line 9-42 in Appendix A); and 4) set up the optimization model (line 43-74 in Appendix A).

First, the users need to import the *dolfin* PDE solver in FEniCS, *numpy*, *openmdao*, *atomics.api*, and *atomics.pdes*, *atomics.general_filter_comp*, and other related tools. Thus, the toolbox is general for many types of PDE problems, and easily switchable from one PDE to another just by importing from a different file (shown in the code below). The user can also implement their own variational form using the FEniCS UFL.

```
from atomics.pdes.<PDE name> import get_residual_form
```

Next, for the mesh input, ATOmiCS wraps the *meshio* mesh conversion tool so that it supports GMSH [46], SolidWorks, or mesh inputs from other CAD tools of *.stl* or *.vtk* file types, as well as FEniCS built-in meshes. We show an example of a FEniCS built-in rectangle-shaped mesh in line 5-8 of Appendix A.

Then, the user needs to define the PDE problem. An example can be found in line 9-42 of Appendix A. The user needs to add inputs, states, and outputs to the PDE problem. In addition, the user also needs to define the traction and kinematic boundary conditions. ATOmiCS supports FEniCS functions for specifying the boundary conditions. A typical FEniCS built-in function for defining the boundary is shown below. For the mesh defined above, this *Boundary*() class below selects four elements on middle of the right edge of the rectangle domain, where y (vertical) values are among the $\frac{y}{2}$, and x (horizontal) values are among largest values with a tolerance.

```
class Boundary(df.SubDomain):
    def inside(self, x, on_boundary):
        return ( (abs(x[1] - LENGTH_Y/2) <
                2 * LENGTH_Y/NUM_ELEMENTS_Y + df.DOLFIN_EPS )
                and (abs(x[0] - LENGTH_X) < df.DOLFIN_EPS) )
```

The user can also choose the solvers for the FEA problem and the linear solvers for the total derivatives by specifying the parameters for the *AtomicsGroup* class.

```
AtomicsGroup(PDEProblem,
             problem_type,
             linear_solver,
             visualization=False),
```

The *PDEProblem* is a dictionary object that contains the mesh, the PDEs, the name of the inputs, the state names, and the name of the outputs. The two options for the *problem_type* are: *linear_problem* and *nonlinear_problem*. The *linear_solver* options are shown in Fig. 1. When *visualization*=True, ATOmiCS saves the density profile and other outputs every certain number (50 by default) of optimization iterations.

Finally, the user needs to define the optimization problem, i.e., the design variables, the objective function, and the constraints (demonstrated in line 43-74 of Appendix A). The options for the outputs (the objective function and the constraints) are shown in Fig. 1. The user can also define customized outputs if needed. Appendix A shows an example of a typical compliance optimization problem that minimizes the compliance with respect to densities, subject to the volume constraint. The user can directly visualize optimization results using ParaView. The user can generate a video of the iteration history using ParaView-Python scripts and a script we recommended in the online tutorial.

5.3 Extensions and customizations

As a flexible toolbox, the user can further extend and customize the tool, such as implementing user-defined PDEs, adding additional design variables (e.g., case III shown in Sec. 6.3) and outputs, and setting up solvers beyond the options we provided. The advanced user guide section in the online documentation explains and demonstrates these functions [3].

Implementing a new physics problem ATOmiCS wraps FEniCS for the solution of the PDEs. The input for FEniCS is the variational (weak) form. To implement a physics problem that is not included in ATOmiCS, the user needs to implement the variational form using FEniCS unified form language (UFL) and add it to the ATOmiCS PDE folder (atomics/atomics/pdes). The user needs to import their customized function as demonstrated in line 3 of Appendix A.

Implementing customized outputs There are three types of outputs we defined in the ATOmiCS toolbox: 1) The output that can be directly computed using OpenMDAO. The interpolation component for case study III serves as an example of this kind. In order to formulate this kind of component, the user needs to refer to OpenMDAO documentation for the *explicit component*. 2) The computation that combines OpenMDAO and FEniCS and outputs a scalar on the entire domain. This kind of output is wrapped as *scalar_output* in ATOmiCS. 3) The computation that combines OpenMDAO and FEniCS and outputs a scalar on each vertex. This kind of output is wrapped as *field_output* in ATOmiCS. The user can define the second and third type of outputs directly in the *run file* using

```
pde_problem.add_scalar_output(<output_name>, <output_form>, <argument_names>)
pde_problem.add_field_output(<output_name>, <output_form>, <argument_names>),
```

where the <output_name> is a string of the name of the customized output; <output_form> is the form expressing the output using FEniCS UFL; <argument_names> are the name of arguments that specified by the user to take derivatives with respect to.

Extending the solver options If the user needs to implement a customized solver to compute the total derivative, they need to modify the *solve_linear()* function in *states_comp.py*. The user can refer to OpenMDAO documentation on implicit component for details. If the user wants to change to a customized solver for solving the finite element problem, they need to modify the *solve_nonlinear()* function in *states_comp.py*.

Adding additional design variables, constraints, or changing to a different objective The design variables, constraints, and the objective for the optimization are all specified in the *run file*. The design variables are specified using OpenMDAO independent variable component. The constraints and objective are the output for the optimization problem, which belongs to one of the three types of outputs described earlier in this Section. Then, the user needs to specify

```
prob.model.add_design_var(<design_var_name>,upper=upper,lower=lower)
prob.model.add_objective(<objective_name>)
prob.model.add_constraint(<constraint_name>,upper=upper_bd,lower=lower_bd)
```

to add those design variables, objective, and constraints to the problem.

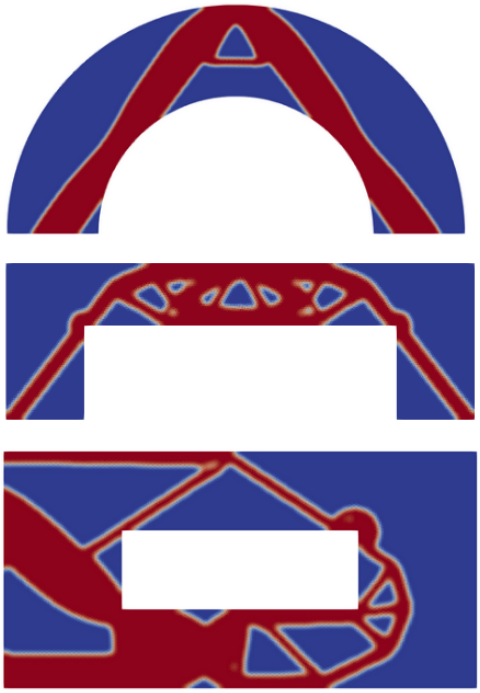


Figure 4: Selected topology optimization results from students in the MDO class.

5.4 Education

We used ATOmiCS in a graduate-level MDO class in the mechanical and aerospace engineering department at the University of California San Diego. Students were able to learn ATOmiCS by reading through its online documentation[2] and run the examples from the GitHub repository[1]. In a homework assignment designed to take four to six hours, students were asked to solve a topology optimization problem of their choice with arbitrary geometries (shown in Fig. 4), generated using external mesh tools such as GMSH or PYGMSH. Their examples involved unstructured meshes such as a bridge, an arch, a rectangular domain with holes, with 500 to 80000 elements. In addition, students also ran parameter sweeps to learn the effects of adjusting various parameters, such as filter radius, mesh size, and area fraction, on their own optimization problems.

