Preprint of article: https://doi.org/10.2514/3.61260

Topology optimization in OpenMDAO

Hayoung Chung · John T. Hwang · Justin S. Gray · H. Alicia Kim

Abstract Recently, topology optimization has drawn interest from both industry and academia as the ideal design method for additive manufacturing. Topology optimization, however, has a high entry barrier as it requires substantial expertise and development effort. The typical numerical methods for topology optimization are tightly coupled with the corresponding computational mechanics method such as a finite element method and the algorithms are intrusive, requiring an extensive understanding. This paper presents a modular paradigm for topology optimization using OpenMDAO, an open-source computational framework for multidisciplinary design optimization. This provides more accessible topology optimization algorithms that can be non-intrusively modified and easily understood, making them suitable as educational and research tools. This also opens up further opportunities to explore topology optimization for multidisciplinary design problems. Two widely used topology optimization methods—the density-based and level-set methods—are formulated in this modular paradigm. It is demonstrated that the modular paradigm enhances the flexibility of the architecture, which is essential for extensibility.

1 Introduction

Topology optimization is a numerical method that computes an optimal structural layout for any given set of objective and constraints. There are two primary formulations used to perform topology optimization. One is the material-

H. Chung
University of California San Diego E-mail: hac210@ucsd.edu

J. T. Hwang
University of California San Diego

J. Gray
NASA Glenn Research Center

H. A. Kim
University of California San Diego; Cardiff University E-mail: alicia@ucsd.edu

distribution formulation where the material/void distribution in the design domain is optimized and the density of each finite element is the design variable. The most common method in this category is Solid Isotropic Materials with Penalization (SIMP) [7]. The other category is the boundary-based formulation where the structure is defined by a function that is iteratively moved to determine the optimal layout. The most common method here is level-set topology optimization which employs an implicit function to represent the boundaries and the level-set advection problem is solved to move the boundaries.

Regardless of their parameterization, the two topology optimization methods share many common aspects as they have a strong tie to the structural analysis and are inherently large-scale optimization. Both categories of topology optimziation require the solution of a boundary value problem to compute the structural response of any given design. Then the derivatives of the structural response with respect to the design variables — the values that control the topology of the structure — are needed to leverage efficient gradient based optimization algorithms and keep the computational cost of topology optimization reasonable.

To reduce the entry barrier of building up these components, there are numerous research and literature published with an educational purpose. For example, the seminal paper by Sigmund (2001) [24] explains the basic set of algorithms for SIMP topology optimization and provides a 99-line MATLAB code that embodies the method. Several variants of the code have been published since: a simpler code structure with enhanced program efficiency [3] and extended capability [1, 21]. There are also works using the different domain representations, including the level-set method [19] and regularization [22]. Additionally, the efforts are made to leverage the benefits of open-source software development [4] in solving the boundary value problems. By providing a wrapper to the third-party applications, the user base of topology optimization methods is expected to be widened. Having a primary goal of communicating a working knowledge of the topology optimization schemes to their readers, these codes have been successful in providing the essential numerical algorithms as a self-sufficient program. However, as a consequence, the functionalities are limited. Also, these programs present procedural architectures, where the numerical algorithms are separated into several subroutines and the analysis and the optimization layers are tightly coupled.

Such an architecture generally limits the reusability of these programs when their extension or modification is desired. First, these subroutines are not independent as they are specifically designed to be used within the designated context. Furthermore, any modification to the code such as changing objective function or adding a discipline requires an intrusive intervention into the interconnected modules. This requires frequent re-programming as well as a thorough knowledge of the entirety of the existing code structures. These limitations are largely derived from the tight coupling between the analysis and optimization. In addition, such coupling also prevents topology optimization to be considered as one module in the context of multidisciplinary design optimization (MDO) and presents challenges integrating into a system level design

process. In order to make topology optimization more accessible, the domains for optimization and analysis are treated as separate modules within the MDO framework. For this goal, numerical algorithms involved in the methods are programmed as separate objects that strictly comply not only with the OOP (object-oriented programming) paradigm, but also the conventions of MDO. These require a careful design of the classes including the encapsulation of the variables and method specialized for optimization.

This paper presents both SIMP optimization [7] and level-set topology optimization (LSTO) [2, 8] implemented within a general MDO context, using NASA's OpenMDAO framework. Our selection of OpenMDAO as the platform is motivated by the easy implementation of the program and its convenient extension—the ease of reusing previously developed OpenMDAO code. The benefits are primarily originated from two aspects of OpenMDAO. The first aspect is a the ability to develop topology optimization as a hierarchically modularized program built from a set of small and reusable parts. Modularity and reusability, while improving code reuse and flexibility, introduce a new challenge when applied to topology optimization because of the need to use gradient based optimization with analytic derivatives. With a code designed to have parts being added, removed, and re-ordered on a regular basis, then it becomes a tedious and expensive task to re-implement the derivative calculations with every change of the model structure. The second aspect of the OpenMDAO framework that motivates its use in this effort directly addresses this challenge related to derivative computation. OpenMDAO can automatically compute total derivatives of a model composed of multiple parts, assuming that each part provides its own partial derivatives. This means that re-ordering of existing models requires no additional work with regard to computing derivatives needed for optimization. Adding new functionality to the model requires only differentation of the new calculations, but relieves you of the need to integrated those partial derivative calculations into the overall model. The modularity and automatic computation of total derivatives enable a truly reusable and reconfigurable code design, as the need for an intrusive inspection into the internal structure of computational units is eliminated.

2 Background

2.1 Material-distribution method: SIMP

SIMP explicitly describes topology as a distribution of the local material represented by densities ρ. The local material property is assumed to be proportional to the local density, hence it is sometimes classified as a density-based method. In this method, the design variables are discretized according to the elements found in the finite element mesh. The local density ρ_i is assigned to the ith element and assumed to be a continuous variable between 0 (void) to 1 (fully filled with the material). With the piecewise continuity and the bound, the optimization problem is well-suited to employ gradient-based op-

timization. However, the continuous density leads to an intermediate density that can make an identification of the resulting topology challenging. The intermediate densities are therefore typically penalized to promote a binary distribution. In addition, it is also customary to apply a filtering on sensitivity or density to obtain a numerically stable solution. The physical analysis of a structure employs the finite element method. For further details, the readers are referred to [7].

A workflow of SIMP is illustrated in Fig. 1 by following the convention of the extended design structure matrix (XDSM) diagram. Further details on the XDSM convention can be found in [20]. First the initial design variable vector $[\rho^0]$ filtered to compute $[\bar{\rho}]$ and later penalized by p. A state variable vector of displacements ($[u]$) is calculated based on the finite element analysis, where a distribution of the discrete densities $[\bar{\rho}^p]$ is employed to construct the global stiffness of the structure. The objective (f), constraints (g), and their derivatives are computed with respect to the state and design variables. Typically there are many more variables than objective and constrain values so an adjoint method is used to efficiently calculate the total derivatives for the optimizer.

