Structural and Multidisciplinary Optimization manuscript No.
(will be inserted by the editor)

Direct Lagrange Multiplier Updates in Topology Optimization Revisited

Tej Kumar · **Krishnan Suresh**

Received: date / Accepted: date

Abstract In topology optimization, the *bisection method* is typically used for computing the Lagrange multiplier associated with a constraint. While this method is simple to implement, it leads to oscillations in the objective and could possibly result in constraint failure if proper scaling is not applied. In this paper, we revisit an alternate and *direct method* to overcome these limitations.

The direct method of Lagrange multiplier computation was popular in the 70s and 80s but was later replaced by the simpler bisection method. In this paper, we show that the direct method can be generalized to a variety of linear and nonlinear constraints. Then, through a series of benchmark problems, we demonstrate several advantages of the direct method over the bisection method including: (1) fewer and faster update iterations, (2) smoother and robust convergence, and (3) insensitivity to material and force parameters. Finally, to illustrate the implementation of the direct method, drop-in replacements to the bisection method are provided for popular Matlab-based topology optimization codes.

Keywords Topology Optimization · Optimality Criteria · Bisection · Design Constraints · Lagrange Multiplier · Design Update

Tej Kumar
Department of Mechanical Engineering
University of Wisconsin-Madison
E-mail: tkumar3@wisc.edu
ORCiD: 0000-0001-8762-8121

Krishnan Suresh
Department of Mechanical Engineering
University of Wisconsin-Madison
E-mail: ksuresh@wisc.edu

1 Introduction

Topology optimization is now a well established method for computing optimal material distribution within a design domain that extremizes an objective while meeting a set of constraints. Popular topology optimization methods include density methods [6, 29], level-set [27, 37], topological derivative [31], evolutionary methods [40], etc. Density methods, in particular, "Solid Isotropic Material with Penalization" (SIMP) are the most popular today. In SIMP, finite element method is used as the analysis engine, and each finite element e is associated with a design variable x_e. Then the topology optimization problem is posed as:

$$\begin{aligned} &\underset{x}{\text{minimize}} && J(x,u) && \text{(1a)}\\ &\text{subject to} && K(x)u = f && \text{(1b)}\\ &&& g(x,u) \leq g^* && \text{(1c)}\\ &&& \underline{x} \leq x_e \leq \overline{x} \qquad \forall e && \text{(1d)} \end{aligned}$$

where the objective function J is dependent on the design variables x and state variables u. The latter is computed via the governing Eqn. (1b) where $K(x)$ is the stiffness matrix and f is the force vector.

Note that the design constraint is defined via Eqn. (1c), while Eqn. (1d) are the box constraints that sets lower ($\underline{x}$) and upper bounds ($\overline{x}$) on the design variables. A typical instance of the above problem is compliance minimization where:

$$J(x,u) = u^{\mathsf{T}} K(x) u \tag{2}$$

subject to a volume constraint:

$$g(x) = \sum_e x_e v_e \tag{3}$$

where v_e is the volume of element-e. There are several open-source SIMP-based codes for solving such problems; the 99-line code [28] being the first. This was later improved for speed in the 88-line version [4]. Several codes followed, and these are listed in Table 1.

A typical density-based method is described in Algorithm 1 which consists of outer and inner iterations. The former corresponds to the outer-loop where a finite element problem is solved to compute design sensitivities, while the latter corresponds to the inner-loop where the design variables and Lagrange multipliers are updated [4]. Inner iterations are also referred to as design update iterations or simply update iterations.

Algorithm 1 Density-based topology optimization

```
1: Initialize variables
2: while $\max(|\Delta x|) > 0.01$ do        ▷ Outer-loop; typical convergence criteria
3:     Analyze and compute sensitivity
4:     while Constraint not satisfied do        ▷ Inner-loop;
5:         Update design variable $x$ and Lagrange multiplier
6:     end while
7: end while
```

In the inner-loop, a *design variable update* can be performed using various methods. For example, mathematical programming methods such as CONLIN [14] or method of moving asymptotes (MMA) [32] is used for multiple constraints [10], whereas *optimality criteria (OC)* is more popular [9, 35, 16] for single constraint problems [4, 2] due to its speed. The equivalence between mathematical programming methods and optimality criteria was established by Fleury and co-workers [13, 26, 14] and later by many other researchers [5, 16, 3, 15]. In OC, the design variables are identified as active or passive, where a design variable is *active* if it lies strictly between the bounds of box constraint (1d), else it is *passive* [35, 16]. Then by representing the ratio of objective sensitivity to constraint sensitivity as:

$$R_e = \frac{\frac{\partial J}{\partial x_e}}{\frac{\partial g}{\partial x_e}} \tag{4}$$

the active design variables are updated in density-based methods as follows [28, 4]:

$$x_e^{\text{new}} = x_e(-R_e/\Lambda)^{\eta} \tag{5}$$

where η is a damping coefficient (typically 0.5), and Λ is the *unknown* Lagrange multiplier. Eqn. (5) is typically used for volume-constrained compliance minimization given in Eqn. (2) and (3). In some cases, e.g., compliance-constrained volume-minimization [2, 1], the following update for active design variables is used

$$x_e^{\text{new}} = x_e(\Lambda/(-R_e))^{\eta} \tag{6}$$

Hereon, the former definition is used unless stated otherwise. Further, to ensure that all design variables lie within the box constraints, additional limits are imposed [28, 4]:

$$x_e^{\text{new}} = \max\left(\underline{x_e}, \min\left(\overline{x_e}, x_e(-R_e)^{\eta}\Lambda^{-\eta}\right)\right) \tag{7}$$

where $\underline{x_e} = \max(\underline{x}, x_e - \mu)$, $\overline{x_e} = \min(\overline{x}, x_e + \mu)$ are the modified box constraints, with μ as the move limit (typically 0.2). Thus, at the end of each inner iteration, the design variable lie within the modified box constraint:

$$\underline{x_e} \le x_e^{\text{new}} \le \overline{x_e} \tag{8}$$

Note that the Lagrange multiplier Λ in Eqn. (5) is unknown, and is typically computed using the bisection method, the main topic of this paper. The bisection method is described in Section 2, followed by a summary of its drawbacks. The objective of this paper is to revisit a direct method for computing Λ that relies on the definition of active and passive design variables. The direct method is described in Section 3, and illustrated using various constraint scenarios. Several numerical examples illustrating the merits of the direct method are analyzed in Section 4. Conclusions and sample code are provided in Section 5 and Appendix, respectively.

2 Bisection Method

The bisection method is the *de facto* standard for estimating the Lagrange multiplier Λ in Eqn. (5). Every open source code in Table 1 except Sui and Yi [30] uses this method. A typical implementation of the bisection method is summarized in Algorithm 2. It starts by initializing two bounds Λ_1 and Λ_2 on the Lagrange multiplier via two constants $\underline{\Lambda}$ and $\overline{\Lambda}$. The lower bound $\underline{\Lambda}$ is almost always zero whereas the upper bound $\overline{\Lambda}$ is a large constant. Next, the inner iteration is carried out where the design variables x^{new} are updated using Eqn. (7), using the mean value Λ_{mid} of Λ_1 and Λ_2 as the estimation for the Lagrange multiplier. Then, the bounds Λ_1 and Λ_2 are updated depending on the violation of the design constraint as in Algorithm 2. The iteration terminates when either $(\Lambda_2 - \Lambda_1)$ or $(\Lambda_2 - \Lambda_1)/2\Lambda_{\text{mid}}$ is relatively small (some authors choose the former, while others prefer the latter).

While the bisection method is simple and effective, it only approximates the Lagrange multiplier Λ to an accuracy dictated by the convergence constant ε. Consequently, since the associated constraint is not satisfied exactly, oscillations can be observed in the objective as demonstrated later via numerical examples. Finally, since the value of the Lagrange multiplier Λ depends on the (scaling of the) problem parameters, one should suitably choose the search window defined by $\underline{\Lambda}$ and $\overline{\Lambda}$, i.e., if Λ lies outside the search window, the bisection method may fail to converge. For example, for the MBB beam problem, with $E = 1$ and $F = 1000$, over a

Table 1: Open source codes that use SIMP.

