Optimization and Engineering (2024) 25:575–603
https://doi.org/10.1007/s11081-023-09808-w

SOFTWARE ARTICLE

Developing an efficient coupled function-based topology optimization code for designing lightweight compliant structures using the BESO algorithm

Mohsen Teimouri[1] · **Masoud Asgari**[1]

Received: 15 February 2023 / Revised: 2 May 2023 / Accepted: 3 May 2023 /
Published online: 17 June 2023

Abstract

In this article, an engineering computational method is developed for implementing the Bi-directional Evolutionary Structural Optimization algorithm by creating an interface between MATLAB and ABAQUS software packages. This interface is a topology optimization (TO) package developed for maximizing the stiffness of 2D structures subjected to a volume constraint. A source code (the main function), including four dependent functions, is generated as the optimization engine. It covers the optimization main algorithm loop and its pre-loop requirements. The dependent functions are responsible for checking background finite element analyses (FEA) in ABAQUS, obtaining necessary parameters from FEA results, conducting the sensitivity analysis and the filtering scheme, and finally, defining an updated design area for a new optimization loop. At the end of the source code, the optimality criterion is checked and results are displayed when it is satisfied. The user can define the geometry of the design area, mesh size, and boundary conditions in the ABAQUS CAE environment while no finite element codes are needed to be written. This package is developed for quadratic plane stress elements in general static problems. To validate and verify its performance, different benchmark strain energy problems are solved as examples, and results are compared to previous well-known research efforts. This TO package can be developed for frequency, non-linear, multi-material, and periodic problems by updating its sensitivity analysis section. Its 3D version can be also achieved by modifying the elemental and nodal matrices for increased degrees of freedom.

Keywords Topology optimization · FEM · MATLAB code · ABAQUS · BESO method · Educational software

✉ Masoud Asgari
asgari@kntu.ac.ir

[1] Research Laboratory of Passive Safety Systems, Faculty of Mechanical Engineering, K. N. Toosi University of Technology, P. O. Box, Tehran 19395-1999, Iran

List of symbols

C	Compliance or strain energy
$\boldsymbol{F}$	Global load vector
$\boldsymbol{K}$	Global stiffness matrix
$\boldsymbol{U}$	Displacement vector
Ne	The total number of elements in the design space
x_j	Design variable
V_j	jTh element volume
ER	Evolutionary ratio.
V_0	Final structure volume
$xmin$	Minimum element density
E_1	Solid material's Young modulus
p	Penalization parameter
$\boldsymbol{K}_{\boldsymbol{J}}^{1}$	Stiffness matrix of the solid material
$\boldsymbol{u_e}$	Elemental displacement vector
α_e	Elemental sensitivity number
w	Weight function
r_{ej}	Central distance between elements e and j
r_{min}	Filtering radius

1 Introduction

Michell's (1904) work on the layout optimization of trusses can be considered the first investigation of topology optimization (TO). Before the emergence of modern TO algorithms, many efforts have been made concerning material distribution optimization in 2D and 3D design areas (Bendsøe et al. 1994, 1995; Asgari 2015, 2016). Since then and mainly over the past decades, with the emergence of various methods including density-based methods (Bendsøe 1989; Zhou and Rozvany 1991), evolutionary procedures (Xie and Steven 1992, 1993), bubble method (Eschenauer et al. 1994), topological derivative (Sokolowski and Zochowski 1999), level-set methods (Sethian and Wiegmann 2000; Allaire et al. 2002; Wang et al. 2003; Xia et al. 2006, 2011, 2012, 2014; Luo et al. 2008; Xia and Shi 2016a, 2016b) and phase-field method (Bourdin and Chambolle 2003), TO has gone under serious changes and developments in both academic and industrial disciplines. The single-material minimum compliance problem subjected to a volume constraint is the most considered design case in this area. However, its borders have gone beyond single material static problems, and novel subjects including non-homogenous and multi-material cases (Banh et al. 2023, 2021a, 2021b; Ngoc et al. 2022; Doan and Lee 2017), non-linear studies (Meng 2022), fuzzy and probabilistic-based TO models (Meng et al. 2020, 2021), etc. have been studied by researchers.

The evolutionary structural optimization (ESO) method proposed by Xie and Steven (1992, 1993), is a heuristic TO method based on the simple concept that a structure's

topology is optimal when low-stress areas are gradually removed from. This simple concept behind the ESO method makes it a user-friendly one for researchers and students. Its extended version, known as the bi-directional evolutionary structural optimization (BESO) method, was developed by Querin et al. (1998, 2000a, 2000b). In this method, adding elements to highly stressed areas became possible as well as their removal from low-stress regions. The next prominent development in the ESO method was presented by Huang and Xie (2007a) which was an improved version of the BESO method. This update has shown a promising performance in a wide range of structural design studies including stiffness and frequency optimizations (Teimouri and Asgari 2019, 2020; Huang et al. 2010), nonlinear material and large deformation problems (Huang and Xie 2007b, 2008a), energy absorption (Huang et al. 2007), multiple materials (Huang and Xie 2009), multiple constraints (Huang and Xie 2010a), periodic structures (Huang and Xie 2008b), hybrid structures (Dong et al. 2020; Teimouri et al. 2021), and so on (Huang and Xie 2010a). Apart from being very easy to understand and implement (at least for the compliance minimization case), the primary motivation for the evolutionary approaches seems to be that mathematically based or continuous variable approaches "involve some complex calculus operations and mathematical programming" (Li et al. 1999). They contain "mathematical methods of some complexity" (Zhao et al. 1998), whereas the evolutionary approach "takes advantage of powerful computing technology and intuitive concepts of evolution processes in nature" (Li et al. 1999). Through years of development, the current form of BESO has become a gradient-based mathematical optimization method with convergent and mesh-independent algorithms (Zuo and Xie 2015).