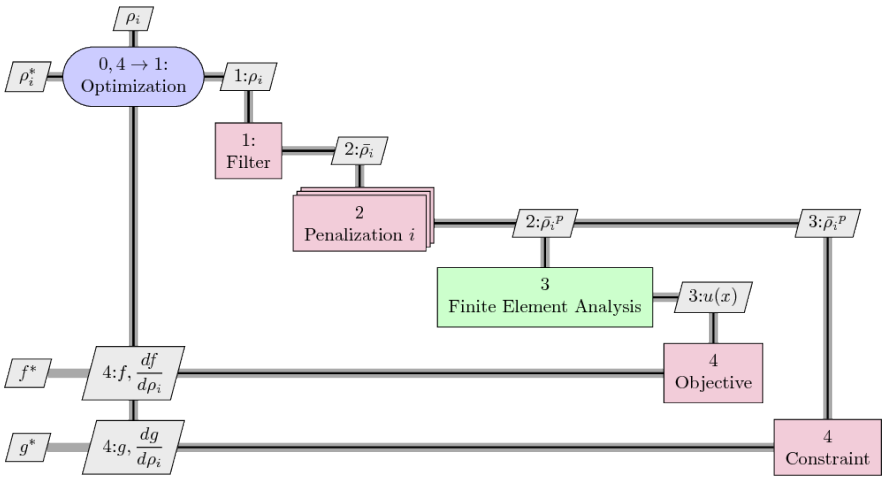

Fig. 1 XDSM diagram of the SIMP method

2.2 Level-set based method: LSTO

LSTO employs an implicit representation of the structural boundaries that characterizes the topological layout Ω. The design is updated by moving the boundaries instead of determining its distribution of the local materials. A primary advantage of LSTO comes from the smooth boundaries that are always

well defined in contrast to the pixelated boundaries and intermediate density elements found in the SIMP solutions. The solution obtained by LSTO is thus easily interpreted to the structure. The boundary of the layout is defined as a zero level-set function $\phi(x) = 0$, and it is modified or moved by solving the Hamilton-Jacobi equation

$$\dot{\phi} + \nabla\phi \cdot \frac{\partial x}{\partial t} = \dot{\phi} + V_n|\nabla\phi| = 0 \tag{1}$$

where t is the pseudo timestep of the advection problem, and V_n is the advection velocity normal to the boundary, which is a key variable to get an optimal solution. An explicit integration scheme is often employed to solve the problem for such discretized space and time domains. This paper employs the LSTO method where V_n at the k-th iteration is determined by solving a linearized suboptimization problem. This is briefly discussed below, and the interested readers are referred to the work by Sivapuram et al [25] for further details.

The change in the objective f and constraint functions g can be written as,

$$\Delta\{f, g\} = \Big\{\frac{\partial f}{\partial \Omega^k}, \frac{\partial f}{\partial \Omega^k}\Big\}\Delta\Omega^k = \big(\Delta t V_n^k\big)\sum_{j}^{B}\{S_{f,j}^k, S_{g,j}^k\}l_j \tag{2}$$

after linearization with respect to current domain Ω^k, and discretization by a set of boundary points (B). $S_{f,j}$ and $S_{g,j}$ represent the shape sensitivities for the objective and constraint, evaluated at the boundary point j. The sensitivities are calculated based on the state variables obtained by finite element analysis. The structure domain Ω^k is projected onto the finite element mesh by Ersatz method, which replaces complex remeshing. The weighted least square method is employed to reconstruct the shape sensitivity at the boundaries [11]. The sensitivities are numerically integrated by multiplying length segment l_j.

Without loss of generality, an advecting velocity V_n is set to αd, where α is a distance that boundary travels along the unit direction d. According to the Ref. [25], (2) is further reduced to the following suboptimization formulation.

$$\begin{aligned} \min\ & \Delta f = \Delta t C_f^k \cdot \alpha^k d^k \\ \text{subject to } & \Delta g = \Delta t C_g^k \cdot \alpha^k d^k < -g^k \\ \mathbf{d}^k = & \frac{C_f^k + \lambda^k C_g}{||C_f^k + \lambda^k C_g||}, \quad \mathbf{z}_{n,min}^k \leq \alpha^k \mathbf{d}^k \leq \mathbf{z}_{n,max}^k \end{aligned} \tag{3}$$

where $C_f = S_f \times l$ and $C_g = S_g \times l$ are numerically integrated sensitivities. λ is the vector of Lagrange multipliers for inequality constraints equation. The bounds on the traveling distance $(z_{n,min}^k, z_{n,max}^k)$ are calculated based on the two criteria: (i) the advected layout must not go outside the design domain, and (ii) the distance cannot violate the CFL (Courant-Frederichs-Lewy) condition. Solving the suboptimization problem (3) gives the optimum values for Δt, α^k, d^k, which are then used to compute the optimum velocity V_n in (1).

The workflow of LSTO is illustrated in Fig. 2. There are two layers of algorithms that corresponds to each optimization. Fig. 2(a) shows an update of

the level-set function ϕ by solving the Hamilton-Jacobi equation. Many of the routines, marked by either pink or green boxes, are also found in the SIMP case. However, there is no filtering or penalization methods that follows after discretization. Suboptimization 3 is shown in Fig 2(b). The scaling operation that normalizes the sensitivities to be bound to the limit [-1, 1] is also applied. This auxiliary operation is essential in making the suboptimization independent to the physical unit, hence decoupling the finite element analysis from the level-set layer.

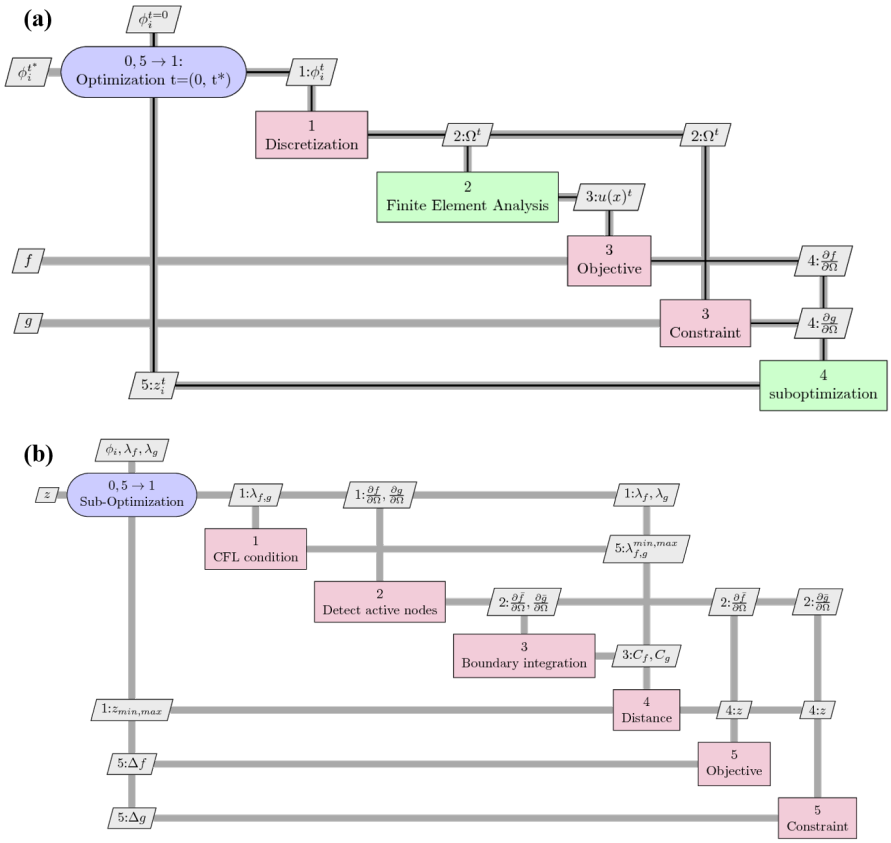

Fig. 2 An XDSM diagram of LSTO (a) General outlook of LSTO (b) Suboptimization

2.3 OpenMDAO

OpenMDAO is an MDO platform developed and maintained by NASA [13, 14] (`http://openmdao.org`). Being capable of handling a large number of variables and disciplines, it has been used to solve MDO problems in a wide

range of engineering fields, e.g. satellite design [16], wind turbine design [13], aircraft design [17] and aircraft trajectory optimization [12].