Year	Authors	Objective	Code	Description
2001 [28]	O. Sigmund	Compliance	top	99-line code
2003 [8]	M. P. Bendsøe, O. Sigmund	Output displacement, thermal compliance,	topm, toph	Compliant mechanism and thermal optimization
2004 [17]	G. Kharmanda, N. Olhoff, A. Mohamed, M. Lemaire	Compliance	RBTO	Reliability based optimization
2011 [4]	E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, O. Sigmund	Compliance	top88	88-line code: faster than 99-line code, includes density filter
2012 [33]	C. Talischi, G.H. Paulino, A. Pereira, I.F.M. Menezes	Compliance	PolyTop	Polygonal mesh
2013 [23]	X. Qian	Compliance	btop85	Spline basis function
2013 [30]	Y. Sui, G. Yi	Volume	top120	Displacement constraint
2014 [21]	K. Liu, A. Tovar	Compliance	top3D	3D code
2014 [2, 1]	N. Aage, B. S. Lazarov, O. Amir	Compliance	top2dmgcg	Multigrid conjugate gradient solver
2014 [34]	R. Tavakoli, S.M. Mohseni	Compliance	multitop	Multimaterial
2015 [39]	L. Xia, P. Breitkopf	Bulk modulus, Shear modulus	topX	Design of material
2017 [42]	W. Zuo, K. Saitou	Compliance	top	Multimaterial
2018 [25]	E.D. Sanders, A. Pereira, M.A. Aguilo, G.H. Paulino	Compliance	PolyMat	Multimaterial code using PolyTop
2019 [20]	T. Kumar, K. Suresh	Compliance	MTO	Multiscale design
2019 [38]	S. Watts, W. Arrighi, J. Kudo, D. A. Tortorelli, D. A. White	Compliance	MultiScale-TopOpt	Lattice surrogate models

Algorithm 2 Bisection method

```
 1: $\Lambda_1 \leftarrow \underline{\Lambda}, \Lambda_2 \leftarrow \overline{\Lambda}$          ▷ Constants $\underline{\Lambda}$ and $\overline{\Lambda}$;
 2: $\Lambda_{\text{mid}} \leftarrow (\Lambda_1 + \Lambda_2)/2$
 3: while $(\Lambda_2 - \Lambda_1) < \varepsilon$ (or $(\Lambda_2 - \Lambda_1)/2\Lambda_{\text{mid}} < \varepsilon$) do   ▷ An arbitrary
    small constant $\varepsilon$
 4:     Compute $x^{\text{new}}$ wherein $\Lambda \leftarrow \Lambda_{\text{mid}}$          ▷ Using Eqn. (7)
 5:     if $g(x^{\text{new}}) > g^*$ then
 6:         $\Lambda_1 \leftarrow \Lambda_{\text{mid}}$
 7:     else
 8:         $\Lambda_2 \leftarrow \Lambda_{\text{mid}}$
 9:     end if
10:     $\Lambda_{\text{mid}} \leftarrow (\Lambda_1 + \Lambda_2)/2$
11: end while
12: $x \leftarrow x^{\text{new}}$
```

60×20 mesh, with a target volume fraction of 0.5, the typical value for Λ is around 6×10^5 (it varies during the optimization process). The 99-line code, with the default search window of $\underline{\Lambda} = 0$ and $\overline{\Lambda} = 1 \times 10^5$, therefore fails to converge.

3 Direct Computation of Lagrange Multiplier

To address these limitations, we revisit the direct method for computing the Lagrange multiplier. This method was used during 70s and 80s [36, 18, 9, 35] for structural optimization. Although the description "direct method" was not used then, we use that term here to distinguish it from the bisection method. The direct method satisfies the design constraint exactly by assuming that the design constraint (for example, the volume constraint) remains active; indeed, the bisection method also relies on this assumption.

As it is evident from Table 1, the bisection method overshadowed the direct method in later years. However, more recently, the direct method was used in a displacement constrained, volume-minimization topology optimization problem [30]. Here, we revisit the direct method, providing and illustrating its MATLAB implementation, for a series of constrained problems. The direct method is characterized by (1) exact satisfaction of the design constraint, obviating the need for scaling and search window for the Lagrange multiplier, and (2) applicability for various combinations of objective and constraints, provided these do not involve a nonlinear coupling of the variables, or a separable approximation is employed, as explained later in the paper.

3.1 Concept

For simplicity, consider the volume-constrained, compliance-minimization problem discussed earlier. Since the volume constraint remains active i.e. $\sum_e x_e v_e = g^*$, the updated design variables must satisfy:

$$\sum_e x_e^{\text{new}} v_e = g^* \tag{9}$$

i.e.,

$$\sum_e x_e(-R_e)^\eta \Lambda^{-\eta} v_e = g^* \tag{10}$$

Therefore, the Lagrange multiplier can be directly computed via:

$$\Lambda = \left(\frac{\sum_e x_e(-R_e)^\eta v_e}{g^*} \right)^{\frac{1}{\eta}} \tag{11}$$

This can now be substituted in Eqn. (5) to compute the new design variables. However, the new design variables can violate the modified box constraints (8). To correct for this, we separate the design variables into two sets: S_P is the set of passive design variables that do not satisfy the modified box constraints, and S_A is the set of active design variables that satisfy the modified box constraints. As before, Eqn. (8) is enforced on all design variables in S_P. This implies that the active variables must satisfy:

$$\sum_{i \in S_A} x_i^{\text{new}} v_i = g^* - \sum_{i \in S_P} x_i^{\text{new}} v_i \tag{12}$$

Thus Λ is recomputed via:

$$\Lambda = \left(\frac{\sum_{i \in S_A} x_i(-R_i)^\eta v_i}{g^* - \sum_{i \in S_P} x_i^{\text{new}} v_i} \right)^{\frac{1}{\eta}} \tag{13}$$

The process is repeated until no change in observed in the two sets of variables (as opposed to checking for convergence in Λ within a certain tolerance). The direct method is summarized in Algorithm 3.

Note that the direct method does not require scaling, i.e., the constraint is satisfied exactly for all values of E and F since there are no artificial bounds on the Lagrange multiplier. Further, as opposed to the bisection method with a convergence criteria $(\Lambda_2 - \Lambda_1)/2\Lambda_{\text{mid}} < \varepsilon$, the total number of inner iterations remain constant for all values of E and F. This is consistent with the well-known observation

Algorithm 3 Direct method

```
 1: $S_A = \{1 \dots N\}$, $S_P = \phi$    ▷ Assign all $N$ elements to $S_A$ and let $S_P$ be empty
 2: $\Delta S_P \leftarrow 1$    ▷ $\Delta S_P$: Change in set $S_P$
 3: while $\Delta S_P$ do
 4:     Compute $\Lambda$    ▷ Using Eqn. (13)
 5:     Compute $x^{\text{new}}$    ▷ Using Eqn. (5)
 6:     Set $S_P = \{i | x_i^{\text{new}} < \underline{x_i} | x_i^{\text{new}} > \overline{x_i}\}$ and identify $\Delta S_P$
 7:     $S_A = \{1 \dots N\} - S_P$
 8:     Update $x_i^{\text{new}}\ \forall i \in S_P$    ▷ Eqn. (8)
 9: end while
10: $x \leftarrow x^{\text{new}}$
```

that the scaling of E and F does not change the final design in linear elastic problems, and therefore, the characteristics of optimizer such as the number of inner iterations should not change. In summary, the direct method: (1) avoids initial search window and convergence tolerance for Lagrange multiplier, (2) obviates the scaling of objective or constraint, and (3) obviates the scaling of E and F.

3.2 Generalizations

In the previous section, we considered the simple case of a volume-constrained, compliance-minimization problem. The direct method can be generalized to other problems as well, as explained in this section.

3.2.1 Generalized Linear Constraint

Consider the case when the constraint is a generic linear combination of the design variables:

$$g(x) = \sum_e \sum_i g_{ei} x_i \tag{14}$$

Once again, *assuming that the design constraint is active*, the Lagrange multiplier can be computed via:

$$\Lambda = \left(\frac{\sum_{i \in S_A} \left(x_i(-R_i)^\eta \sum_e g_{ei} \right)}{g^* - \sum_{i \in S_P} \left(x_i^{\text{new}} \sum_e g_{ei} \right)} \right)^{\frac{1}{\eta}} \tag{15}$$

This leads to the design update equation:

$$x_e^{\text{new}} = x_e(-R_e)^\eta \left(\frac{g^* - \sum_{i \in S_P} \left(x_i^{\text{new}} \sum_e g_{ei} \right)}{\sum_{i \in S_A} \left(x_i(-R_i)^\eta \sum_e g_{ei} \right)} \right) \quad \forall e \in S_A \tag{16}$$

3.2.2 Filtered Design Variables

It is fairly common in topology optimization to use filtered design variables [4]:

$$\tilde{x}_e = \frac{\sum_{i \in N_e} H_{ei} x_i}{\sum_{i \in N_e} H_{ei}} = \sum_{i \in N_e} \tilde{H}_{ei} x_i \tag{17}$$

where $\tilde{H}_{ei} = H_{ei}/\sum_{i\in N_e} H_{ei}$ serves as a smoothening filter. Observe that the filter can be expressed as

$$h_{ei} = \begin{cases} \tilde{H}_{ei} & \text{if } i \in N_e \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

that leads to:

$$\tilde{x}_e = \sum_i h_{ei} x_i \tag{19}$$

Thus the constraint can be expressed as:

$$g(x) = \sum_e \tilde{x}_e v_e = \sum_e \sum_i h_{ei} x_i v_e \tag{20}$$

Comparing Eqn. (20) and Eqn. (14), we have $g_{ei} = h_{ei} v_e$ (direct multiplication of the two scalars is implied here, not a summation); thus Eqns. (15) and (16) directly apply.