Having ATOmiCS available for this and other classes provides students with first-hand experience working with a large-scale optimization problem without background knowledge for topology optimization and the effort of implementing the computations of the partial and total derivatives. Moreover, students can easily learn about the effects of different parameters in topology optimization through running parameter sweeps. In addition, the modular architecture of ATOmiCS allows the flexibility to explore problems with different governing equations, constraints, and objectives, by modify the corresponding modules, which brings opportunities for the students to apply ATOmiCS to their own research.

[3] `https://lsdolab.github.io/atomics/_src_docs/extension.html`

6 Case studies

This section presents two classical case studies: 1) topology optimizations of linear and nonlinear cantilever beam problems for compliance minimization (Sec. 6.1); 2) battery pack compliance minimization using a thermoelastic model (Sec. 6.2). We use these case studies to demonstrate the capability of ATOmiCS to solve various classical problems. Then, we also include an unconventional example—liquid crystal elastomer (LCE) optimization for shape matching (Sec. 6.3) to demonstrate the extensibility and flexibility of the toolbox. In order to provide the user with more information to construct these unconventional examples, we include detailed instructions on performing these extensions in the advanced user guide in our online documentation[3].

We demonstrate the unique advantages of ATOmiCS using the case studies: easily switchable PDE and solvers, dealing with nonlinear problems (case I); multiphysics approach, synergy with GMSH and ability to handle unstructured meshes (case II); topology optimization for shape matching of a smart material, 3D problem, adding preprocessing and postprocessing OpenMDAO *components*, and having design variables in addition to densities (case III).

We summarize case studies in Table 2. The parameters in the case studies are shown in Table 3. The organization of the subsequent sections for each case study is organized in the same structure: 1) introduction to the problem and the reason for choosing this particular problem; 2) formulation of the problem (model and optimization); and 3) optimization results and discussion.

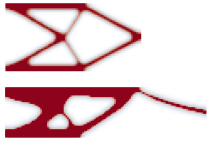
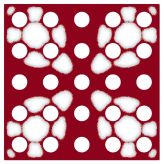
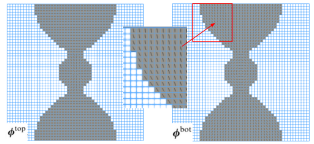
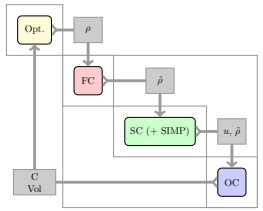
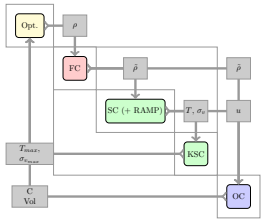
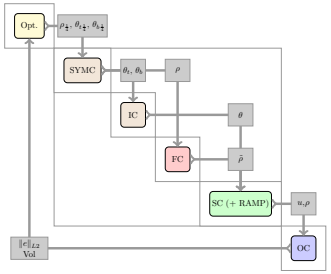
Table 3: Parameters for case studies

	Size of the domain $(x \times y(\times z))$	Young's modulus $E_{\max}$ [Pa]	Thermal conductivity κ $[Wm^{-1}K^{-1}]$	Thermal expansion coefficient α $[K^{-1}]$
Case I-linear	160×80	1		
Case I-nonlinear	0.12×0.3	8		
Case II	$0.2 \times 0.2 \times 0.05$	6.90×10^{10}	235	1.30×10^{-5}
Case III	$0.0025 \times 0.0025 \times 0.00005$	5.00×10^{6}		2.50×10^{-3}

6.1 Case study I: topology optimizations of cantilever beams

In this section, we perform 2D cantilever beam topology optimizations to show the flexibility of the framework—interchangeable PDEs (linear and nonlinear) and solvers for the states. Section 6.1.1 describes the modeling of the linear and nonlinear cantilever beam problems and the formulation of the optimizations. Section 6.1.2 introduces the optimization results.

Table 2: Summary of case studies

	Case I	Case II	Case III
Number of elements ($x \times y (\times z)$)	(a) 80×40 (b) 120×30	5428 (unstructured)	$50 \times 50 \times 4$
Assumption	(a) linear (b) nonlinear	linear	linear
Dimension	2D	2D	3D
Type	elasticity	thermoelasticity	modified elasticity (LCE)
Variational form	Eq. (8) and (10)	Eq. (17)	Eq. (8), (20), and (21)
Objective	compliance	compliance	shape matching error norm
Design variables	$\boldsymbol{\rho}$	$\boldsymbol{\rho}$	$\boldsymbol{\rho}$, $\boldsymbol{\phi_t}$, $\boldsymbol{\phi_b}$
Constraints	(a) $\int_\Omega \rho \, d\Omega = 40\%,\ 1 \times 10^{-5} < \boldsymbol{\rho} < 1$ (b) $\int_\Omega \rho \, d\Omega = 50\%,\ 5 \times 10^{-3} < \boldsymbol{\rho} < 1$	$\int_\Omega \tilde{\rho} \, d\Omega = 51\%,\ 1 \times 10^{-5} < \boldsymbol{\rho} < 1,$ $\mathrm{KS}(\mathbf{T}) < 50\,^\circ\mathrm{C}$	$\int_\Omega \rho \, d\Omega = 45\%,\ 1 \times 10^{-5} < \boldsymbol{\rho} < 1$
Results			
DSM			

* abbreviations DSM: data structure matrix; Opt.: optimizer; FC: filter *component*; SC: states *component*; OC: output *component*; SYMC: symmetric *component*; IC: interpolation *component*; Vol: volume constraint; C: compliance; IC: interpolation *component*; KSC: Kreisselmeier-Steinhauser *component*.

6.1.1 Case I: problem formulation:

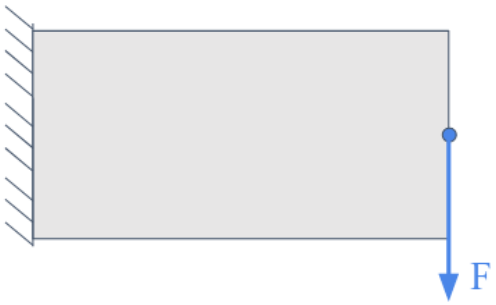


Figure 5: Boundary conditions for the cantilever beam problems.

Model description We formulate two optimization problems with their boundary and loading conditions depicted in Fig. 5. The structure is supported on the left edge with a traction load applied on four elements in the middle of the right edge. We model the beams using 2D linear and neo-Hookean hyperelastic elements. Table 3 shows the geometry parameters of the beams.