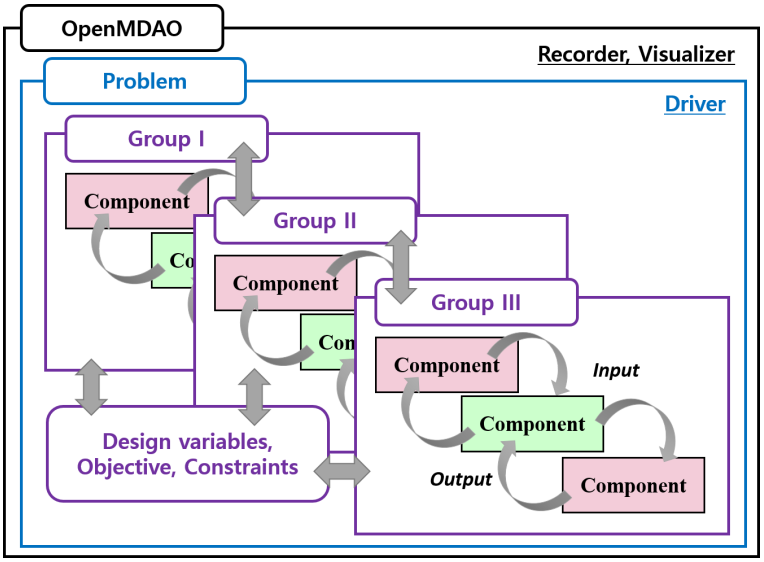

Fig. 3 Illustration of modularized architecture found in OpenMDAO. Three set of the layers are found at the different hierarchies: Component, Group, and Problem.

OpenMDAO has a hierarchical OOP architecture written in Python, oriented for both system construction and optimization, as shown in Fig. 3. An OpenMDAO model is constructed by a combination of Component classes, which are located at the lowest level of the class hierarchy. The governing equations of the physics are embodied into the Components. The Component class is designed for gradient-based optimization as it has methods to calculate not only the output for the given inputs but also the partial derivatives between the arguments. Such a self-containing ferature enhances the reusability of an object. The contents of the Components is determined by the degree of modularization; the Component can be as comprehensive as a single discipline (e.g., finite element analysis), or as small as a subset of the equations (e.g., the penalization equation found in the SIMP-based optimization). Since increasing the number of communicating variables leads to inefficiency, one needs to consider the possible extensions of the system and design variables in choosing a degree of decomposition. In this work, computation illustrated as a rectangular box in a XDSM diagram is directly implemented as a Component. The Component objects are contained within the Group class layer. The connectivity between the Components is defined within a Group as shown in the Appendix. It reflects the arrangement between the equations that is illustrated as a line in the XDSM diagram. For every OpenMDAO program, a Problem object is

defined as the highest level class. The Problem class contains a system, and a Driver object that executes the optimization.

During the assembling of a system, the total derivatives are computed based on the partial derivatives of the corresponding Components. The connectivity between the Components is reflected during the sensitivity analysis. OpenMDAO assists the analysis as it is designed around a collection of equations and algorithms, called the MAUD architecture [18]. MAUD provides a generalized method that unifies the derivative computation methods using a matrix equation. This is useful when models contain implicit equations because the user is no longer required to manually implement the adjoint method.

With OpenMDAO, the need for reprogramming and the likelihood of errors are reduced. For example, programming for partial derivatives is simplified as the smaller components have fewer inputs and outputs, hence there is less to differentiate. Furthermore, an intrusive inspection into an object is not necessary for the analytic calculation of total derivatives as this computation is abstracted and automatically handled by the framework based on the partial derivatives. It enhances the flexibility of the existing objects as the programmed equations can be reused or reconfigured with ease.

3 Topology Optimization Algorithm in OpenMDAO

3.1 Optimization problem

Compliance minimization is a classic problem in topology optimization as shown below [7].

$$\begin{aligned} &\min f(x) = u(x)^T F \\ &\text{subject to } g(x) - g^* \leq 0 \end{aligned} \tag{4}$$

where Ω is the structural domain, f is the compliance of the structure, which is determined by design variable x, displacement function $u(x)$, and mechanical load F. A volume constraint, $g(x)$ is usually imposed to be less than or equal to g^*. Note that u is also a function of x.

3.2 SIMP Algorithm

In OpenMDAO, SIMP has six Component objects, which include five subroutines described in the XDSM diagram (Fig. 1) and one Component that defines the design variables. They are aggregated into one Group object to create the SIMP method. In this section, each of the Components classes is briefly explained in terms of its input and output arguments. The governing equation and its partial derivatives are also described.

The Components and their connectivities are visualized using the design structure matrix shown in Fig. 4, natively generated by OpenMDAO. Each Component is marked by a blue box, and its inputs (pink) and outputs (either

gray or orange) exposed to the Group layer are also specified. An orange output is an unknown variable from the implicit function, while a gray variable is an explicit output. The connectivities between variables are marked by black squares. The connectivities found in the design structure matrix is diagonal-dominant, as there is only one discipline and the coupling between Components are forward-dominant.

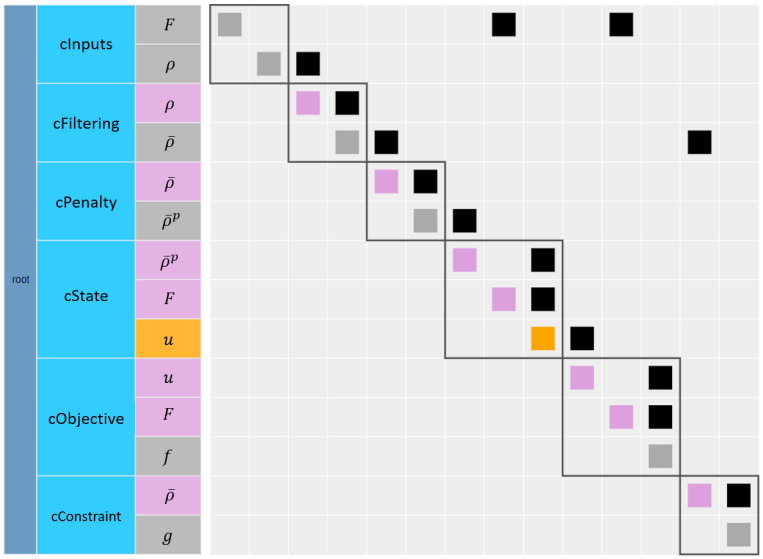

Fig. 4 Design structure matrix of SIMP method.

3.2.1 Global parameters

Scalar parameters that are neither input or output argument of the equation are declared as global parameters. These parameters are not explicitly shown in the XDSM, but all the Component objects have an access to the global parameter. There are two global parameters in SIMP implementation. A penalization parameter p is set to be 3, and a filtering radius R is set to be the double of the length of the elements.

3.2.2 Independent Variables (cInputs)

In SIMP, discretized piecewise material densities are defined as a design variable. Although not shown in the XDSM diagram (Fig. 1) as a design variable, the mechanical force F is also defined herein because it is an input argument of the finite element analysis object that solves a boundary value problem.

- Inputs: None
- Output: ρ_i, F

3.2.3 Filtering (cFiltering)

A linear density filter is used herein.

$$\bar{\rho}_i = \sum_j w_{ij} \rho_j, \tag{5}$$

where the $\bar{\rho}_i$ refers the filtered density at element i. w_{ij} indicates the weighting factor that are calculated by simple conic weight equation [26], which averages the density values found i and the neighboring elements j.