3.2.3 Multiple Volume Constraints

Sometimes multiple volume constraints are imposed for achieving local volume control [24], or in multi-material optimization, constraints are applied to assign one or more materials to a region of interest [25]. In both cases, one can identify mutually exclusive set of design variables associated with k^{th} constraint represented by G_k. This leads to a set of constraint equations:

$$\sum_{e\in G_k} v_e x_e = g^*_k \qquad k = 1 \ldots N_c \tag{21}$$

for N_c volume constraints. Since the sets G_k are mutually exclusive, each Lagrange multiplier Λ_k, and the corresponding set of design variables x_e can be computed using Algorithm 3 independently. On the other hand, when the constraints are imposed on the filtered design variables $\tilde{x}_e$, then we have:

$$\sum_{e\in G_k} v_e \sum_i h_{ei} x_i = g^*_k \qquad k = 1 \ldots N_c \tag{22}$$

Each constraint is now dependent on all the design variables, i.e., the sets are not mutual exclusive. In this case, the filtered design variables $\tilde{x}_e$ which are still mutually exclusive, are used to update individual Lagrange multiplier Λ_k. However, for imposing the modified box constraint, Eqn. (17) is solved by inverting the filter matrix $[h_{ei}]$. Finally, the sets S_P and S_A are identified as in Algorithm 3 and the procedure continues till convergence. An alternative is to use constraint approximation discussed below for nonlinear constraints.

Another class of multi-constraint problem is where one design variable might be directly shared across two or more constraints. In this case, while active-set OC algorithm was suggested in [16, 41], mathematical programming methods such as method of moving asymptotes (MMA) [32], are perhaps a better alternative [8].

3.2.4 Nonlinear Constraints

We now consider the case when $g(x)$ is a nonlinear function of the design variables. This can typically arise in a *compliance-constrained* or *displacement-constrained*, volume-minimization problem where the design variables are nonlinearly coupled. Here, it is typical to rely on local approximations [1]. For example, a simple Taylor series approximation leads to:

$$g(x^{\text{new}}) = g(x) + \sum_e \left(\frac{\partial g}{\partial x_e} (x^{\text{new}}_e - x_e) \right) = g^*$$

Rearranging terms:

$$\sum_e \frac{\partial g}{\partial x_e} x^{\text{new}}_e = \sum_e \frac{\partial g}{\partial x_e} x_e + (g^* - g(x)) \tag{23}$$

Thus, we arrive at the expression for Λ using alternate definition of active design variable given by Eqn. (6):

$$\Lambda = \left(\frac{g^* - g(x) + \sum_{i\in S_A} \frac{\partial g}{\partial x_i} x_i - \sum_{i\in S_P} \left((x^{\text{new}}_i - x_i) \frac{\partial g}{\partial x_i} \right)}{\sum_{i\in S_A} \left(x_i (-R_i)^{-\eta} \frac{\partial g}{\partial x_i} \right)} \right)^{\frac{1}{\eta}} \tag{24}$$

In the case of displacement-constrained problem, the sensitivity of displacement is computed via the adjoint method [19, 11].

For other approximations, the Taylor series expansion typically involves an intermediate variable $y_e(x_e)$ as:

$$g(y^{\text{new}}) = g(y) + \sum_e \left(\frac{\partial g}{\partial y_e} (y^{\text{new}}_e - y_e) \right) \tag{25}$$

For example, the popular *reciprocal approximation* [13] is achieved by substituting $y_e = 1/x_e$, leading to the approximation:

$$g(x^{\text{new}}) = g(x) + \sum_e x_e \frac{\partial g}{\partial x_e} \left(1 - \frac{x_e}{x^{\text{new}}_e} \right) = g^* \tag{26}$$

This leads to:

$$\Lambda = \left(\frac{g(x) - g^* + \sum_{i\in S_A} x_i \frac{\partial g}{\partial x_i} + \sum_{i\in S_P} x_i \frac{\partial g}{\partial x_i} \left(1 - \frac{x_i}{x^{new}_i} \right)}{\sum_{i\in S_A} \left(x_i (-R_i)^{\eta} \frac{\partial g}{\partial x_i} \right)} \right)^{\frac{1}{\eta}} \tag{27}$$

again using Eqn. (6). Similarly, we have the *exponential approximation* [12] obtained by substituting $y_e = x^{\alpha}_e$, leading to:

$$g(x^{\text{new}}) = g(x) + \sum_i \left[\left(\frac{x^{new}_i}{x_i} \right)^{\alpha} - 1 \right] \frac{x_i}{\alpha} \frac{\partial g}{\partial x_i} = g^* \tag{28}$$

The Lagrange multiplier is obtained via:

$$\Lambda = \left(\frac{g^* - g(x) + \sum_{i \in S_A} \frac{x_i}{\alpha} \frac{\partial g}{\partial x_i} - \sum_{i \in S_P} \left[\left(\frac{x_i^{new}}{x_i} \right)^{\alpha} - 1 \right] \frac{x_i}{\alpha} \frac{\partial g}{\partial x_i}}{\sum_{i \in S_A} (-R_i)^{-\alpha\eta} \frac{x_i}{\alpha} \frac{\partial g}{\partial x_i}} \right)^{\left(\frac{1}{\alpha\eta}\right)} \tag{29}$$

Each of the above expressions for Λ can be substituted in Eqn. (6) to compute the active design variables. In compliance-minimization, when Eqn. (6) must be used instead of Eqn. (5), then Λ will be the reciprocal of Eqns. 24, 27, and 29.

3.2.5 Vanishing g^*

Finally, consider the rare case of $g(x) \le 0$ i.e. $g^* = 0$ in Eqn. (1c). In this case, a divide-by-zero error occurs while solving Eqn. (13) in the very first iteration of Algorithm 3 since the passive set S_P is empty. To overcome the issue, one of the constraint approximations can be used. For instance, using a linear approximation with Eqn. (5), even though g^* and $g(x)$ can be zero, the third term in the denominator (in the reciprocal form of Eqn. (24)) i.e. $\sum_{i \in S_A} \frac{\partial g}{\partial x_i} x_i$ must be non-zero and thus the divide-by-zero error is avoided.

3.2.6 Handling Pathological Conditions

While computing the Lagrange multiplier, scenarios where either the numerator or the denominator in, say Eqn. (13), is zero or negative must be avoided. Observe that R_i is always negative, v_i is always positive, and every x_i in active set is positive; therefore the numerator can be zero only when the active set S_A is empty. The denominator can be zero or negative typically when the initial design point is far from satisfying the constraint, e.g., starting with a fully solid design (all $x_i = 1$) for a desired volume fraction of 0.5. Both these issues are easily resolved by increasing the prescribed move limit μ; please see the full implementation in Appendix.

4 Numerical Experiments

We now compare the bisecton method against the direct method through numerical experiments. The MBB beam [4] is used for the 2D experiments, whereas the edge-loaded cantilever problem [21] is chosen for the 3D experiment. The first three experiments involve a volume-constrained, compliance minimization problem; for the remaining experiments, the underlying topology optimization problem will be described. The objective of the numerical experiments are to investigate the following questions:

1. **Impact on Topology, Objective and Computational Time**: First, we study the impact of the two methods on the final topology, objective, and computational time.
2. **Oscillations in Objective**: It was noted earlier that the bisection method could lead to oscillations in the objective; we explore if the direct method resolves this issue.
3. **Generalization to 3D**: We then briefly investigate the above questions in 3D.
4. **Constraint Approximation**: Next, we consider the performance of the direct method to solve a (nonlinear) compliance constrained problem; various constraint approximations are considered.
5. **Multi-Constraint**: The applicability of the direct method to handle multiple constraints is explored through a multi-material topology optimization problem.
6. **Multi-resolution**: The performance of the direct method when more than one design variable is associated with each finite element is studied through a multi-resolution topology optimization problem.
7. **Ease of Implementation**: Finally, we consider the implementation of the direct method, as a drop-in replacement to the bisection method, in several open-source software.

For the purpose of this paper, only the inner iteration in various topology optimization codes is modified by replacing the bisection method with the direct method.

4.1 Impact on Topology, Objective and Computational Time

First, we consider the impact of the direct method on the final topology, objective and computational time. In particular, the three mesh sizes and filters described in [4] were considered in the experiment while using the 88-line code provided therein. The topologies obtained via the direct method are illustrated in Fig. 1, and they are indistinguishable from those obtained via the bisection method [4].

The final objective, the total computational time for optimization, time for inner iterations, and the number of outer (and average inner) iterations are reported in Table 2, for the two methods, and for two different filters. While the objective and the number of outer-iterations are almost identical, the average number of inner iterations reduces almost by a factor of 10, and therefore, the time for inner iteration reduces, especially when a density filter is used, the reduction is reflected in the reduction in total computational time. This is to be expected since the density filter is applied in the inner iteration, while the sensitivity filter is applied only in the outer iteration. This reduction was more pronounced as the filter size was increased.