The weak form of the linear elastic problem is: Find a displacement $\mathbf{u} \in \mathcal{S}_u$ such that for all test functions $\mathbf{v} \in \mathcal{V}_u$,

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \nabla \mathbf{v} \, d\Omega - \int_\Omega \mathbf{b} \cdot \mathbf{v} \, d\Omega - \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{v} \, d\partial\Omega = 0 \ , \tag{8}$$

where

$$\boldsymbol{\sigma}(\mathbf{u}) = \frac{E}{1+\nu}\boldsymbol{\epsilon}(\mathbf{u}) + \frac{E\nu}{(1+\nu)(1-2\nu)} \operatorname{tr} \boldsymbol{\epsilon}(\mathbf{u})\mathbf{I} \tag{9}$$

is the Cauchy stress tensor, $\boldsymbol{\epsilon}$ is the symmetric gradient operator, E is Young's modulus, ν is the Poisson ratio, $\mathbf{b}$ is the body force density ($\mathbf{b} = 0$ in this problem), and $\mathbf{t}$ is the traction on the boundary. The spaces $\mathcal{S}_u$ and $\mathcal{V}_u$ are the trial and test function spaces for the displacement. Young's modulus E is the coefficient which we modify using the filtered density, as detailed in Sec. 3.2.

The weak form of the nonlinear elastic problem is: Find $\mathbf{u} \in \mathcal{S}_u$ such that for all $\mathbf{v} \in \mathcal{V}_u$,

$$D_{\mathbf{v}} \left(\int_\Omega \psi(\mathbf{u}) \, d\Omega - \int_\Omega \mathbf{b} \cdot \mathbf{u} \, d\Omega - \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u} \, d\partial\Omega \right) = 0 \ , \tag{10}$$

where $D_{\mathbf{v}}$ is a Gateaux derivative with respect to $\mathbf{u}$ in direction $\mathbf{v}$, $\psi(\mathbf{u})$ is the hyperelastic energy density, and $\mathbf{b}$ and $\mathbf{t}$ are the body force ($\mathbf{b} = 0$) and the traction force. The energy density is taken to be the compressible neo-Hookean potential

$$\psi(\mathbf{u}) = \frac{E}{2(1+\nu)} \left(\frac{1}{2}(\operatorname{tr} \mathbf{C} - 3) - \ln J \right) + \frac{E\nu}{2(1+\nu)(1-2\nu)} (\ln J)^2 \ , \tag{11}$$

where $\mathbf{C} = \mathbf{F}^T\mathbf{F}$, $\mathbf{F} = \mathbf{I} + \nabla\mathbf{u}$, $J = \det \mathbf{F}$, and E and ν are effective Young's modulus and Poisson ratio. As in the linear case, the Young's modulus is modified using the filtered density. To prevent convergence difficulties due to extreme deformations of low-density regions, the elastic potential ψ is augmented with a stabilizing term inspired by Liu et al. [47]:

$$\psi_{\text{stab}} = (1 - E_{\max}\tilde{\rho})(C_{1,e}(\operatorname{tr} \mathbf{C} - 3) + (C_{2,e}(I_C - 3))^2) \ , \tag{12}$$

where $C_{1,e} = E_{\max}\hat{\rho}_{\min}$.

The coefficient $C_{2,e}$ varies from optimization iterations, according to

$$C_{2,e}^{(k+1)} = \begin{cases} C_{2,e}^{(k)} \sqrt{\eta_e^{(k)}}, & \text{if } \eta_e^{(k)} < 1 \\ C_{2,e}^{(k)} \cdot \left(\eta_e^{(k)}\right)^3, & \text{if } \eta_e^{(k)} \geq 1 \end{cases} , \tag{13}$$

where k and $k+1$ denotes the current and the next optimization iteration; and η_e is defined as

$$\eta_e = \frac{1}{\epsilon^*} \sqrt{\frac{3}{2} \mathbf{E}_{\text{dev}} : \mathbf{E}_{\text{dev}}} , \tag{14}$$

where $\mathbf{E}_{\text{dev}}$ is the deviatoric part of the Green–Lagrange strain $\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I})$, and ϵ^* is a dimensionless constant with a default value of 0.2.

Optimization formulation We minimize compliance,

$$C = \int_\Omega \mathbf{b} \cdot \mathbf{u} \, d\Omega + \int_{\partial\Omega} \mathbf{t} \cdot \mathbf{u} \, d\partial\Omega , \tag{15}$$

with respect to the densities of the elements, subject to constant volume-fraction constraints (40% for the linear elastic problem and 50% for the nonlinear elastic problem):

$$\begin{aligned} \textbf{minimize} \quad & C \\ \textbf{with respect to} \quad & \boldsymbol{\rho}, \\ \textbf{subject to} \quad & \int_\Omega \rho \, d\Omega = (\text{volume fraction})|\Omega| \\ & 1 \times 10^{-5} < \boldsymbol{\rho} < 1 \end{aligned} \tag{16}$$

Implementation details The detailed documentation of Case I can be found in our tutorial[4]. We first define a mesh using FEniCS built-in *RectangleMesh* function. Then, we set up the PDE problem, including the governing equation, boundary conditions, and the outputs. The functions returning variational forms for linear and nonlinear elasticity can be imported from ATOmiCS via

```
from atomics.pdes.linear_elastic import get_residual_form
```

and

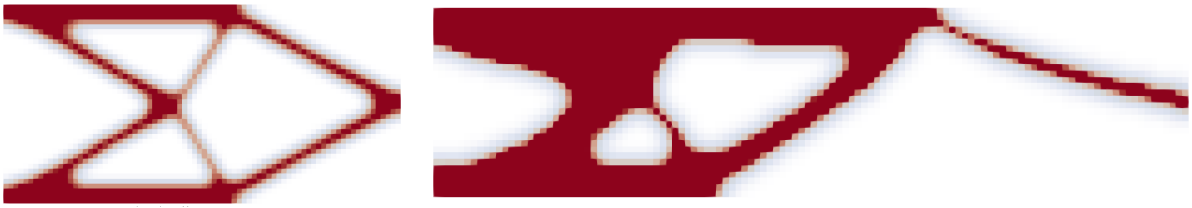
```
from atomics.pdes.hyperelastic_neo_hookean import get_residual_form ,
```

respectively. Finally, we specify the OpenMDAO model by adding the *components* and *groups* to the OpenMDAO *problem*, and defining the objective, constraints, and design variables.

6.1.2 Case I: optimization results

We use a direct solver (*MUMPS*) to solve the states for the linear problem, and a Newton solver with a line search (*SNES*) for the nonlinear problem. The penalized densities are shown in Fig. 6. The optimizer forms a tapered shape for both cases to reduce the compliance value. Our geometry for the linear elastic case is the same as that studied by Chung et al. [18], and we are able to recover the same optimized structure. The nonlinear case results in asymmetric shapes due to the nonlinear behavior of the material.

[4] `https://lsdolab.github.io/atomics/_src_docs/examples/case_studies/case_1.html`



(a) (b)

Figure 6: Topology optimization of linear and nonlinear cantilever-beam compliance minimization problems. (a) topology optimization results with a linear elastic model; (b) topology optimization result with a nonlinear neo-Hookean model.

6.2 Case study II: topology optimization of a battery pack

This case study demonstrates the topology optimization of a battery pack as explored in previous work [48]. The battery pack structure has battery cell arrays inserted on it, which can be used as a compact power source for electric aircraft. It serves the functions of both carrying load and dissipating heat. A maximum working temperature constraint is needed to prevent the batteries from failing. We use the design of the battery pack as a test case for the multiphysics (i.e., thermoelastic) problem. Section 6.2.1 describes the modeling of the battery pack and the formulation of the optimizations. Section 6.2.2 introduces the optimization results.