- Inputs: ρ_j
- Output: $\bar{\rho}_i$
- Partial derivatives: w_{ij} only when $i = j$; otherwise 0

3.2.4 Penalization (cPenalty)

- Inputs: $\bar{\rho}_i$
- Output: ${\bar{\rho}_i}^p$
- Partial derivatives: $p{\bar{\rho}_i}^{p-1}$

3.2.5 Finite Element Analysis (cState)

In the finite element analysis object, an implicit equation

$$R({\bar{\rho}_i}^p, u, F) = K({\bar{\rho}_i}^p)u - F = 0 \tag{6}$$

is solved where $K({\bar{\rho}_i}^p)$ refers a global stiffness matrix.

- Inputs: ${\bar{\rho}_i}^p$, F
- Output: state variable u
- Partial derivatives: $\frac{\partial R}{\partial {\bar{\rho}_i}^p}$, $\frac{\partial R}{\partial u}$, $\frac{\partial R}{\partial F}$

3.2.6 Objective (cObjective)

- Inputs: u, F
- Output: $f = u^T F$
- Partial derivatives: $\frac{\partial f}{\partial u}$, $\frac{\partial f}{\partial F}$

3.2.7 Constraints (cConstraint)

- Inputs: ${\bar{\rho}_i}^p$
- Output: $g = \sum_i {\bar{\rho}_i}^p$

3.3 LSTO Algorithm

As described in the XDSM diagram (Fig. 2), LSTO is composed of two sets of computation layers: (i) an update of the level-set function by solving a Hamilton-Jacobi equation on the level-set grid, and (ii) suboptimization, where the optimal V_n is calculated at each iteration. In this work, the suboptimization routine is implemented as the OpenMDAO Problem object, while the rest of the updating operations are implemented to wrap around the object for computational efficiency. Given that optimal advecting distance z is obtained for the given instance, their values are firstly extended to the level-set grid using the 5th order WENO scheme. The signed distance function is then updated using the fast marching method. The finite element discretization of the current design domain employs an Ersatz model that cuts the material using a marching square method. The structure of the level-set numerical libraries are shown in the Appendix (6), and detailed descriptions of the numerical procedures and their algorithms can be found in Ref. [10, 15].

The suboptimization scheme is implemented as eight Components. Since all the equations are explicit, their partial derivatives are not shown herein for brevity.

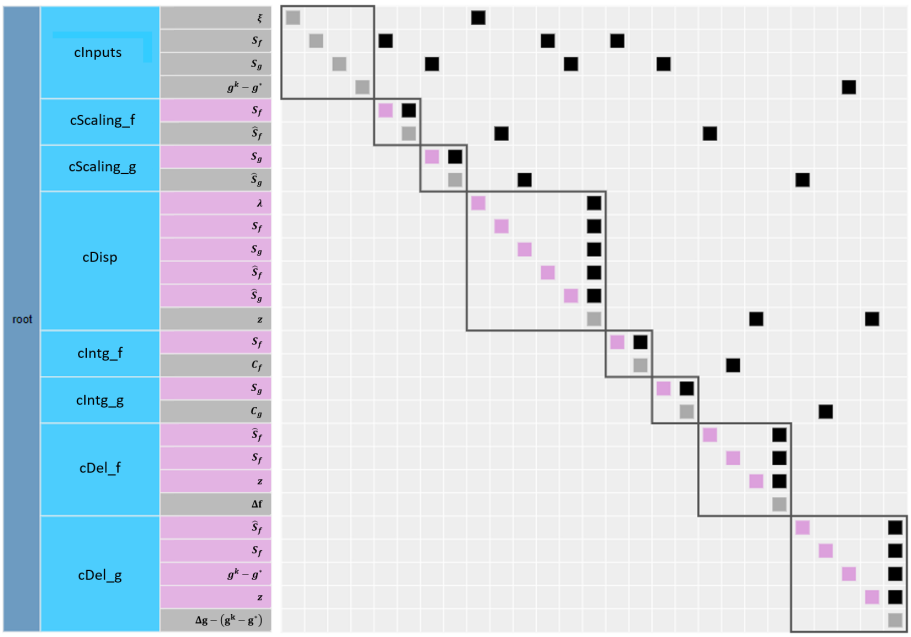

Fig. 5 Design structure matrix of the suboptimization found in LSTO

3.3.1 Global parameters

The descritized properties that describe the structure are declared as the global parameters. The length segment l is defined at each boundary node. The bound of the Lagrange multiplier $\xi_{min,max}$ is computed by considering the limit of the boundary movement of the nodes.

3.3.2 Independent Variables (cInputs)

The ξ vectors contains the Lagrange multipliers and pseudo-time, found in (3). The sensitivities S_f and S_g are given as the independent variables. g and g^*, which refer the constraint value and the corrent step and the problem constraint are specified, since their difference $g - g^*$ constrains the z to enforce the solution to be within inactive constraint region.

- Inputs: None
- Outputs: ξ, S_f, S_g, $g - g^*$

3.3.3 Scale parameters (cScaling_f, cScaling_g)

These two Components are based on the same subroutine that normalizes the sensitivity to be bounded to [-1, 1]. Each compute the scaling parameter for objective (*cScaling_f* or constraints (*cScaling_g*) .

$$\hat{S_f} = 1/max(||S_f||), \quad \hat{S_g} = 1/max(||S_g||) \tag{7}$$

The normalization is a essential in calculating z, as it should not be biased by the magnitude of the sensitivites, which is affected by a definition of the objective and constraint functions.

- Input: S_a (a can be either f or g)
- Output: $\hat{S_a}$

3.3.4 Displacement (cDisp)

The displacement z is calculated based on (3). The scaling parameters are included alongside with the sensitivities.

- Inputs: S_f, S_g, $\hat{S_f}$, $\hat{S_g}$, ξ
- Output: z

3.3.5 Boundary integration (cIntg_f, cIntg_g)

- Inputs: S_a (a can be either f or g)
- Output: C_a

3.3.6 Objective (cDel_f)

– Inputs: S_f, z, $\hat{S_f}$
– Output: $\Delta f = S_f \times z \times \hat{S_f}$

3.3.7 Constraints (cDel_g)

– Inputs: S_g, z, $\hat{S_g}$, $g^k - g^*$
– Output: $\Delta g - (g^k - g^*)$

4 Numerical results and Discussions

4.1 Problem definition

We demonstrate SIMP and LSTO algorithms in OpenMDAO using a familiar numerical example. A compliance of a rectangular cantilevered beam of Fig. 6 is minimized while the total material is subject to be less than or equal to 40% of the area of total design domain.

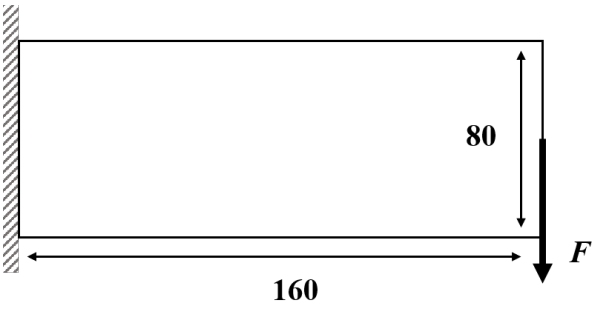

Fig. 6 The cantilevered plate for the compliance minimization problem.

Young's modulus E and Poisson ratio ν are set to 1.0 and 0.3, respectively. A vertically downward force F is applied with a value of 1 at the mid-point of the right edge. A linear elasticity is assumed. Sequential Quadratic Programming (SLSQP) from SciPy optimization library is used, with a convergence criteria ($\Delta f < 10^{-5}$).