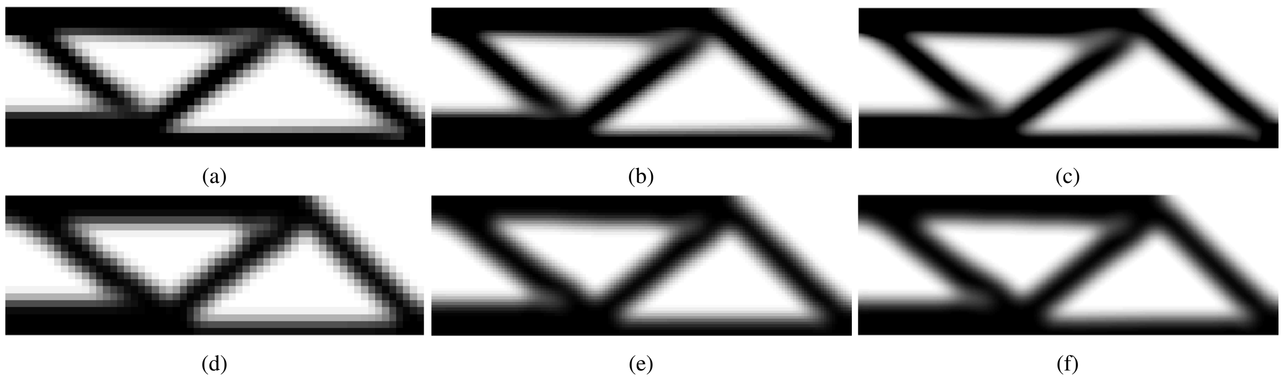


Fig. 1: Optimized topologies of MBB beam using direct method. Mesh: 60×20 (left), 150×50 (middle), and 300×100 (right). Top row: sensitivity filter, bottom row: density filter.

Table 2: Bisection versus direct method: 2D MBB problem using two types of filters and three mesh sizes

Filter type	Mesh size	Objective		Total design (inner iteration) time (s)		Outer (avg. inner) iterations	
		Bisection	Direct	Bisection	Direct	Bisection	Direct
Sensitivity filter	60×20	216.81	216.79	3.46 (0.11)	2.96 (0.03)	106 (39.83)	103 (4.49)
	150×50	219.52	219.62	10.17 (0.31)	9.93 (0.06)	95 (41.96)	95 (4.83)
	300×100	222.29	222.23	42.29 (1.00)	41.84 (0.21)	93 (43.96)	94 (4.93)
Density filter	60×20	233.71	233.71	4.52 (0.33)	4.30 (0.10)	144 (39.91)	146 (6.11)
	150×50	235.73	235.74	56.70 (18.02)	44.52 (6.49)	362 (42.95)	363 (7.59)
	300×100	238.31	238.32	783.87 (511.86)	468.60 (196.18)	625 (44.97)	623 (8.69)

4.2 Oscillations in Objective

As noted earlier, since the bisection method approximates the Lagrange multiplier, oscillations are typically observed in the objective. We expect these oscillations to be eliminated when the direct method is used. This is confirmed in Fig. 2a and Fig. 2b that capture the convergence history using the two methods, for a 300×100 mesh 2D MBB problem, using the two filters.

4.3 Extension to 3D

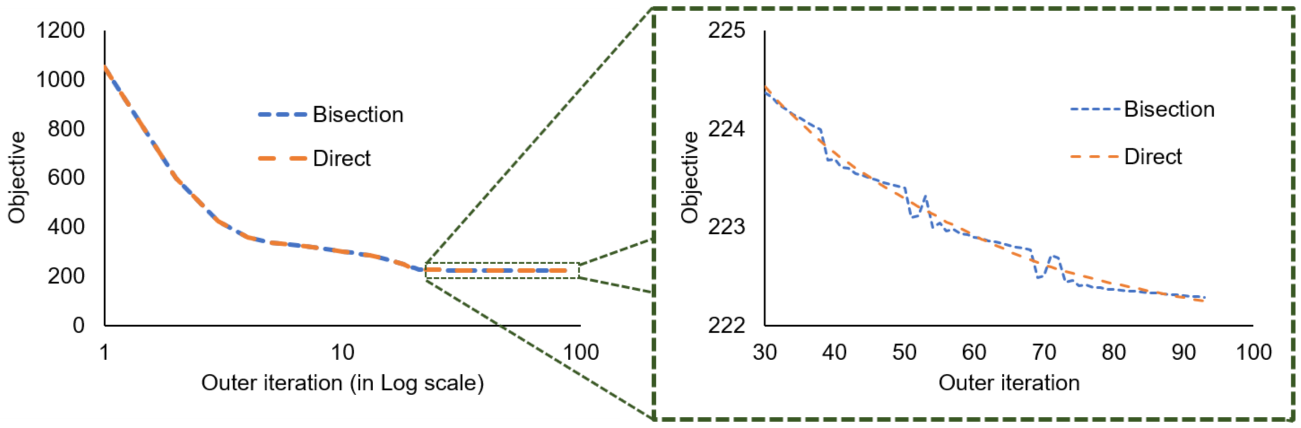
We now repeat the above two experiments in 3D. Among various 3D topology optimization codes [21, 1], the 169 line Matlab code [21] is most similar to the 88-line code [4], and is therefore used here. We consider the 3D cantilever problem described in [21] with a density filter radius of 1.5 times element size. The mesh used here is $40 \times 20 \times 20$ as opposed to $30 \times 10 \times 2$ in [21] in order to obtain a true 3D design instead of an extruded 2D design. The resulting topology using the direct method is illustrated in Fig 3 which is once again indistinguishable from those obtained using the bisection method. Using the bisection method, the algorithm converged with an objective of 1607.50 in 309 iterations, whereas, using the direct method it converged to 1607.55 in 264 iterations. Thus, the final design and objective are indistinguishable, but the direct method is faster.

The convergence history for the 3D problem is illustrated in Fig. 4. Oscillations are once again observed in the bisection method. Further, due to the oscillations, an increased number of outer iterations is observed without any significant improvement in objective (leading to larger computational time).

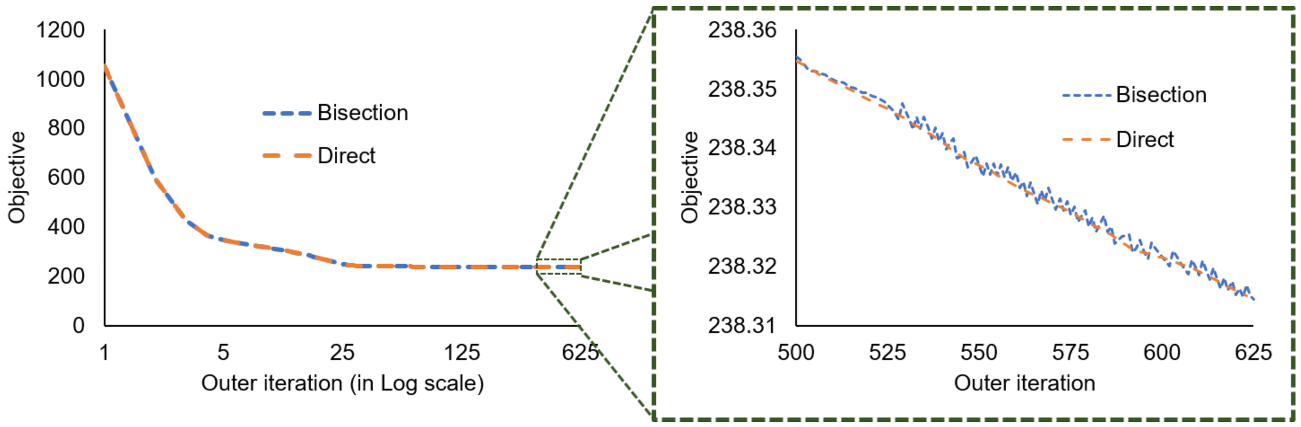
4.4 Nonlinear Constraint

Next we consider a compliance-constrained, volume minimization problem using the code provided in [1]. As stated earlier, the nonlinear constraint mandates an approximation at the design point. Various approximation methods are therefore considered in this experiment. Once again, we use the MBB beam problem with 300×100 mesh, and a compliance constraint of 250, with a filter radius of 2. The final volume fraction, total computational (and inner iteration) time, and the number of outer (and average inner) iterations are summarized in Table 3, for various approximations. The exponent values α are based on the generic approximation in Eqn. (29).

As one can observe, the final volume fraction does not depend on whether the bisection method or the direct method



(a)



(b)

Fig. 2: Convergence history for 300 $\times$ 100 mesh MBB beam: (a) Sensitivity filter (b) Density filter.