6.2.1 Case II: problem formulation

Thermoelastic analysis The material surrounding battery cells is modeled as a thermoelastic continuum. Its displacement ($\mathbf{u}$) and temperature (T) fields satisfy the following weak problem: Find $(\mathbf{u}, T) \in \mathcal{S}_u \times \mathcal{S}_T$ such that for all $(\mathbf{v}, \widehat{T}) \in \mathcal{V}_u \times \mathcal{V}_Q$

$$\int_\Omega \kappa \nabla T \nabla \widehat{T}\, d\Omega + \int_\Omega \boldsymbol{\sigma}(\mathbf{u}, T) : \nabla \mathbf{v}\, d\Omega - \int_\Omega Q\widehat{T}\, d\Omega - \int_\Omega \mathbf{b}\cdot\mathbf{v}\, d\Omega - \int_{\partial\Omega} q\widehat{T}\, d\partial\Omega - \int_{\partial\Omega} \mathbf{t}\cdot\mathbf{v} d\partial\Omega = 0\ , \tag{17}$$

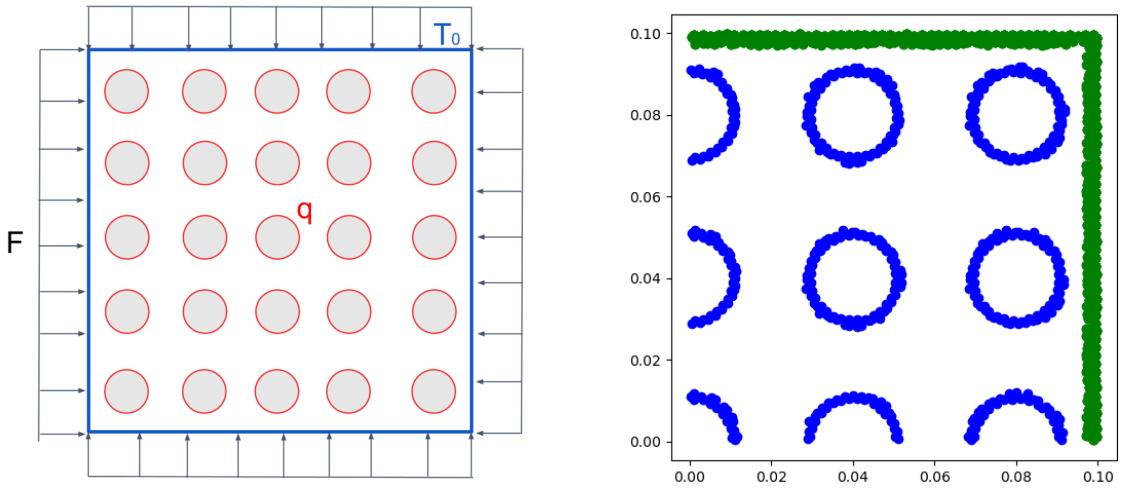
where κ is the thermal conductivity, $\mathbf{b} = 0$ is the body force density, Q is a distributed heat sources, $\mathbf{t} = 1\times10^6\,\mathrm{Nm}^{-1}$ is the boundary traction, and q is the prescribed normal heat flux at the boundary. The thermoelastic Cauchy stress $\boldsymbol{\sigma}(\mathbf{u}, T)$ is given by (9), but with the substitution

$$\boldsymbol{\epsilon}(\mathbf{u}) \quad \rightarrow \quad \boldsymbol{\epsilon}(\mathbf{u}) - \alpha(T - T_r)\mathbf{I} \tag{18}$$

to account for thermal strains, where α is the thermal expansion coefficient, and $T_r = 20\,^\circ\mathrm{C}$ is the reference temperature. For topology optimization with this model, both the thermal conductivity and Young's modulus are modulated by penalized density.

We model the heat generated by battery cells through the prescribed heat flux q on boundaries abutting the cells. The material of the battery pack is aluminum, with $\kappa = 235\,\mathrm{W/(m\cdot K)}$ and $\alpha = 1.3\times10^{-5}\,\mathrm{K}^{-1}$. The length and width of the battery pack are both 0.2 m, and the depth is 0.05 m. Thus, we use a 2D plane stress model. The radii of the holes for the batteries are 0.01 m.

The boundaries in the problem are shown in Fig. 7. The red rings denote the heat flux boundaries used to model the heat dissipation for the 5×5 cell array of the battery pack. The equivalent heating power of each cell is 90 W. The surrounding blue edge denotes the constant temperature heat sink with $\mathbf{T_s} = 20.0\,^\circ\mathrm{C}$. A traction force of magnitude $1\times10^6\,\mathrm{N/m}^2$ is also applied on the surrounding



(a) (b)

Figure 7: Geometry and the boundary conditions of the battery pack problem. (The surrounding blue contour is the constant temperature boundary; the red circles are the heat flux boundaries; the arrows are the traction forces applied on the battery pack). The right figure shows the constrained densities corresponding to $\boldsymbol{\rho}_{\text{selec}}$.

edge with their directions shown in Fig. 7. For simplicity, we model the top right one-quarter of the structure and apply sliding boundary conditions since the structure is symmetric.

Optimization formulation We perform an optimization to minimize the compliance of the battery pack. We formulate this problem (shown in Eq. (19)) as minimizing an objective function C (compliance) with respect to the density of each element. The constraints of the optimization include a density constraint, a maximum temperature constraint, a constraint $\boldsymbol{\rho}_{\mathbf{selec}} = 1$ on the edges, and the elements surrounding the battery cells (Fig. 7) to preserve the key geometry, and a fixed volume constraint. The maximum temperature constraint is implemented using a constraint aggregation method called Kreisselmeier–Steinhauser function [49], which is implemented as a stock *component* in ATOmiCS.

$$
\begin{aligned}
\textbf{minimize} \quad & C \\
\textbf{with respect to} \quad & \boldsymbol{\rho}, \\
\textbf{subject to} \quad & 1 \times 10^{-5} < \boldsymbol{\rho} < 1 \\
& \mathrm{KS}(\mathbf{T}) \leq 50\,^{\circ}\mathrm{C} \\
& \boldsymbol{\rho}_{\mathbf{selec}} = 1 \\
& \int_{\Omega} \tilde{\rho}\, d\Omega = 51\%
\end{aligned}
\tag{19}
$$

Implementation details The detailed documentation of Case II can be found in our tutorial[5]. The first step is similar to the previous case study, except that the mesh is unstructured and generated