4.2 SIMP

Initial material densities are assumed to be uniform throughout the design domain, while their sum is set to 0.861 of the total volume to match with the LSTO example. The finite element domain and the equivalent design space is

discretized with 80×40 quadrilateral elements. Figure 7(a) shows the convergence graph of the compliance $f(\rho)$ and the total material density $g(\rho)$. The compliance profile fluctuates in the first 10 iterations while the constraint is satisfied within 2 iterations. After the oscillation, compliance keeps decreasing until it converges as shown in the Fig. 7(a). The optimal layout is obtained at iteration 120. The intermediate densities, and the disconnected materials that are observed in the early iterations are removed. The jagged boundary is found in the optimal layout, which is a characteristic of the material-distribution method.

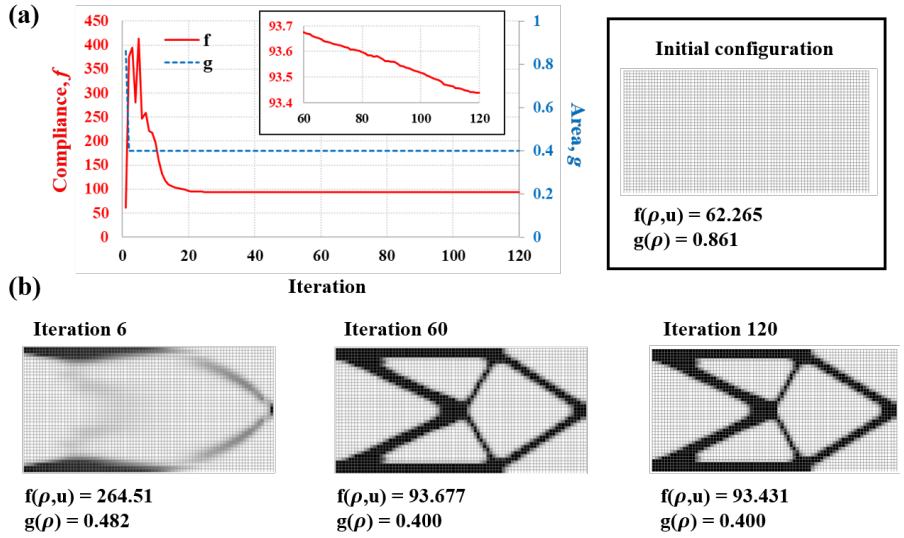

Fig. 7 Numerical results of SIMP. (a) a convergence profile and initial configuration (b) topological solutions, compliance (f), and area fraction (g) at the 6^{th}, 60^{th}, and 120^{th} iterations.

4.3 LSTO

The same compliance minimization problem is solved with the LSTO method. The initial design domain has seeded holes, instead of utilizing the hole-creation algorithms [9]. Both level-set grid and finite element mesh for structural domain are set to 80×40. The CFL condition of 0.5 is used. Figure 8(a) shows the convergence. In contrast to the SIMP example, both the compliance and constraint values exhibit a smooth convergence. The convergence is achieved at iteration 121, which is equivalent to the SIMP example. The changes of the topological layouts are illustrated in the Fig. 8(b). The optimal solution agrees well with the established optimum solution. An optimal topology is found to be similar to the that from SIMP method, and the overall

solutions have a similar quality. Although it is true that the resulting compliance is lower in the LSTO case than in the SIMP case, it does not necessarily ensure the relative optimality of the former layout. These optimization schemes have different design space hence the values are not directly comparable. The lower value of the LSTO case is partially accounted by a removal of a jagged boundary, and an allowed partial elements, but the direct comparison between these results are not the aim of the present work.

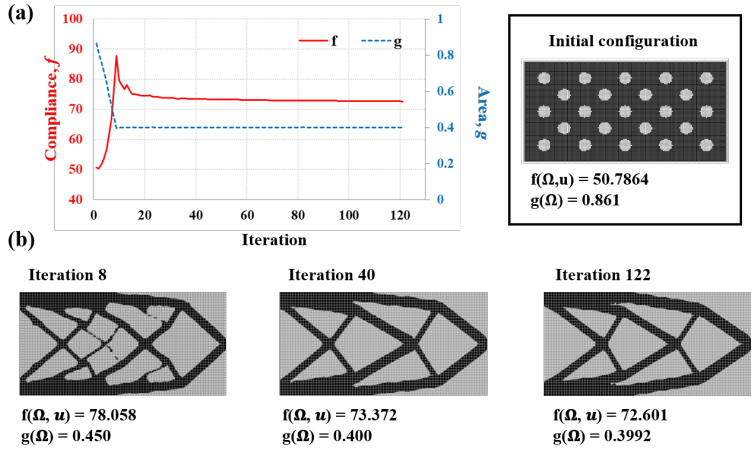

Fig. 8 Numerical results of LSTO. (a) a convergence profile and initial configuration (b) topological solutions, compliance (f), and area fraction (g) at the 8^{th}, 40^{th}, and 122^{th} iterations.

5 Demonstration of Reusability and Reconfigurability

This section demonstrates the benefits of the architecture's reusability and reconfigurability for education and research via two examples. The first example shows the enhanced reusability of the Component objects. The second example shows the easy reconfigurability, where the sequence of the Component objects within the Group is randomly arranged.

5.1 Example 1: Reusability

The reusability of the modules is exemplified by reusing the existing Components for a new topology optimization formulation that bears similarities to the parameterized level-set topology optimization approach [6,8,23]. This method is, in essence, a density-based approach, but the densities are parametrized using a level-set function that implicitly represents the boundary. Instead of tracking the boundary directly, the value of this function is used to control

the densities in the elements, using a Heaviside function to map to the $(0, 1)$ interval and SIMP-type penalization to discourage intermediate densities.

The Components of finite element analysis (*cState*), penalization (*cPenalty*), objective (*cObjective*) and constraints (*cConstraint*) are reused. Only the methods for the parameterizing level-set function ϕ are newly implemented. The design structure matrix of this optimization method is shown in Fig. 9.

5.1.1 Independent variables (cInputs)

The design variables are the values of the regularly spaced points p. The nodal force F is also declared as an independent variable as done in SIMP-based optimization.

- Inputs: None
- Output: p, F

5.1.2 Parameterization (cParam)

The continuous hypersurface ϕ is computed using a bivariate B-spline interpolation of the given control points p, which are the design variables. The order of the B-spline is set to 3, and 9 Gauss quadrature points are selected to evaluate the function. The function is evaluated at the Gauss quadrature points g.

- Inputs: p
- Output: ϕ_g

5.1.3 Heaviside filtering (cHeaviside)

A smooth approximate $H(x)$ to the Heaviside function is employed to map the large values found in the Gauss quadrature points to the $(0, 1)$ interval. The hyperbolic tangent function is used as an analytical Heaviside function.

$$\rho = H(\phi) = \frac{1}{2}(1 + tanh(\phi)) \tag{8}$$

- Inputs: ϕ_g
- Output: ρ_g

5.1.4 Averaging (cAverage)

The filtered values found in the ith element are averaged to generate piecewise constant densities ρ_i, which are interpreted as SIMP-like material densities.