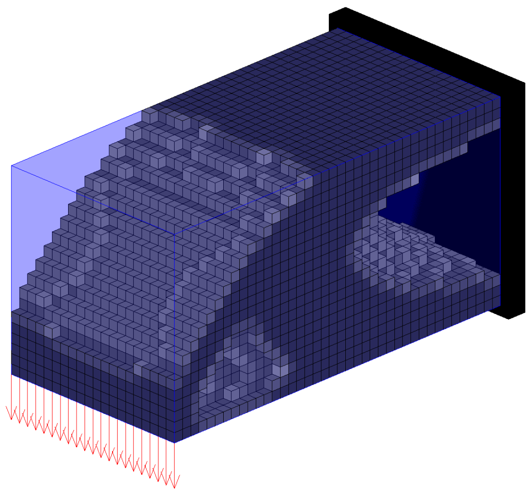


Fig. 3: Optimized design for 3D cantilever beam.

is employed. Since the filter is not applied within the inner iteration even with the density filter, the gain in computational time using the direct method is only marginal. Here, the convergence constant ε used in the bisection method is very small ($\varepsilon = 1 \times 10^{-6}$) compared to $\varepsilon = 1 \times 10^{-3}$ used in 88-line code. This can lead to increased number of inner iterations and high computational cost. A smart heuristic update is implemented where the upper limit $\overline{\Lambda}$ is set to twice the final Lagrange multiplier value from previous inner iteration. For the very first inner iteration, $\overline{\Lambda}$ is set as 1×10^{9}.

Designs for various constraint approximations and filters are illustrated in Fig. 5. These are indistinguishable from those obtained via the bisection method. Further, the designs depend more on the constraint approximation than on the filter type.

4.5 Multiple Constraints

To illustrate the use of the direct method in handling multiple constraints, a *multi-material* problem is considered. In particular, the Matlab code PolyMat [25] is used to solve the MBB problem with 10000 polygonal elements [33]. Two different material distribution problems are considered. In the first problem, four materials with different Young's moduli are assigned to different regions as illustrated in Fig. 6.

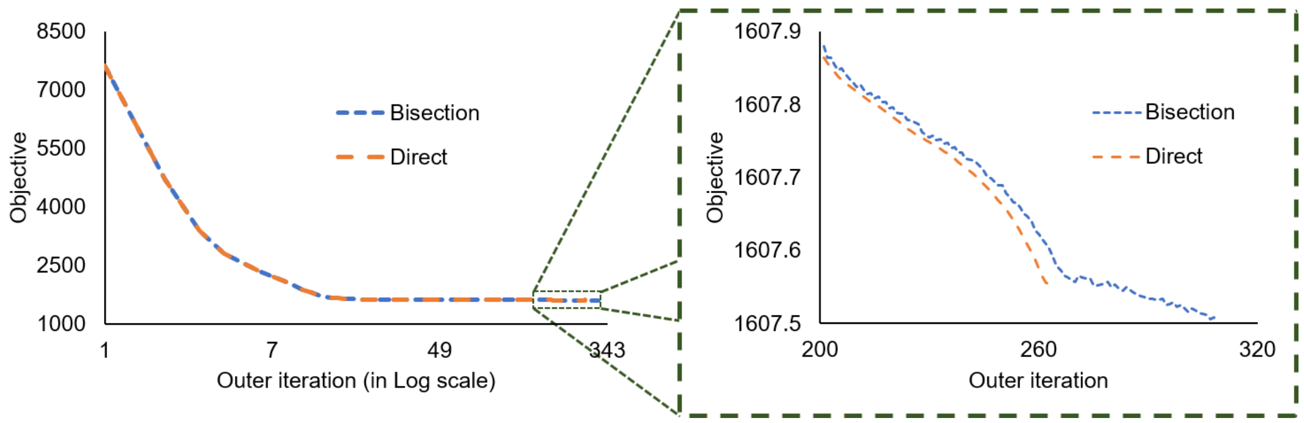


Fig. 4: Convergence history for 3D cantilever beam with density filter

Table 3: Bisection versus direct method: Compliance-constrained, volume minimization.

Constraint approximation	Filter	α	Final volume fraction		Total design (inner iteration) time (s)		Outer (avg. inner) iterations	
			Bisection	Direct	Bisection	Direct	Bisection	Direct
Linear	Sensitivity	(1)	0.357	0.357	117.19 (2.03)	116.50 (1.03)	306 (20.09)	306 (4.78)
	Density		0.370	0.370	779.23 (14.52)	775.43 (11.01)	2029 (20.01)	2029 (8.90)
Reciprocal	Sensitivity	(-1)	0.357	0.357	145.79 (2.66)	144.13 (1.55)	381 (20.06)	381 (6.03)
	Density		0.373	0.373	247.47 (4.76)	247.35 (3.68)	647 (20.04)	647 (8.92)
Exponent	Sensitivity	-0.5	0.356	0.356	151.63 (5.22)	148.12 (2.45)	380 (20.06)	380 (6.12)
		-1.5	0.357	0.357	163.97 (5.75)	161.74 (2.62)	412 (20.07)	412 (6.02)
		-2	0.359	0.359	306.13 (10.64)	303.35 (4.87)	775 (20.03)	775 (6.06)
		-2.5	0.356	0.356	225.41 (7.95)	221.95 (3.64)	584 (20.05)	584 (5.99)
		-3	0.355	0.355	227.42 (8.12)	225.10 (5.23)	584 (20.05)	584 (11.78)
	Density	-0.5	0.372	0.372	157.21 (4.59)	155.99 (3.07)	396 (20.06)	396 (8.87)
		-1.5	0.373	0.373	380.65 (10.46)	379.47 (7.43)	963 (20.02)	964 (9.03)
		-2	0.374	0.374	290.33 (8.40)	289.81 (5.73)	761 (20.03)	761 (8.71)
		-2.5	0.375	0.375	135.11 (4.19)	134.85 (2.63)	352 (20.07)	352 (8.27)
		-3	0.373	0.373	183.66 (5.47)	182.33 (3.09)	476 (20.06)	476 (6.11)

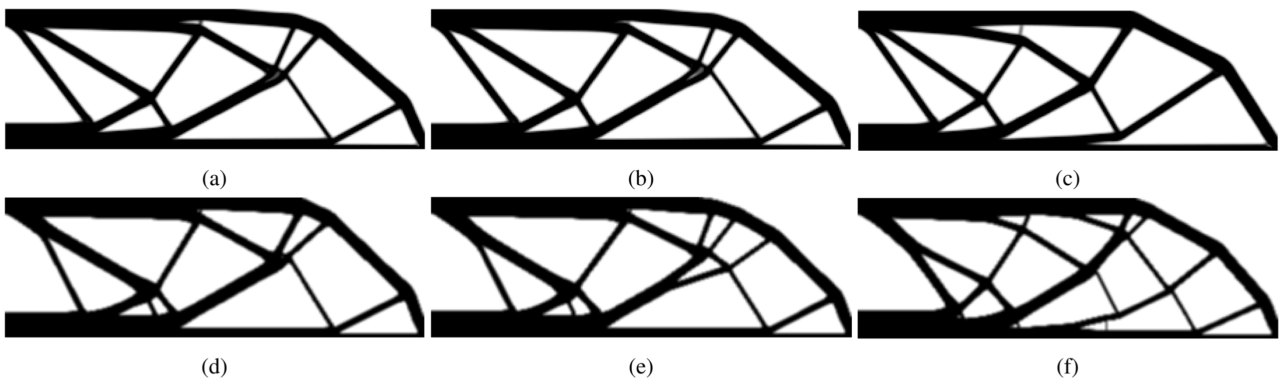


Fig. 5: Design obtained for Linear (*left*), Reciprocal (*middle*), and $\alpha = -3$ Exponential (*right*) constraint approximations and sensitivity (*top*) and density (*bottom*) filter.

In the second problem, the region assignment is removed; instead each material is allowed to occupy 12.5% of the design domain. Both problems involve 4 constraints, a multi-material interpolation model and a continuation scheme [7]. In the PolyMat paper [25], a linear approximation of the constraint was used along with a density filter; this is retained here. Due to the use of constraint approximation, filter is not applied in the inner iteration.

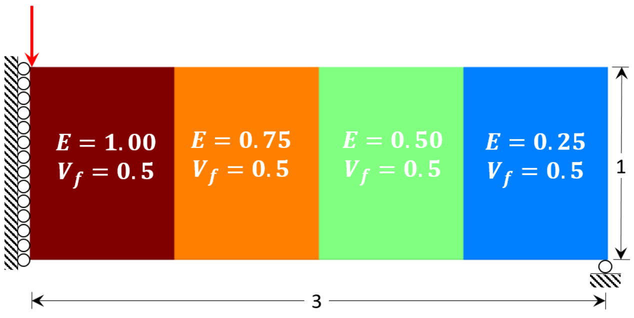


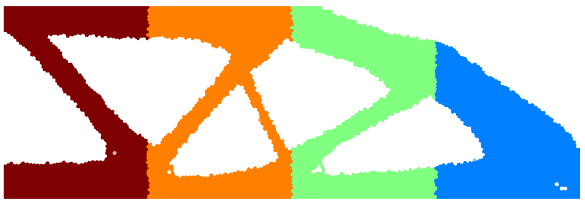
Fig. 6: MBB design domain divided into four regions; each region is assigned a different material with specified volume fraction V_f.