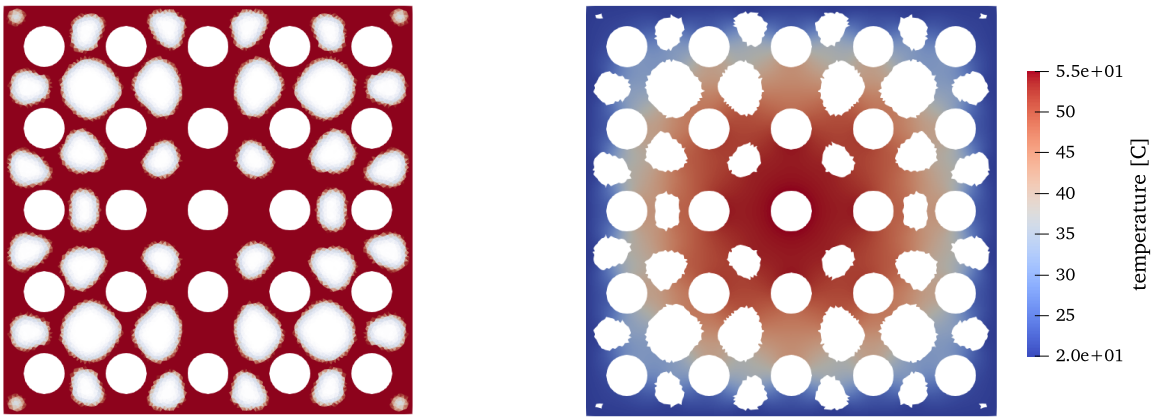
[5] `https://lsdolab.github.io/atomics/_src_docs/examples/case_studies/case_2.html`

in PYGMSH. The user can specify the PDE corresponding to the thermoelastic problem in the *run file* with a command of the form:

```
from atomics.pdes.thermo_mechanical_mix_2d_stress import get_residual_form.
```

Finally, we set up the OpenMDAO model with a similar approach as Case I.

6.2.2 Case II: optimization results



(a) (b)

Figure 8: Results for the battery pack compliance minimization; (a) the optimized topology; (b) the temperature field.

We compare the results with the work from Kambampati et al. [48]. They performed three optimizations for the battery pack: stress minimization with volume and maximum temperature constraints, compliance minimization with volume and maximum temperature constraints, and mass minimization subject to maximum stress and temperature constraints. Here, we solve the second optimization problem, which was shown by Kambampati et al. to yield a solution at which the maximum temperature constraint is active [48].

We validate our battery pack optimization with the same physical model, subject to a maximum temperature constraint of 55 °C (the same setting as Kambampati et al.), and a volume constraint of 51% (13.33% larger than the volume constraint in the reference). The penalized density and the temperature field of the compliance minimization are shown in Fig. 8. As is shown in Fig. 8, the maximum temperature constraint is active. Furthermore, the maximum von Mises stress for the compliance optimization has been reported as 106 MPa in the reference, while our optimization results in a maximum von Mises stress of 111.35 MPa, which is a 5.04% difference. The compliance of our optimized structure is 204.79 N/m. The compliance value was not reported in the study by Kambampati et al.

Overall, our optimization results agree well with the numerical results from Kambampati et al. with some minor differences. We believe the differences mainly come from two aspects. First, Kambampati et al. have used the level-set method as opposed to the density-based method used in this work. Thus, some minor discrepancies in the results are to be expected. Furthermore, we used a fixed volume constraint of 51% on the penalized density $\tilde{\rho}$, while Kambampati et al. have used

a volume constraint of 45%. This choice is made based on numerical experimentation to produce a qualitatively similar result, accounting for the grey regions that appear due to our density-based topology optimization approach and are not present in the level-set approach.

6.3 Case study III: LCE shape matching optimization

This case study demonstrates the application of topology optimization to the shape matching of liquid crystal elastomer (LCE). We formulate a similar problem setting as previously described by Fuchi et al. [50] to test our toolbox. Liquid crystal elastomers are attractive candidates for the actuators of soft robots. The LCE is often fabricated as thin films composed of ordered and disordered regions. In the disordered region, the material does not contract or expand due to the change of temperature. In the ordered region, the LCE material contracts along the director orientations and expands in the other two perpendicular directions when it is heated. Thus, we can form bending and other complex deformation using multiple layers of LCE films.

This case study serves as a novel application for topology optimization used as shape matching for smart material, adding user-defined preprocessing and postprocessing components from OpenMDAO, and a test for 3D elements in ATOmiCS. Section 6.3.1 describes the modeling of the LCE and the formulation of the optimizations for the shape matching application. Section 6.3.2 introduces the optimization results.

6.3.1 Problem formulation

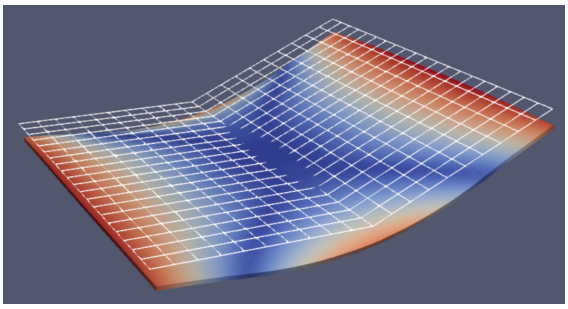


Figure 9: LCE shape matching problem with a target shape of a 2.5° pure folding on each side. The white wireframe is the target deformation. The gradient color is the actual deformation after optimization.

Model description We formulate the LCE shape matching problem with a target shape of a 2.5° pure folding on each side (shown in Fig. 9). The magnitude of the strain for the ordered element is set as a constant for a spatially uniform temperature change ΔT. We model the LCE as four layers of films using 3D elements. The director angles on each of the elements on the top and bottom layers are set as design variables to match the target shape. The director angles on the middle two layers are linearly interpolated from the top and bottom director angles that are vertically aligned with them.

The LCE continuum is modeled using the linear elastic problem (8), but modifying the definition of the Cauchy stress (9) through the substitution

$$\boldsymbol{\epsilon}(\mathbf{u}) \quad \rightarrow \quad \boldsymbol{\epsilon}(\mathbf{u}) - \alpha \Delta T \mathbf{L}^T \mathbf{S} \mathbf{L} \ , \tag{20}$$

to include anisotropic material expansion. The matrices $\mathbf{L}$ and $\mathbf{S}$ are defined by

$$\mathbf{L} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{S} = \begin{bmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{21}$$

where θ is the director orientation angle, and α is the principal expansion coefficient.

Unlike in the compliance minimization problem, we are not using density to determine material or void, but rather to determine where LCE actuation occurs (i.e., where α is nonzero). Thus, Young's modulus E is not modulated by density, but the expansion coefficient α is. The user can specify the PDE corresponding to the LCE shape matching problem in the *run file* by

```
from atomics.pdes.thermo_mechanical_lce import get_residual_form
```

Optimization formulation

$$\begin{aligned} \textbf{minimize} \quad & \int_\Omega \frac{(\boldsymbol{u} - \boldsymbol{u}^*)^2}{V} dx \\ \textbf{with respect to} \quad & \boldsymbol{\rho},\ \boldsymbol{\phi}_{\text{top}},\ \boldsymbol{\phi}_{\text{bot}} \\ \textbf{subject to} \quad & \int_\Omega \rho \, d\Omega = 45\% \\ & 1 \times 10^{-5} < \boldsymbol{\rho} < 1 \end{aligned} \tag{22}$$

We formulate this problem (shown in Eq. 22) as minimizing the L_2 error of the difference of the deformed deformations $\boldsymbol{u}$ and the target deformations $\boldsymbol{u}^*$ scaled by a constant V (total volume), by design the order region and the director orientations on the top and bottom layers ($\boldsymbol{\phi}_{\text{top}}$ and $\boldsymbol{\phi}_{\text{bot}}$). The optimization is subject to a volume fraction constraint—the ordered region is 45% of the total volume.