- Inputs: ρ_g
- Output: ρ_i

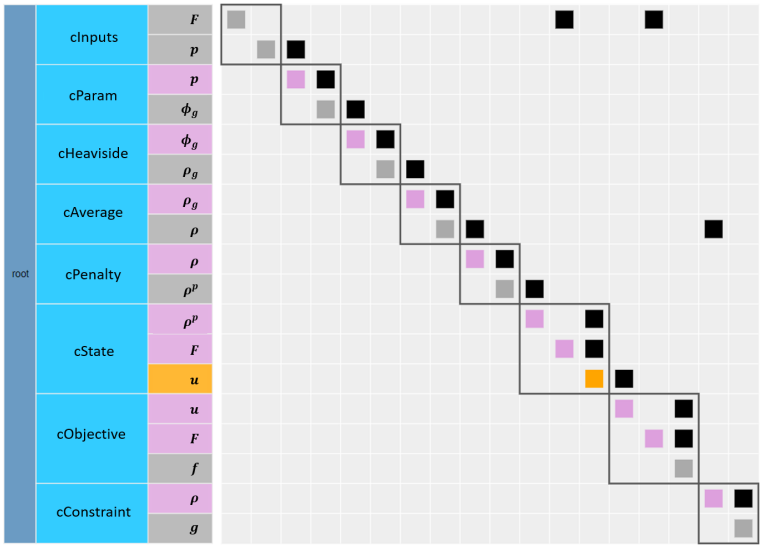

Fig. 9 Design structure matrix of the parametric level-set approach for topology optimization.

The convergence graphs are presented in Fig. 10(a). Since the method is essentially a density-based method like SIMP, oscillations are observed in the earlier steps during optimization. The material layout and compliance value found in Fig. 10(b) are comparable to those from the SIMP-based optimization. There are several oscillations in the optimized layouts that are presumably numerical artifacts due to the coarseness of the level-set function parameterization [23]. However, a detailed discussion on the optimality of the results is out of the scope of this work, as our goal here is to highlight the reusability as a benefit from using a modular computational approach.

5.2 Example 2: Reconfigurability

This example demonstrates the ease of reconfigurability. We investigate the density filtering effect by filtering the penalized densities instead of the unpenalized densities. This can be achieved simply by switching the order of the *cFiltering* and *cPenalty* Components found in Fig. 4. The new design structure matrix is shown in Fig. 11.

In practice, this can be achieved by changing three lines of code, as shown in Fig. 12. This contrasts with the previous code structures [21, 24, 27], where making such a change requires significant re-programming effort for the sensitivity analysis method and also intrusive inspection of the existing density filter.

Although the whole code is not shown herein, one may easily see how the removal of the filter can be realized as well. The numerical results of these modifications are shown in Fig. 13. All the numerical parameters are

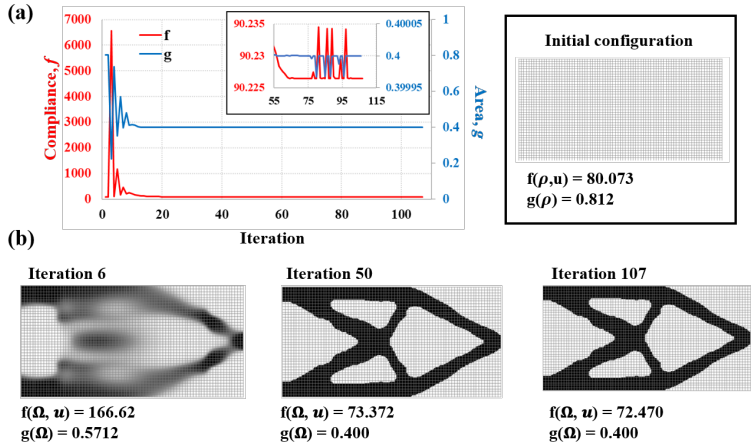

Fig. 10 Numerical results of SIMP. (a) a convergence profile and initial configuration (b) topological solutions, compliance (f), and area fraction (g) at the 6^{th}, 50^{th}, and 107^{th} iterations.

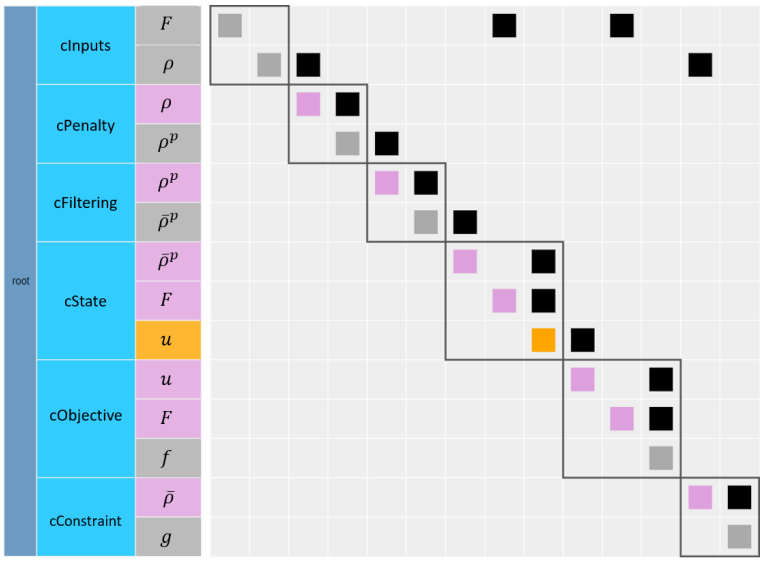

Fig. 11 The design structure matrix of the SIMP method, where the filter object is rearranged with respect to that of the penalization step

equivalent to the SIMP example shown in Sec. 4.2. As one may expect, either rearrangement (Fig. 13(a)) or removal (Fig. 13(b)) of the existing filter each saliently generates an intermediate density or a checkerboard pattern in the converged solutions.

Although the presented system configurations are not typical in the topology optimization method, these exhibit the effect of the filtering and its ar-

(a)

```
self.connect('cInputs.dvs', 'cFiltering.dvs')

# filter
comp = DensityFilterComp(…)
self.add_subsystem('cFiltering', comp)
self.connect('cFiltering.dvs_bar', 'cPenalty.x')

# penalization
comp = PenalizationComp()
self.add_subsystem('cPenalty', comp)
self.connect('cPenalty.y', 'cState.rho')
```

(b)

```
self.connect('cInputs.dvs', 'cPenalty.x')

# filter
comp = DensityFilterComp(…)
self.add_subsystem('cFiltering', comp)
self.connect('cFiltering.dvs_bar', 'cState.rho')

# penalization
comp = PenalizationComp()
self.add_subsystem('cPenalty', comp)
self.connect('cPenalty.y', ''cFiltering.dvs')
```

Fig. 12 a code snippet for the reconfiguration that exemplifies the easy reconfiguration: (a) a declaration of the connectivities between Components found in original SIMP Group (b) corresponding code where the *cPenalty* and *cFiltering* are switched.

rangement to the resulting layouts. It also illustrates the educational benefits of using OpenMDAO for topology optimization, since the ill-posedness of certain formulations can be tested and demonstrated with minimal programming effort.

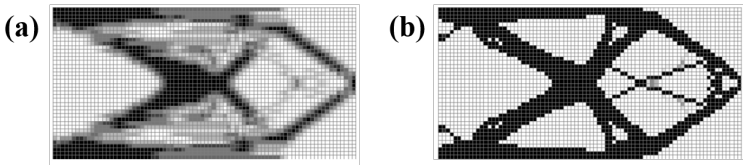

Fig. 13 Optimal solutions where the filter object is (a) rearranged or (b) removed

6 Conclusion

This paper demonstrates the feasibility of formulating topology optimization as an MDO problem, using the OpenMDAO software framework. OpenMDAO offers a distinctive modular architecture with a hierarchical class design, and it is designed specifically for gradient-based optimization. The model used for topology optimization is decomposed into a set of modular Component objects that each define equations and their partial derivatives. These Components are assembled in a Group object where the total derivatives are calculated semi-automatically. Using this modular approach, the programming effort is reduced and the reusability of the resulting components is increased. Moreover, the MAUD architecture enables non-intrusive programming thus making the existing objects truly reusable and reconfigurable.