Designs for the two problems using the direct method are illustrated in Fig. 7; these are indistinguishable from the ones obtained by using the bisection method. The objective, total time required for optimization and inner iteration along with number of outer iterations and average number of inner iterations per outer iteration are provided in Table 4. The bisection method in PolyMat employs a convergence criteria $(\Lambda_2 - \Lambda_1) < \varepsilon$ with $\varepsilon = 1 \times 10^{-4}$ and $\overline{\Lambda} = 1 \times 10^{6}$ which gives a constant number (34) of inner iterations, as reported in Table 4. The objective, number of iterations, and the computational time replicates the observations in previous examples where the gain in inner-iteration time via the direct method is insignificant compared to the total time.

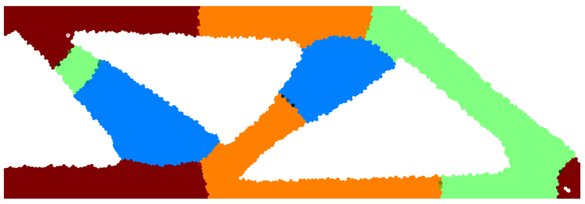
4.6 Multi-resolution

In multi-resolution topology optimization [22], numerous design variables are assigned to a finite element to increase the resolution of optimized designs. For example, consider the L-bracket design domain with a unit point load shown in Fig. 8a with 1600 finite elements; the material has unit Young's modulus and a Poisson's ratio of 0.3. Fig. 8b illustrates the optimized design at 50% volume fraction with 64 design variables per element.

As before, the number of inner iterations decreases with the direct method use. However, unlike the results from previous experiments, this also leads to a reduction in total computational time due to the increased design variables per



(a) With region restriction imposed per Fig. 6



(b) Without region restriction, each material $V_f = 12.5\%$

Fig. 7: Multi-material design.

element (independent of whether the volume constraint is imposed on the design variables or the filtered density variables). Further, the computational gain increases with increase in the number of design variables as summarized in Table 5.

4.7 Ease of implementation

Finally, we discuss the ease of implementation of the direct method, as a drop-in replacement to the bisection method. Popular codes employing the bisection method were modified to implement the direct method. These include the 88-line code [4], 99-line code [28], 169-line 3D code [21], both 2D and 3D versions of compliance-constrained volume minimization codes [2, 1], and PolyMat code [25]. The modifications are summarized in Appendix 7.

5 Conclusions

The bisection method is the *de facto* standard for design variable update in density-based topology optimization. It relies on computing the Lagrange multiplier up to a predefined accuracy. While the method is simple to implement, oscillations in the objective are typically observed. Some of the proposed improvements to the bisection method include (1) stricter convergence control at the cost of increased inner iterations, (2) heuristic updates on the bounds [2, 1], (3) computing the Lagrange multiplier bounds in each outer iteration, and (4) use of constraint approximation to avoid expensive filter application during inner iterations.

In this paper, we revisit the direct method of Lagrange multiplier computation as an alternate. It is shown that in the direct method, the constraint is always satisfied, and the objective oscillations are completely eliminated. Further, the

Table 4: Bisection versus direct method: Multiple constraint multi-material problem.

Problem	Final objective		Total design (inner iteration) time (s)		Outer (avg. inner) iterations	
	Bisection	Direct	Bisection	Direct	Bisection	Direct
With region constraint	304.176	304.176	103.74 (1.52)	101.08 (0.48)	248 (34)	248 (4.82)
Without region constraint	274.445	274.388	176.90 (7.58)	168.58 (2.15)	417 (34)	417 (4.89)

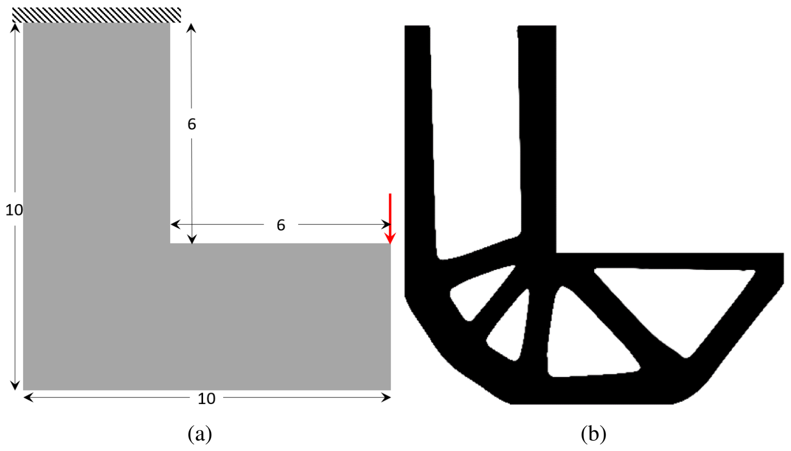


Fig. 8: Multi-resolution TO: (a) design problem, and (b) the final design variable distribution with 64 design variables per element, with constraint on the design variables.

Table 5: Multi-resolution TO: the direct method leads to a reduction in computational time.

# Design variable/element	Objective		Total design (inner iteration) time (s)		Outer (avg. inner) iterations	
	Bisection	Direct	Bisection	Direct	Bisection	Direct
4	198.405	198.406	18.30 (1.34)	17.14 (0.36)	405 (45.03)	406 (7.96)
64	198.396	198.396	106.30 (17.03)	95.57 (4.32)	500 (45.02)	500 (9.33)

direct method is insensitive to scaling of the Young's modulus and force values and hence scaling of the objective and constraint is not required.

Several numerical experiments were conducted to compare the two methods. The important conclusions were that the computed designs, and the final objective do not depend on the choice of the method. However, the direct method leads to smooth convergence of the objective, and reduces the design update time, i.e. the time spent in inner iterations. This improvement is usually insignificant except when the density filter is used or when the number of design variables are large as in multi-resolution topology optimization.

The direct method is applicable to any topology optimization problem with single active constraint including mutually exclusive multiple constraints and nonlinear constraints. Finally, to illustrate the simplicity of the direct method, drop-in implementations are provided for popular Matlab-based topology optimization codes. Although, only compliance-related problems were considered here, the direct method can be applied to other topology optimization problems as well.

6 Replication of results

Modifications in various open-source Matlab codes have been provided in Appendix to help readers reproduce the results. The complete set of Matlab code is available at `ersl.wisc.edu/software/DirectLagrangeMultiplier.zip`

Acknowledgments

The authors would like to thank the support of National Science Foundation through grant CMMI 1561899. Prof. Suresh is a consulting Chief Scientific Officer of SciArt, Corp.

Compliance with ethical standards

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

7 Appendix

Here, we summarize the required changes to popular topology optimization codes, in order to replace the bisection method with the direct method. These changes can be easily adapted to other codes such as fast topology optimization based on reanalysis and conjugate gradient solvers [2, 1].

7.1 Modifications to 99-line code

For the classic 99-line code (top.m) [28], all that is needed is to replace the `OC` function with the following:

```
39  function xnew=OC(nelx,nely,x,volfrac,dc)
40  change = true; move = 0.2; eta = 0.5;
41  varIn = true(nely,nelx);
42  xMax = min(x+move,1); xMin = ...
        max(x-move,0.001);
43  volToDistribute = volfrac*numel(x);
44  varTimesGrad = x.*(-dc).^eta;
45  while(change)
46  xnew = varTimesGrad/ ...
        (sum(varTimesGrad(varIn)) ...
        /volToDistribute);
47  volToDistribute = volfrac*nelx*nely ...
        -sum(xMax(xnew>=xMax)) ...
        -sum(xMin(xnew<=xMin));
48  change = ~...
        isequal(and(xnew<xMax,xnew>xMin),varIn);
49  varIn = and(xnew<xMax,xnew>xMin);
50  end
51  xnew(xnew>xMax) = xMax(xnew>xMax); ...
        xnew(xnew<xMin) = xMin(xnew<xMin);
```