6.3.2 Optimization results

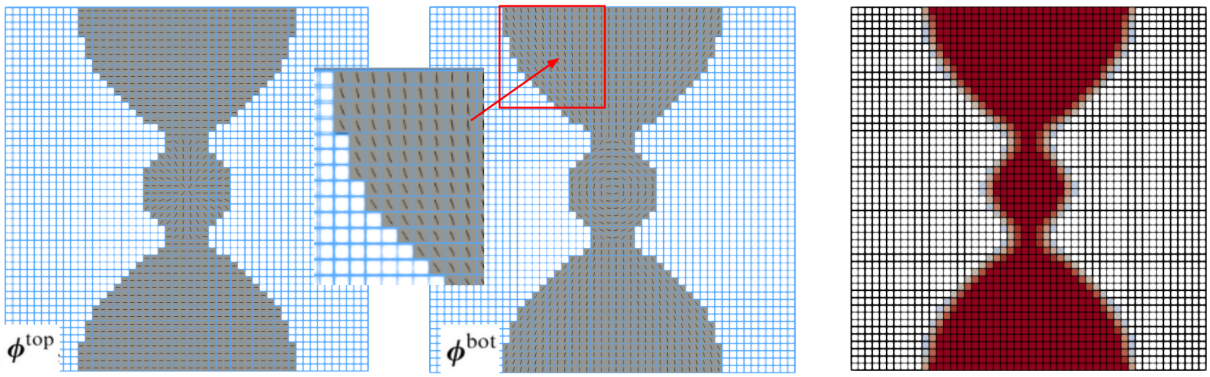


Figure 10: Optimization results for LCE shape matching problem: (a) top layer director angles ($\boldsymbol{\phi}_{\text{top}}$); (b) bottom layer director angles ($\boldsymbol{\phi}_{\text{bot}}$); (c) the ordered region (shown in red).

The optimized director angles on the top and bottom layers and the ordered region are shown in Fig. 10. The optimizer forms horizontal director orientations on the top layer to realize a horizontal contraction on the top layer. In addition, there are more vertical director angles on the bottom

layer so that the bottom layer expanses horizontally. The effect of the top layer contraction and bottom layer expansion forms the target folding shape.

7 Conclusion and future work

In this work, we presented ATOmiCS, a modular topology optimization toolbox with automated derivative computation. In Sec. 2, we reviewed recent educational topology optimization papers with a focus on multiphysics problems and the methods previously used for derivative computation. Then, we reviewed density-based topology optimization methods that are used in ATOmiCS in Sec. 3. Section 4 described the methodology—the modular construction of the toolbox, the use of the unified form language for building up the variational form, the partial derivative computation, and the synergy of OpenMDAO and FEniCS for fully automating the derivative computation. In Sec. 5, we introduced the procedure for performing a topology optimization in ATOmiCS, and the experience of using ATOmiCS as an educational tool in a graduate-level class. Section 6 then demonstrated the flexibility and the generality by solving three case studies with different physics equations, objectives, design variables, and constraints, all within ATOmiCS.

ATOmiCS is an open-source tool for multiphysics problems that is modular and flexible with less implementation effort required from the user than with implementations that are currently available. This reduction in implementation effort is achieved by fully automating the derivative computation and the solution of the PDE, beyond the specification of the variational form. Furthermore, the use of OpenMDAO leads to a modular and flexible toolbox. Thus, the users can easily change the governing equation, penalizations, solvers, constraints, and objectives by modifying or selecting the corresponding module. The implementation in an MDO framework such as OpenMDAO also makes ATOmiCS extensible to multidisciplinary problems. In addition, ATOmiCS leverages version control on GitHub and automatically generates documentation, making it a high-quality software both for educational and research purposes.

A Example code of a linear elastic cantilever beam topology optimization

1. Installation and imports: The user needs to install FEniCS, OpenMDAO, and ATOmiCS (SNOPT optimizer is recommended but not required). Then, the user needs to import the functions as shown below.

```
import dolfin as df; import numpy as np; import openmdao.api as om
from atomics.api import PDEProblem, AtomicsGroup
from atomics.pdes.linear_elastic import get_residual_form
from atomics.general_filter_comp import GeneralFilterComp
```

2. Define the mesh: The second step is to define the mesh. The example below shows a 2D rectangular mesh generated by FEniCS built-in functions.

```
NUM_ELEMENTS_X = 80; NUM_ELEMENTS_Y = 40
LENGTH_X = 160.; LENGTH_Y = 80.
mesh = df.RectangleMesh.create([df.Point(0., 0.), df.Point(LENGTH_X, LENGTH_Y)],
    [NUM_ELEMENTS_X, NUM_ELEMENTS_Y], df.CellType.Type.quadrilateral)
```

3. Setup the PDE problem: The third step is to set up the PDE problem. This includes several substeps: 1) add inputs to the problem; 2) add states to the problem; 3) define traction boundary conditions; 4) add outputs to the problem; and 5) add kinematic boundary conditions.

```
pde_problem = PDEProblem(mesh)
# Add inputs to the PDE problem:
# input = density (Inputs can be multiple, but here 'density' is the only input.)
density_function_space = df.FunctionSpace(mesh, 'DG', 0)
density_function = df.Function(density_function_space)
pde_problem.add_input('density', density_function)
# Add states to the PDE problem
displacements_function_space = df.VectorFunctionSpace(mesh, 'Lagrange', 1)
displacements_function = df.Function(displacements_function_space)
v = df.TestFunction(displacements_function_space)
method = 'SIMP' # penalization method
residual_form = get_residual_form(displacements_function, v,
    density_function, method=method) - df.dot(f, v) * dss(6)
# Define the traction boundary conditions
class TractionBoundary(df.SubDomain):
    def inside(self, x, on_boundary):
        return ((abs(x[1] - LENGTH_Y/2) < LENGTH_Y/NUM_ELEMENTS_Y + df.DOLFIN_EPS)
            and (abs(x[0] - LENGTH_X ) < df.DOLFIN_EPS*1.e3))
sub_domains = df.MeshFunction('size_t', mesh, mesh.topology().dim() - 1)
upper_edge = TractionBoundary(); upper_edge.mark(sub_domains, 6)
dss = df.Measure('ds')(subdomain_data=sub_domains)
f = df.Constant((0, -1. / 4 )) # traction force magnitude
pde_problem.add_state('displacements',
    displacements_function, residual_form, 'density')
# Add output-avg_density to the PDE problem:
volume = df.assemble(df.Constant(1.) * df.dx(domain=mesh))
avg_density_form = density_function/(df.Constant(1.* volume))*df.dx(domain=mesh)
pde_problem.add_scalar_output('avg_density', avg_density_form, 'density')
# Add output-compliance to the PDE problem:
compliance_form = df.dot(f, displacements_function) * dss(6)
pde_problem.add_scalar_output('compliance', compliance_form, 'displacements')
# Add Dirichlet boundary conditions to the PDE problem:
pde_problem.add_bc(df.DirichletBC(displacements_function_space,
    df.Constant((0., 0.)), '(abs(x[0]-0.) < DOLFIN_EPS)'))
```