Both the material-distribution (SIMP) and boundary-based (LSTO) topology optimization methods are decomposed into a set of subroutines, and implemented in a modular way. A classical compliance minimization problem is solved, and similar topological designs are obtained. The benefits from the architecture, the reusability, and the reconfigurability are also demonstrated through two examples. The first example demonstrates the ease of reusing the

pre-existing Component objects. Several Components found in SIMP are easily reused to implement a different optimization method, where the level set function is constructed using B-spline interpolation and it parameterizes the material density. Such an example implies that the objects can be reused in a different context hence a user can easily implement the new idea or algorithms based on the existing methods. Additionally, the flexible rearrangement of the objects is shown in the second example. The effect of the density filtering to the optimization is easily investigated by only changing a few lines of code. In both examples, the programming effort involved and required understanding are both minimized.

Although the modular approach presented here offers significant advantages there are some potential counterpoints worth discussing. The primary reason for the fine-grained decomposition used here was to rely on OpenMDAO's built-in capability to reduce the overall implementation complexity and effort for computing analytic derivatives for the optimizer. However, if gradient-free optimization methods were employed there would obviously be no need for the analytic derivatives which would reduce the motivation for the modularity here, and a more tightly integrated code might be preferred. Additionally, it should be noted that at the time of writing this OpenMDAO does not support analytic hessian calculations, which some practitioners are taking advantage of to achieve more efficient gradient-based optimizations. If Hessians are required, then at the current time OpenMDAO cannot support the needed computations.

Despite these potential minor limitations, the advantages of this modular approach remain significant because of the dramatic improvement in code flexibility and re-use as well as the significant reduction in implementation effort for computing analytic derivatives.

A. Appendix

A.1 Numerical Libraries

In this work, the existing finite element and level-set libraries written by C++ are interfaced to OpenMDAO by Cython [5] for an efficient computation. OpenLSTO (open-sourced level-set topology optimization) is a recently-published level-set topology optimization suite and in active development by researchers at the University of Cardiff and the University of California San Diego, which is freely available (`https://github.com/M2DOLab/OpenLSTO-lite`). The program is designed to provide users with easy access to the level-set method. To make a compilation process straightforward and demonstrate its usage within OpenMDAO, we provide the aforementioned OpenMDAO classes and OpenLSTO as a single suite (`https://github.com/chungh6y/openmdao_TopOpt/tree/master/OpenLSTO-master`). Detailed installation process is provided therein. After compilation, the user may find two libraries (the finite element and the level-set) as a shared library, which can be imported as the

Python object. The concise examples of the given libraries along with the comments are found below.

A.1.1 Finite Element Method

As specified in the Fig. 6, dimensions (*lx*, *ly*), number of elements (*nelx*, *nely*), and force conditions are specified. Detailed descriptions of functions that specify boundary conditions can also be found in the GitHub repository. To be used within OpenMDAO, the calculation routines for the stiffness matrices and their derivatives give out the entities of sparse matrix, which is built by SciPy.sparse library.

```
from pyBind import py_FEA

# Meshing ==============================
fem_solver = py_FEA(lx = 160, ly = 80,
                    nelx=80, nely=40, element_order=2)
[node, elem, elem_dof] = fem_solver.get_mesh()

# Material =============================
fem_solver.set_material(E=1., nu=.3)

# Boundary condition  ==================
coord = np.array([0,0])
tol = np.array([1e-3,1e10])
fem_solver.set_boundary(coord = coord,tol = tol)
BCid = fem_solver.get_boundary()

# External force as specified at Fig. 2
coord = np.array([length_x,length_y/2])
tol = np.array([1,1])
GF_ = fem_solver.set_force(coord = coord, tol = tol,
                           direction = 1, f = -1.0)

# Stiffness matrix given as the entities of sparse matrix (CSC)
nELEM = elem.shape[0]
(rows, cols, vals) = fem_solver.compute_K_SIMP(np.ones(nELEM))
```

A.1.2 Level-set Method

In the present section, a brief usage of the level-set library is illustrated. The method is classified into three categories: (1) geometric property extraction (2) boundary sensitivity evaluation, (3) and the update of ϕ after suboptimization. After the signed distance function is set (*set_levelset*), based on the current domain, Ersatz model *area fraction* as well as the geometric properties are evaluated using the *discretise* function. The boundary sensitivities *bptSens* are then obtained from the sensitivity module, where the compliance sensitivities are computed at the boundary points by various subroutines including least square method. Note that these values are the ingredients of suboptimization, wherein advection velocities at the boundary points (*Bpt_Vel*, *timestep*) are computed. The level-set function is then updated and reinitialized accordingly.

```
from py_lsmBind import py_LSM
from from pyBind import py_Sensitivity

# LSM initialization (with initial holes) ================
lsm_solver = py_LSM(nelx = nelx, nely = nely,
                    moveLimit = movelimit)
hole = np.array([[16, 14, 5], ... , [144, 66, 5]])
lsm_solver.add_holes(locx = list(hole[:,0]),
                     locy = list(hole[:,1]),
                     radius = list(hole[:,2]))
lsm_solver.set_levelset()
```

```
# Discretization ========================================
(bpts_xy, areafraction, seglength) = lsm_solver.discretise()
(lb2,ub2) = lsm_solver.get_Lambda_Limits()

... FEA that gives out structural response u ...

# Sensitivity module  ===================================
pySens = py_Sensitivity(fem_solver, u)
bptSens = pySens.compute_boundary_sens(bpts_xy)

... suboptimization through the OpenMDAO ...

# Advection of the sign distance function ================
lsm_solver.advect(Bpt_Vel, timestep)
lsm_solver.reinitialise()
```

A.1.3 Configuration of the connectivity

For an illustration of the connectivities established between variables of different Components, we present here the code snippet of the Group object of SIMP topology optimization method.

As the Group component initializes before components are called, a number of shared member variables in relation to the finite element meshes (e.g., nNODE, nELEM) as well as optimization parameters (e.g., penalization order p) are defined using *metadata*.

The numerous Component objects shown in Fig. 4 are added via *add_subsystem*, and their variables are connected by *connect* functions. Note that connections are made at the Group level; therefore, the connections can be modified easilywithout intrusive re-programming of the Components.

```
from openmdao.api import Group, IndepVarComp
from pyBind import py_FEA

# SIMP Group initialization ==============================================
class SimpGroup(Group):

    def initialize(self):
        # Prescribing shared member variables within group
        self.metadata.declare('fem_solver', type_=py_FEA, required=True)
        self.metadata.declare('force', type_=np.ndarray, required=True)
        self.metadata.declare('num_elem_x', type_=int, required=True)
        self.metadata.declare('num_elem_y', type_=int, required=True)
        self.metadata.declare('penal', type_=(int, float), required=True)
        self.metadata.declare(
            'volume_fraction', type_=(int, float), required=True)

    def setup(self): # Where the Connectivity between components are set up
        fem_solver = self.metadata['fem_solver']
        force = self.metadata['force']
        num_elem_x = self.metadata['num_elem_x']
        num_elem_y = self.metadata['num_elem_y']
        p = self.metadata['penal']
        volume_fraction = self.metadata['volume_fraction']
        (nodes, elem, elem_dof) = fem_solver.get_mesh()

        (length_x, length_y) = (np.max(nodes[:, 0], 0), np.max(nodes[:, 1], 0))
        (num_nodes_x, num_nodes_y) = (num_elem_x + 1, num_elem_y + 1)

        nNODE = num_nodes_x * num_nodes_y
        nELEM = (num_nodes_x - 1) * (num_nodes_y - 1)
        nDOF = nNODE * 2

        # Input component ================================
        # Independent variables are defined as outputs of IndepVarComp component
        comp = IndepVarComp()
        comp.add_output('force', val=force)
        comp.add_output('dvs', val=0.5, shape=nELEM)
        comp.add_design_var('dvs', lower=0.01, upper=1.0) ## Design variables
        self.add_subsystem('cInputs', comp)

        self.connect('cInputs.dvs', 'cFiltering.dvs')
        self.connect('cInputs.rhs', 'cState.rhs')
        self.connect('cInputs.rhs', 'cObjective.forces')
```

```
# density filter ================================
comp = DensityFilterComp(length_x=length_x, length_y=length_y,
            num_nodes_x=num_nodes_x, num_nodes_y=num_nodes_y,
            num_dvs=nELEM, radius=length_x / (float(num_nodes_x) - 1) * 2)
self.add_subsystem('cFiltering', comp)
self.connect('cFiltering.dvs_bar', 'cPenalty.x')
self.connect('cFiltering.dvs_bar', 'cConstraint.x')

# penalization ==================================
comp = PenalizationComp(num=nELEM, p=p)
self.add_subsystem('cPenalty', comp)
self.connect('cPenalty.y', 'cState.multipliers')

# states ========================================
comp = StatesComp(fem_solver=fem_solver,
                  num_nodes_x=num_nodes_x, num_nodes_y=num_nodes_y,
                  isSIMP = True)
self.add_subsystem('cState', comp)
self.connect('cState.states', 'cObjective.disp')

# compliance ====================================
comp = ComplianceComp(num_nodes_x=num_nodes_x, num_nodes_y=num_nodes_y)
comp.add_objective('compliance') ## Objective
self.add_subsystem('cObjective', comp)

# weight ========================================
comp = WeightComp(num=nELEM)
comp.add_constraint('weight', upper=volume_fraction) ## Constraint
self.add_subsystem('cConstraint', comp)
```