7.2 Modifications to 88-line code

In the 88-line code (top88.m) [4], one must replace lines 70-80 with the following:

```
70  change1 = true;move = 0.2;eta = 0.5;
71  varIn = true(nelx*nely,1); x1 = x(:); ...
        dc = dc(:); dv = dv(:);
72  xMax = min(x1+move,1); xMin = ...
        max(x1-move,0);
73  gRem = volfrac*nelx*nely; gToDist = gRem;
74  xTimesR = x1.*((-dc./dv).^eta);
75  if ft == 1
76  while(change1)
77  xnew = xTimesR/ ...
        (sum(xTimesR(varIn))/gToDist);
78  upLgc = xnew>xMax;dnLgc = xnew<xMin;
79  gToDist = gRem - sum(xMax(upLgc)) - ...
        sum(xMin(dnLgc));
80  change1 = ~isequal(~varIn,(upLgc|dnLgc));
81  varIn = ~(upLgc|dnLgc);
82  end
83  xnew(upLgc) = xMax(upLgc);xnew(dnLgc) = ...
        xMin(dnLgc);
84  xPhys(:) = xnew(:);
85  elseif ft == 2
86  temp1 = zeros(nelx*nely,1);    temp2 = ...
        zeros(nelx*nely,1);
87  while(change1)
88  temp1(:) = 0;temp1(varIn) = xTimesR(varIn);
89  xnew = xTimesR/ ...
        (sum((H*temp1)./Hs)/gToDist);
90  upLgc = xnew>xMax;dnLgc = xnew<xMin;
91  temp2(:) = 0;temp2(upLgc) = ...
        xMax(upLgc);temp2(dnLgc) = xMin(dnLgc);
92  gToDist = gRem - sum((H*temp2)./Hs);
93  change1 = ~isequal(~varIn,(upLgc|dnLgc));
94  varIn = ~(upLgc|dnLgc);
95  end
96  xnew(upLgc) = xMax(upLgc);xnew(dnLgc) = ...
        xMin(dnLgc);
97  xPhys(:) = (H*xnew(:))./Hs;
98  end
```

To incorporate non-design regions, further modifications to the 88-line code is given below. A logical array named `passive` with *true* entry for every non-design element must be created. Then lines 79-81 in the modified 88-line code becomes,

```
79  gToDist = gRem - sum(xMax(upLgc)) ...
        -sum(xMin(dnLgc)) -sum(passive);
80  change1 = ~...
        isequal(~varIn,(upLgc|dnLgc|passive));
81  varIn = ~(upLgc|dnLgc|passive);
```

In addition, line 91 changes to

```
91  temp2(:) = 0;temp2(upLgc) = ...
        xMax(upLgc);temp2(dnLgc) = ...
        xMin(dnLgc);temp2(passive)=1;
```

Lines 93-94 must be modified identical to the modifications in line 80-81. Finally, `xnew(passive) = 1;` should be appended to line 83 and 96 so that they read:

```
83  xnew(upLgc) = xMax(upLgc);xnew(dnLgc) = ...
        xMin(dnLgc); xnew(passive) = 1;
```

7.3 Modifications to 3D code

In the 3D code (top3d.m) [21], one must replace lines 82-88 with the following

```
82 change1 = true;move = 0.2;eta = ...
       0.5;ocIter = 0;
83 varIn = true(nele,1);
84 xMax = min(x+move,1);xMin = max(x-move,0);
85 gRem = volfrac*nele; gToDist = gRem;
86 xTimesGrad = x.*((-dc./dv).^eta);
87 temp1 = zeros(nele,1);    temp2 = ...
       zeros(nele,1);
88 while(change1)
89 temp1(:) = 0;temp1(varIn) = ...
       xTimesGrad(varIn);
90 xnew = ...
       xTimesGrad/(sum((H*temp1)./Hs)/gToDist);
91 upLgc = xnew>xMax;dnLgc = xnew<xMin;
92 temp2(:) = 0;temp2(upLgc) = ...
       xMax(upLgc);temp2(dnLgc) = xMin(dnLgc);
93 gToDist = gRem - sum((H*temp2)./Hs);
94 change1 = ~isequal(~varIn,(upLgc|dnLgc));
95 varIn = ~(upLgc|dnLgc);ocIter = ocIter ...
       + 1;
96 end
97 xnew(upLgc) = xMax(upLgc);xnew(dnLgc) = ...
       xMin(dnLgc);
98 xPhys(:) = (H*xnew(:))./Hs;
```

Further, to update the display in every iteration, line 6 (i.e. the `displayflag`) must be removed, and line 94 in the original code must be replaced with `volshow(xPhys)`.

7.4 Modifications for nonlinear constraint

The following code can replace the bisection method in any volume-minimization code [2, 1] available at `github.com/odedamir/topopt-mgcg-matlab`. For instance in the code `minV.m`, replace lines 83-96 with the lines below:

```
change1 = true;eta = 0.5;ocIter = ...
    0;move = 0.2;
dc = dc(:);x1 = x(:);dcx1 = dc.*x1;
R = ((-dc./dv(:)).^eta);gRem = ...
    (compconst - comp);
xMax = min(x1+move,1);xMin = ...
    max(x1-move,1e-10);
varIn = true(nelx*nely,1);
```

Then, depending on the approximation used, the following changes are sufficient:

Linear approximation :

```
gToDist = gRem+sum(dcx1);
cUps = dc.*(xMax-x1); cDns = dc.*(xMin-x1);
while(change1)
xnew = x1.*R*gToDist/ sum(dcx1.*R.*varIn);
upLgc=xnew>xMax;dnLgc=xnew<xMin;
change1 = ~isequal(~(upLgc|dnLgc),varIn);
varIn = ~(upLgc|dnLgc); ocIter = ocIter ...
    + 1;
gToDist=gRem -sum(cUps(upLgc)) ...
    -sum(cDns(dnLgc)) + sum(dcx1(varIn));
if (gToDist>=0||ocIter>15)
if move<1
move = min(move*2,1);ocIter = 0;
xMax = min(x1+move,1); xMin = ...
    max(x1-move,1e-10);
gToDist = gRem+sum(dcx1);varIn = ...
    true(nelx*nely,1);
cUps = dc.*(xMax-x1); cDns = dc.*(xMin-x1);
else, break;  end
end
end
xnew(xnew>xMax) = xMax(xnew>xMax); ...
    xnew(xnew<xMin) = xMin(xnew<xMin);
```

Reciprocal approximation :

```
gToDist = sum(dcx1)-gRem;
cUps = dcx1.*(1-(x1./xMax)); cDns = ...
    dcx1.*(1-(x1./xMin));
while(change1)
xnew = x1.*R*sum(varIn.*dcx1./R)/gToDist;
upLgc=xnew>xMax;dnLgc=xnew<xMin;
change1 = ~isequal(~(upLgc|dnLgc),varIn);
varIn = ~(upLgc|dnLgc); ocIter = ocIter ...
    + 1;
gToDist = -gRem +sum(cUps(upLgc)) ...
    +sum(cDns(dnLgc)) +sum(dcx1(varIn));
if (gToDist>=0||ocIter>15)
if move<1
move = min(move*2,1);ocIter = 0;
xMax = min(x1+move,1); xMin = ...
    max(x1-move,1e-10);
gToDist = -gRem+sum(dcx1);varIn = ...
    true(nelx*nely,1);
cUps = dcx1.*(1-(x1./xMax)); cDns = ...
    dcx1.*(1-(x1./xMin));
else, break;  end
end
end
xnew(xnew>xMax) = xMax(xnew>xMax); ...
    xnew(xnew<xMin) = xMin(xnew<xMin);
```

Exponential approximation :

```
alpha = -3;xDcS2 = (dcx1.*R.^alpha)/alpha;
gToDist = gRem + sum(dcx1)/alpha;
cUps = dcx1.*(((xMax./x1).^alpha)-1)/alpha;
cDns = dcx1.*(((xMin./x1).^alpha)-1)/alpha;
while(change1)
xnew = x1.*R ...
    *(gToDist/sum(xDcS2(varIn))) ...
    .^(1/alpha);
upLgc=xnew>xMax;dnLgc=xnew<xMin;
change1 = ~isequal(~(upLgc|dnLgc),varIn);
varIn = ~(upLgc|dnLgc); ocIter = ocIter ...
    + 1;
gToDist = gRem - sum(cUps(upLgc)) - ...
    sum(cDns(dnLgc)) + ...
    sum(dcx1(varIn))/alpha;
if (gToDist<=0||ocIter>15)
if move<1
move = min(move*2,1);ocIter = 0;
```

```
xMax = min(x1+move,1); xMin = ...
    max(x1-move,1e-10);
gToDist = gRem + sum(dcx1)/alpha; varIn ...
    = true(nelx*nely,1);
cUps = dcx1.*(((xMax./x1).^alpha)-1)/alpha;
cDns = dcx1.*(((xMin./x1).^alpha)-1)/alpha;
else, break;  end
end
end
xnew(xnew>xMax) = xMax(xnew>xMax); ...
    xnew(xnew<xMin) = xMin(xnew<xMin);
```