4. Define the optimization problem in OpenMDAO: The last step is to define the optimization problem. This includes: 1) adding subsystems (*groups* and *components*), design variables, the objective, and constraints to the optimization problem; 2) setting up the optimizer (*driver*); and 3) running the optimization and saving the optimized results.

```
prob = om.Problem()
num_dof_density = pde_problem.inputs_dict['density'][
    'function'].function_space().dim()
om.IndepVarComp().add_output('density_unfiltered', shape=num_dof_density,
    val=np.random.random(num_dof_density) * 0.86)
prob.model.add_subsystem('indep_var_comp', comp, promotes=['*'])
```

```
comp = GeneralFilterComp(density_function_space=density_function_space)
prob.model.add_subsystem('general_filter_comp', comp, promotes=['*'])
group = AtomicsGroup(pde_problem=pde_problem)
prob.model.add_subsystem('atomics_group', group, promotes=['*'])
prob.model.add_design_var('density_unfiltered',upper=1, lower=1e-4)
prob.model.add_objective('compliance')
prob.model.add_constraint('avg_density',upper=0.40)
# set up the driver (i.e., optimizer)
try:
    prob.driver = driver = om.pyOptSparseDriver()
    driver.options['optimizer'] = 'SNOPT'
    driver.opt_settings['Major feasibility tolerance'] = 1.0e-6
    driver.opt_settings['Major optimality tolerance'] = 1.e-8
except:
    print('SNOPT is not installed, setting SciPy as the optimizer')
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options['optimizer'] = 'SLSQP'
prob.setup(); prob.run_driver()
# save the solution vector
if method =='SIMP':
    penalized_density = df.project(density_function**3, density_function_space)
else:
    penalized_density = df.project(density_function/(1+8.*(1.-density_function)),
        density_function_space)
df.File('solutions/displacement.pvd') << displacements_function
df.File('solutions/penalized_density.pvd') << penalized_density
```


Acknowledgments

This work was in part supported by NASA Grant No. 80NSSC21M0070. The authors also thank Luca Scotzniovsky, Xiangbei Liu, and Mark Sperry for sharing their assignment results from a graduate-level multidisciplinary design optimization course at UC San Diego to demonstrate the application of the ATOmiCS toolbox in a classroom setting.

Declarations

Replication of results The source code of ATOmiCS, and the case studies discussed in this work have been provided in our open-source GitHub repository (`http://github.com/LSDOlab/atomics`). The users can also refer to the online documentation (`http://lsdolab.github.io/atomics`) for implementation details.

Conflict of interests The authors declare that they have no conflict of interest.

References

[1] Martin Philip Bendsoe and Ole Sigmund. *Topology optimization: theory, methods, and applications*. Springer Science & Business Media, 2013.

[2] Ole Sigmund and Kurt Maute. "Topology optimization approaches". In: *Structural and Multidisciplinary Optimization* 48.6 (2013), pp. 1031–1055.

[3] Joshua D Deaton and Ramana V Grandhi. "A survey of structural and multidisciplinary continuum topology optimization: post 2000". In: *Structural and Multidisciplinary Optimization* 49.1 (2014), pp. 1–38.

[4] Guanghui Shi, Chengqi Guan, Dongliang Quan, Dongtao Wu, Lei Tang, and Tong Gao. "An aerospace bracket designed by thermo-elastic topology optimization and manufactured by additive manufacturing". In: *Chinese Journal of Aeronautics* 33.4 (2020), pp. 1252–1259.

[5] Hayoung Chung, Oded Amir, and H Alicia Kim. "Level-set topology optimization considering nonlinear thermoelasticity". In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112735.

[6] Chandrashekhar Jog. "Distributed-parameter optimization and topology design for non-linear thermoelasticity". In: *Computer Methods in Applied Mechanics and Engineering* 132.1-2 (1996), pp. 117–134.

[7] H Rodrigues and P Fernandes. "A material based model for topology optimization of thermoelastic structures". In: *International Journal for Numerical Methods in Engineering* 38.12 (1995), pp. 1951–1965.

[8] Joshua D Deaton and Ramana V Grandhi. "Stiffening of restrained thermal structures via topology optimization". In: *Structural and multidisciplinary optimization* 48.4 (2013), pp. 731–745.

[9] Joshua D Deaton and Ramana V Grandhi. "Stress-based design of thermal structures via topology optimization". In: *Structural and Multidisciplinary Optimization* 53.2 (2016), pp. 253–270.

[10] Thomas Borrvall and Joakim Petersson. "Topology optimization of fluids in Stokes flow". In: *International journal for numerical methods in fluids* 41.1 (2003), pp. 77–107.

[11] Allan Gersborg-Hansen, Ole Sigmund, and Robert B Haber. "Topology optimization of channel flow problems". In: *Structural and multidisciplinary optimization* 30.3 (2005), pp. 181–192.

[12] Joe Alexandersen and Casper Schousboe Andreasen. "A review of topology optimisation for fluid-based problems". In: *Fluids* 5.1 (2020), p. 29.

[13] Anderson Pereira, Cameron Talischi, Glaucio H Paulino, Ivan FM Menezes, and Marcio S Carvalho. "Fluid flow topology optimization in PolyTop: stability and computational implementation". In: *Structural and Multidisciplinary Optimization* 54.5 (2016), pp. 1345–1364.

[14] Chang-Hwan Im, Hyun-Kyo Jung, and Yong-Joo Kim. "Hybrid genetic algorithm for electromagnetic topology optimization". In: *IEEE Transactions on Magnetics* 39.5 (2003), pp. 2163–2169.

[15] Shiwei Zhou, Wei Li, and Qing Li. "Level-set based topology optimization for electromagnetic dipole antenna design". In: *Journal of Computational Physics* 229.19 (2010), pp. 6915–6930.

[16] Antoine Laurain. "A level set-based structural optimization code using FEniCS". In: *Structural and Multidisciplinary Optimization* 58.3 (2018), pp. 1311–1334.

[17] Anders Logg, Kent-Andre Mardal, and Garth Wells. *Automated solution of differential equations by the finite element method: The FEniCS book*. Vol. 84. Springer Science & Business Media, 2012.

[18] Hayoung Chung, John T. Hwang, Justin S. Gray, and H. Alicia Kim. "Topology optimization in OpenMDAO". In: *Structural and multidisciplinary optimization* 59.4 (2019), pp. 1385–1400. DOI: 10.1007/s00158-019-02209-7.

[19] Justin S. Gray, John T. Hwang, Joaquim R. R. A. Martins, Kenneth T. Moore, and Bret A. Naylor. "OpenMDAO: An open-source framework for multidisciplinary design, analysis, and optimization". In: *Structural and Multidisciplinary Optimization* 59.4 (Apr. 2019), pp. 1075–1104. DOI: 10.1007/s00158-019-02211-z.

[20] Peng Wei, Zuyu Li, Xueping Li, and Michael Yu Wang. "An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions". In: *Structural and Multidisciplinary Optimization* 58.2 (2018), pp. 831–849.