Acknowledgement

We acknowledge the support of the NASA Transformational Tools and Technologies Project, contract number NNX15AU22A.

Conflict of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Replication of Results

To comply with Replication of Results and help the reader in using the present work in research and education, the codes of the present work is open to public e codes(`https://github.com/chungh6y/openmdao_TopOpt`). Instruction for installation and running a program can be found therein.

References

1. Aage, N., Andreassen, E., Lazarov, B.S.: Topology optimization using petsc: An easy-to-use, fully parallel, open source topology optimization framework. Structural and Multidisciplinary Optimization **51**(3), 565–572 (2015)
2. Allaire, G., Jouve, F., Toader, A.M.: Structural optimization using sensitivity analysis and a level-set method. Journal of Computational Physics **194**(1), 363–393 (2004)
3. Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B.S., Sigmund, O.: Efficient topology optimization in MATLAB using 88 lines of code. Structural and Multidisciplinary Optimization **43**(1), 1–16 (2011)

4. Antoine, L.: A level set-based structural optimization code using fenics. arXiv preprint arXiv:1705.01442 (2017)
5. Behnel, S., Bradshaw, R., Seljebotn, D.S., Ewing, G., et al.: Cython: C-extensions for python (2008)
6. Belytschko, T., Xiao, S., Parimi, C.: Topology optimization with implicit functions and regularization. International Journal for Numerical Methods in Engineering **57**(8), 1177–1196 (2003)
7. Bendsøe, M.P., Sigmund, O., Bendsøe, M.P., Sigmund, O.: Topology optimization by distribution of isotropic material. Springer (2004)
8. van Dijk, N.P., Maute, K., Langelaar, M., Keulen, F.V.: Level-set methods for structural topology optimization: a review. Structural and Multidisciplinary Optimization **48**(3), 437–472 (2013)
9. Dunning, P., Kim, H.: A new method for creating holes in level-set function based topology optimisation. International Journal for Numerical Methods in Engineering (2013)
10. Dunning, P.D., Kim, H.A.: Introducing the sequential linear programming level-set method for topology optimization. Structural and Multidisciplinary Optimization **51**(3), 631–643 (2015)
11. Dunning, P.D., Kim, H.A., Mullineux, G.: Investigation and improvement of sensitivity computation using the area-fraction weighted fixed grid fem and structural optimization. Finite Elements in Analysis and Design **47**(8), 933–941 (2011)
12. Falck, R.D., Chin, J.C., Schnulo, S.L., Burt, J.M., Gray, J.S.: Trajectory optimization of electric aircraft subject to subsystem thermal constraints. In: 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. Denver, CO (2017)
13. Gray, J.S., Hearn, T.A., Moore, K.T., Hwang, J., Martins, J., Ning, A.: Automatic evaluation of multidisciplinary derivatives using a graph-based problem formulation in openMDAO. In: 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. American Institute of Aeronautics and Astronautics (2014). DOI doi:10.2514/6.2014-2042. URL `http://dx.doi.org/10.2514/6.2014-2042`
14. Gray, J.S., Moore, K.T., Naylor, B.A.: Openmdao: An open-source framework for multidisciplinary analysis and optimization. In: 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Fort Worth, TX, AIAA, AIAA-2010-9101. AIAA, Fort Worth, Texas (2010). URL `http://www.aric.or.kr/treatise/journal/content.asp?idx=134451`
15. Hedges, L.O., Kim, H.A., Jack, R.L.: Stochastic level-set method for shape optimisation. Journal of Computational Physics **348**, 82–107 (2017)
16. Hwang, J.T., Lee, D.Y., Cutler, J.W., Martins, J.R.R.A.: Large-scale multidisciplinary optimization of a small satellite's design and operation. Journal of Spacecraft and Rockets **51**(5), 1648–1663 (2014). DOI 10.2514/1.A32751
17. Hwang, J.T., Martins, J.R.R.A.: Allocation-mission-design optimization of next-generation aircraft using a parallel computational framework. In: 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference. American Institute of Aeronautics and Astronautics (2016). DOI 10.2514/6.2016-1662
18. Hwang, J.T., Martins, J.R.R.A.: A computational architecture for coupling heterogeneous numerical models and computing coupled derivatives. ACM Transactions on Mathematical Software (2018). (In press)
19. Kambampati, S., Jauregui, C., Museth, K., Kim, H.: Fast level set topology optimization using a hierarchical data structure. In: AIAA Aviation and Aeronautics Forum and Exposition 2018 (2018)
20. Lambe, A.B., Martins, J.R.: Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes. Structural and Multidisciplinary Optimization **46**(2), 273–284 (2012)
21. Liu, K., Tovar, A.: An efficient 3D topology optimization code written in matlab. Structural and Multidisciplinary Optimization **50**(6), 1175–1196 (2014)
22. Otomori, M., Yamada, T., Izui, K., Nishiwaki, S.: Matlab code for a level set-based topology optimization method using a reaction diffusion equation. Structural and Multidisciplinary Optimization **51**(5), 1159–1172 (2015)

23. Pingen, G., Waidmann, M., Evgrafov, A., Maute, K.: A parametric level-set approach for topology optimization of flow domains. Structural and Multidisciplinary Optimization **41**(1), 117–131 (2010)
24. Sigmund, O.: A 99 line topology optimization code written in matlab. Structural and multidisciplinary optimization **21**(2), 120–127 (2001)
25. Sivapuram, R., Dunning, P.D., Kim, H.A.: Simultaneous material and structural optimization by multiscale topology optimization. Structural and multidisciplinary optimization **54**(5), 1267–1281 (2016)
26. Svanberg, K., Svard, H.: Density filters for topology optimization based on the geometric harmonic means. In: 10th world congress on structural and multidisciplinary optimization. Orlando (2013)
27. Wei, P., Li, Z., Li, X., Wang, M.Y.: An 88-line matlab code for the parameterized level set method based topology optimization using radial basis functions. Structural and Multidisciplinary Optimization pp. 1–19 (2018)