7.5 Modifications to PolyMat

Finally, for the PolyMat code [25], the function `UpdateScheme` must be replaced with the following.

```
function [zNew,Change] = ...
    UpdateScheme(dfdz,g,dgdz,z0,V0,opt)
nelem = size(dfdz,1); nmat = size(dfdz,2);
dfdz = reshape(dfdz,nelem*nmat,1); dgdz ...
    = reshape(dgdz,nelem*nmat,1);
z0 = reshape(z0,nelem*nmat,1); V0 = ...
    reshape(V0,nelem*nmat,1);
zMin=opt.zMin; zMax=opt.zMax;
move=opt.ZPRMove*(zMax-zMin); ...
    eta=opt.ZPREta;
change = true; ocIter = 0;  gRem = ...
    sum(dgdz.*z0) - g;
zCnd = zMin + (V0-zMin) ...
    .*max(0,-(dfdz./dgdz)).^eta;
zCndGrad = zCnd.*dgdz;
while (change)
varIn = true(nelem*nmat,1);  gToDist = ...
    gRem;
zMax1 = min(z0+move,zMax);  zMin1 = ...
    max(z0-move,zMin);
zUpGrad = zMax1.*dgdz;  zDnGrad = ...
    zMin1.*dgdz;
while(change & rem(ocIter+1,15) ~= 0 & ...
    gToDist>=0)
zNew = zCnd*gToDist/ sum(zCndGrad(varIn));
upLgc=zNew>zMax1; dnLgc=zNew<zMin1;
gToDist = gRem - sum(zUpGrad(upLgc)) - ...
    sum(zDnGrad(dnLgc));
change = ~isequal(~varIn,(upLgc|dnLgc));
varIn = ~(upLgc|dnLgc);ocIter = ocIter+1;
end
if move<1, move = min(1,move*1.5); ...
    else, break; end
end
zNew(upLgc) = zMax1(upLgc);
zNew(dnLgc) = zMin1(dnLgc);
Change = max(abs(zNew-z0))/(zMax-zMin);
zNew = reshape(zNew,nelem,nmat,1);
```

References

1. Amir, O.: Revisiting approximate reanalysis in topology optimization: on the advantages of recycled preconditioning in a minimum weight procedure. Structural and Multidisciplinary Optimization **51**(1), 41–57 (2015)
2. Amir, O., Aage, N., Lazarov, B.S.: On multigrid-CG for efficient topology optimization. Structural and Multidisciplinary Optimization **49**(5), 815–829 (2014)
3. Ananiev, S.: On Equivalence Between Optimality Criteria and Projected Gradient Methods with Application to Topology Optimization Problem. Multibody System Dynamics **13**, 25–38 (2005)
4. 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)
5. Arora, J.S., Chahande, A.I., Paeng, J.K.: Multiplier methods for engineering optimization. International Journal for Numerical Methods in Engineering **32**(7), 1485–1525 (1991)
6. Bendsøe, M.P.: Optimal shape design as a material distribution problem. Structural Optimization **1**(4), 193–202 (1989)
7. Bendsøe, M.P., Sigmund, O.: Material interpolation schemes in topology optimization. Archive of Applied Mechanics **69**(9-10), 635–654 (1999)
8. Bendsøe, M.P., Sigmund, O.: Topology optimization. Springer Berlin Heidelberg (2004)
9. Berke, L., Khot, N.S.: Structural Optimization Using Optimality Criteria. In: Computer Aided Optimal Design: Structural and Mechanical Systems, pp. 271–311 (1987)
10. Chandrasekhar, A., Kumar, T., Suresh, K.: Build optimization of fiber-reinforced additively manufactured components. Structural and Multidisciplinary Optimization **61**(1), 77–90 (2020)
11. Deng, J., Chen, W.: Concurrent topology optimization of multiscale structures with multiple porous materials under random field loading uncertainty. Structural and Multidisciplinary Optimization **56**(1), 1–19 (2017)
12. Fadel, G.M., Riley, M.F., Barthelemy, J.M.: Two point exponential approximation method for structural optimization. Structural Optimization **2**(2), 117–124 (1990)
13. Fleury, C.: Structural weight optimization by dual methods of convex programming. International Journal for Numerical Methods in Engineering **14**(12), 1761–1783 (1979)
14. Fleury, C., Braibant, V.: Structural optimization: A new dual method using mixed variables. International Journal for Numerical Methods in Engineering **23**(3), 409–428 (1986)
15. Groenwold, A.A., Etman, L.F.: On the equivalence of optimality criterion and sequential approximate optimization methods in the classical topology layout problem. International Journal for Numerical Methods in Engineering **73**(3), 297–316 (2008)
16. Haftka, R.T., Gürdal, Z.: Elements of Structural Optimization. Third revised and expanded edition, vol. 1, third rev. edn. SPRINGER-SCIENCE+BUSINESS MEDIA, B.V. (1992)
17. Kharmanda, G., Olhoff, N., Mohamed, A., Lemaire, M.: Reliability-based topology optimization. Structural and Multidisciplinary Optimization **26**(5), 295–307 (2004)
18. Khot, N.S., Venkayya, V.B., Berke, L.: Optimum structural design with stability constraints. International Journal for Numerical Methods in Engineering **10**(5), 1097–1114 (1976)
19. Kiyono, C.Y., Vatanabe, S.L., Silva, E.C., Reddy, J.N.: A new multi-p-norm formulation approach for stress-based topology optimization design. Composite Structures **156**, 10–19 (2016)
20. Kumar, T., Suresh, K.: A density-and-strain-based K-clustering approach to microstructural topology optimization. Structural and Multidisciplinary Optimization **61**(4), 1399–1415 (2020)
21. Liu, K., Tovar, A.: An efficient 3D topology optimization code written in Matlab. Structural and Multidisciplinary Optimization **50**(6), 1175–1196 (2014)
22. Nguyen, T.H., Paulino, G.H., Song, J., Le, C.H.: A computational paradigm for multiresolution topology optimization (MTOP). Structural and Multidisciplinary Optimization **41**(4), 525–539 (2010)

23. Qian, X.: Topology optimization in B-spline space. Computer Methods in Applied Mechanics and Engineering **265**, 15–35 (2013)
24. Sanders, E.D., Aguiló, M.A., Paulino, G.H.: Multi-material continuum topology optimization with arbitrary volume and mass constraints. Computer Methods in Applied Mechanics and Engineering **340**, 798–823 (2018)
25. Sanders, E.D., Pereira, A., Aguiló, M.A., Paulino, G.H.: PolyMat: an efficient Matlab code for multi-material topology optimization. Structural and Multidisciplinary Optimization **58**(6), 2727–2759 (2018)
26. Schmit, L.A., Fleury, C.: Structural synthesis by combining approximation concepts and dual methods. AIAA Journal **18**(10), 1252–1260 (1980)
27. Sethian, J.A., Wiegmann, A.: Structural Boundary Design via Level Set and Immersed Interface Methods. Journal of Computational Physics **163**(2), 489–528 (2000)
28. Sigmund, O.: A 99 line topology optimization code written in matlab. Structural and Multidisciplinary Optimization **21**(2), 120–127 (2001)
29. Stolpe, M., Svanberg, K.: An alternative interpolation scheme for minimum compliance topology optimization. Structural and Multidisciplinary Optimization **22**(2), 116–124 (2001)
30. Sui, Y., Yi, G.: A discussion about choosing an objective function and constraint conditions in structural topology optimization. In: 10th World Congress on Structural and Multidisciplinary Optimization (2013)
31. Suresh, K.: A 199-line Matlab code for Pareto-optimal tracing in topology optimization. Structural and Multidisciplinary Optimization **42**(5), 665–679 (2010)
32. Svanberg, K.: The method of moving asymptotes-a new method for structural optimization. International Journal for Numerical Methods in Engineering **24**(2), 359–373 (1987)
33. Talischi, C., Paulino, G.H., Pereira, A., M Menezes, I.F., M Menezes, I.F.: PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Structural and Multidisciplinary Optimization **45**, 329–357 (2012)
34. Tavakoli, R., Mohseni, S.M.: Alternating active-phase algorithm for multimaterial topology optimization problems: A 115-line MATLAB implementation. Structural and Multidisciplinary Optimization **49**(4), 621–642 (2014)
35. Venkayya, V.B.: Optimality criteria: A basis for multidisciplinary design optimization. Computational Mechanics **5**(1), 1–21 (1989)
36. Venkayya, V.B., Khot, N.S., Berke, L.: Application of optimality criteria approaches to automated design of large practical structures. In: Second symposium on structural optimization, pp. 1–19 (1973)
37. Wang, M., Wang, X., Guo, D.: A level set method for structural topology optimization. Computer Methods in Applied Mechanics and Engineering **192**(1-2), 227–246 (2003)
38. Watts, S., Arrighi, W., Kudo, ·J., Tortorelli, D.A., White, D.A., Kudo, J., Tortorelli, D.A., White, D.A.: Simple, accurate surrogate models of the elastic response of three-dimensional open truss micro-architectures with applications to multiscale topology design. Structural and Multidisciplinary Optimization (2019)
39. Xia, L., Breitkopf, P.: Design of materials using topology optimization and energy-based homogenization approach in Matlab. Structural and Multidisciplinary Optimization **52**(6), 1229–1241 (2015)
40. Xie, Y.M., Steven, G.P.: A simple evolutionary procedure for structural optimization. Computers and Structures **49**(5), 885–896 (1993)
41. Yin, L., Yang, W.: Optimality criteria method for topology optimization under multiple constraints. Computers and Structures **79**(20-21), 1839–1850 (2001)
42. Zuo, W., Saitou, K.: Multi-material topology optimization using ordered SIMP interpolation. Structural and Multidisciplinary Optimization **55**(2), 477–491 (2017)