[21] Masaki Otomori, Takayuki Yamada, Kazuhiro Izui, and Shinji Nishiwaki. "Matlab code for a level set-based topology optimization method using a reaction diffusion equation". In: *Structural and Multidisciplinary Optimization* 51.5 (2015), pp. 1159–1172.

[22] Benliang Zhu, Xianmin Zhang, Hai Li, Junwen Liang, Rixin Wang, Hao Li, and Shinji Nishiwaki. "An 89-line code for geometrically nonlinear topology optimization written in FreeFEM". In: *Structural and Multidisciplinary Optimization* 63.2 (2021), pp. 1015–1027.

[23] Qi Chen, Xianmin Zhang, and Benliang Zhu. "A 213-line topology optimization code for geometrically nonlinear structures". In: *Structural and Multidisciplinary Optimization* 59.5 (2019), pp. 1863–1879.

[24] Niels Aage, Erik Andreassen, and Boyan Stefanov Lazarov. "Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework". In: *Structural and Multidisciplinary Optimization* 51.3 (2015), pp. 565–572.

[25] Morten Nobel-Jørgensen, Niels Aage, Asger Nyman Christiansen, Takeo Igarashi, J Andreas Bærentzen, and Ole Sigmund. "3D interactive topology optimization on hand-held devices". In: *Structural and Multidisciplinary Optimization* 51.6 (2015), pp. 1385–1391.

[26] Haidong Lin, An Xu, Anil Misra, and Ruohong Zhao. "An ANSYS APDL code for topology optimization of structures with multi-constraints using the BESO method with dynamic evolution rate (DER-BESO)". In: *Structural and Multidisciplinary Optimization* 62.4 (2020), pp. 2229–2254.

[27] Abbas Homayouni-Amlashi, Thomas Schlinquer, Abdenbi Mohand-Ousaid, and Micky Rakotondrabe. "2D topology optimization MATLAB codes for piezoelectric actuators and energy harvesters". In: *Structural and Multidisciplinary Optimization* 63.2 (2021), pp. 983–1014.

[28] Yuan Liang and Gengdong Cheng. "Topology optimization via sequential integer programming and canonical relaxation algorithm". In: *Computer Methods in Applied Mechanics and Engineering* 348 (2019), pp. 64–96.

[29] Rubén Ansola Loyola, Osvaldo M Querin, Alain Garaigordobil Jiménez, and Cristina Alonso Gordoa. "A sequential element rejection and admission (SERA) topology optimization code written in Matlab". In: *Structural and Multidisciplinary Optimization* 58.3 (2018), pp. 1297–1310.

[30] Martin P Bendsøe and Ole Sigmund. "Material interpolation schemes in topology optimization". In: *Archive of applied mechanics* 69.9 (1999), pp. 635–654.

[31] Xuefeng Zhu, Chao Zhao, Xuan Wang, Yu Zhou, Ping Hu, and Zheng-Dong Ma. "Temperature-constrained topology optimization of thermo-mechanical coupled problems". In: *Engineering Optimization* (2019).

[32] Michael Bruyneel and Pierre Duysinx. "Note on topology optimization of continuum structures including self-weight". In: *Structural and Multidisciplinary Optimization* 29.4 (2005), pp. 245–256.

[33] Tong Gao and Weihong Zhang. "Topology optimization involving thermo-elastic stress loads". In: *Structural and multidisciplinary optimization* 42.5 (2010), pp. 725–738.

[34] Tong Gao, Pengli Xu, and Weihong Zhang. "Topology optimization of thermo-elastic structures with multiple materials under mass constraint". In: *Computers & Structures* 173 (2016), pp. 150–160.

[35] Ole Sigmund. "Morphology-based black and white filters for topology optimization". In: *Structural and Multidisciplinary Optimization* 33.4-5 (2007), pp. 401–424.

[36] Boyan Stefanov Lazarov and Ole Sigmund. "Filters in topology optimization based on Helmholtz-type differential equations". In: *International Journal for Numerical Methods in Engineering* 86.6 (2011), pp. 765–781.

[37] Atsushi Kawamoto, Tadayoshi Matsumori, Shintaro Yamasaki, Tsuyoshi Nomura, Tsuguo Kondoh, and Shinji Nishiwaki. "Heaviside projection based topology optimization by a PDE-filtered scalar function". In: *Structural and Multidisciplinary Optimization* 44.1 (2011), pp. 19–24.

[38] John T. Hwang and Joaquim R. R. A. Martins. "A Computational Architecture for Coupling Heterogeneous Numerical Models and Computing Coupled Derivatives". In: *ACM Trans. Math. Softw.* 44.4 (June 2018), 37:1–37:39. ISSN: 0098-3500. DOI: 10.1145/3182393.

[39] Joaquim R R A Martins and John T Hwang. "Review and unification of methods for computing derivatives of multidisciplinary computational models". In: *AIAA journal* 51.11 (2013), pp. 2582–2599. DOI: 10.2514/1.J052184.

[40] Joaquim RRA Martins, Juan J Alonso, and James J Reuther. "A coupled-adjoint sensitivity analysis method for high-fidelity aero-structural design". In: *Optimization and Engineering* 6.1 (2005), pp. 33–62.

[41] Hans Petter Langtangen and Anders Logg. *Solving PDEs in python: the FEniCS tutorial I*. Springer Nature, 2016.

[42] Martin Sandve Alnæs. "UFL: a finite element form language". In: *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012, pp. 303–338.

[43] R. C. Kirby and A. Logg. "A Compiler for Variational Forms". In: *ACM Trans. Math. Softw.* 32.3 (Sept. 2006), pp. 417–444. ISSN: 0098-3500.

[44] Martin S Alnæs, Anders Logg, Kristian B Olgaard, Marie E Rognes, and Garth N Wells. "Unified form language: A domain-specific language for weak formulations of partial differential equations". In: *ACM Transactions on Mathematical Software (TOMS)* 40.2 (2014), pp. 1–37.

[45] Melvyn S Berger. *Nonlinearity and functional analysis: lectures on nonlinear problems in mathematical analysis*. Vol. 74. Academic press, 1977.

[46] Christophe Geuzaine and Jean-François Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities". In: *International journal for numerical methods in engineering* 79.11 (2009), pp. 1309–1331.

[47] Liying Liu, Jian Xing, Qingwei Yang, and Yangjun Luo. "Design of large-displacement compliant mechanisms by topology optimization incorporating modified additive hyperelasticity technique". In: *Mathematical Problems in Engineering* 2017 (2017).

[48] Sandilya Kambampati, Justin S Gray, and H Alicia Kim. "Level set topology optimization of structures under stress and temperature constraints". In: *Computers & Structures* 235 (2020), p. 106265.

[49] Mehmet Akgun, Raphael Haftka, K Wu, and Joanne Walsh. "Sensitivity of lumped constraints using the adjoint method". In: *40th Structures, Structural Dynamics, and Materials Conference and Exhibit*. 1999, p. 1314.

[50] Kazuko Fuchi, Taylor H Ware, Philip R Buskohl, Gregory W Reich, Richard A Vaia, Timothy J White, and James J Joo. "Topology optimization for the design of folding liquid crystal elastomer actuators". In: *Soft matter* 11.37 (2015), pp. 7288–7295.