Implementation of the TO algorithms in different engineering problems has been the subject of many studies, and researchers, either academic or industrial have published several codes and programming tools for this purpose. Meanwhile, these codes have been helpful for new users to understand the underlying idea of structural topology optimization (Zhu et al. 2021). In the review and educational articles of (Zhu et al. 2020, 2021), these optimization programs are discussed in detail. In Table 1 (Han et al. 2021), educational computer programs for continuum topology optimization problems are gathered from different articles.

Most of the presented codes are written by solving the simple mean compliance problem; one can easily apply them to solve different compliance problems. Among other published codes for different algorithms, there have been several efforts to implement the ESO and BESO methods including soft-kill and SERA for 2D problems, and PYTHON3D for 3D problems. In a comprehensive review of the BESO method on advanced structures and materials, MATLAB codes esoL and esoX can be referred to Xia et al. (2018). The BESO method, with its discrete nature, can overcome numerical instabilities such as checkerboarding, mesh dependency, and intermediate material densities. Yongsheng Han's article (Han et al. 2021) introduces an efficient 137-line MATLAB code for geometrically nonlinear compliance topology optimization problems using the BESO method. These codes have also smoothed the way for many commercial TO software packages like TOSCA, n-Topology, Altair Inspire, and so on, which have been used by researchers in many articles (Teimouri and Asgari 2021). Among the published programs, few of them have focused on using well-known FEM packages in their optimization code. It seems that optimization codes equipped with

Table 1 Educational computer programs for topology optimization (Dong et al. 2020)

Program	Author	References	Environment	Method	Assumption
99-line	Sigmund 2001	Sigmund 2001)	MATLAB	SIMP	Linear
199-line	Wang et al	Gao et al. 2021)	MATLAB	LSM	Linear
SOFT-KILL	Huang and Xie 2010a, 2010b	Huang and Xie 2010a; Huang and Xie 2010b)	MATLAB	BESO	Linear
dLSM	Challis 2010	Challis 2010)	MATLAB	LSM	Linear
88-line	Andreassen et al. 2011	Andreassen et al. 2011)	MATLAB	SIMP	Linear
PolyTop	Talischi et al. 2012	Talischi et al. 2012)	MATLAB	SIMP	Linear
169-line 3D	Liu and Tovar 2014	Liu and Tovar 2014)	MATLAB	SIMP	Linear
115-line	Tavakoli and Mohseni 2014	Tavakoli and Mohseni 2014)	MATLAB	SIMP	Linear
PYTHON 3D	Zuo and Xie 2015	Zuo and Xie 2015)	Python, Abaqus	ESO	Linear
topX	Xia and Breitkopf 2015	Xia and Breitkopf 2015)	MATLAB	BESO	Linear
MMC188	Zhang et al. 2016	Zhang et al. 2016)	MATLAB	MMC	Linear
SERA	Loyola et al. 2018	Ansola Loyola et al. 2018)	MATLAB	ESO	Linear
FEniCS	Laurain 2018	Laurain 2018)	FEniCS	LSM	Linear
88-line	Wei et al. 2018	Wei et al. 2018)	MATLAB	LSM	Linear
esoL esoX	Xia et al. 2018	Xia et al. 2018)	MATLAB	BESO	Linear
185-line	Laurain 2018	Laurain 2018)	FEniCS	LSM	Linear
213-line	Chen et al. 2019	Chen et al. 2019)	MATLAB, ANSYS	SIMP	Nonlinear
AC# 3D	Lagaros et al. 2019	Lagaros et al. 2019)	SAP2000	SIMP	Linear
128-line	Liang and Cheng 2020	Liang and Cheng 2020)	MATLAB	CDT	Linear
108-line	Kim et al. 2020	Salgarello et al. 2013)	FreeFEM	LSM	Linear
62-line	Yaghmaei et al. 2020	Yaghmaei et al. 2020)	MATLAB	LSM	Linear
89-line	Zhu et al. 2020b	Zhu et al. 2021)	FreeFEM	SIMP	Nonlinear
Top99neo	Ferrari and Sigmund 2020	Ferrari and Sigmund 2020)	MATLAB	SIMP	Linear
Top3D125	Ferrari and Sigmund 2020	Ferrari and Sigmund 2020)	MATLAB	SIMP	Linear
Tobs101	Picelli et al. 2020	Picelli et al. 2021)	MATLAB	TOBS	Linear

this feature can be more successful in solving versatile structural problems. Moreover, higher calculation speed and more precise and valid outputs are expected while the FEA part is done by an efficient powerful software like ANSYS or ABAQUS. This feature makes the designer more convenient to focus on the code and optimization process rather than solving a complicated FEM problem for a specific geometry or mesh size.

Being aware of the above-mentioned research efforts' pros and cons, and also considering the FE modeling complexity of different published codes, this work is aimed at developing a multipurpose efficient optimization software interface that could be easily understood, used, and developed by students and researchers all around the world.

In this work, a BESO-based MATLAB-ABAQUS interface is presented for solving 2D compliance problems. This interface includes MATLAB functions generated based on the BESO optimization steps, which are capable of reading the ABAQUS FEA results file, exporting and analyzing its required data, updating the design area, and finally, writing a new FE input file in each iteration.

In Sect. 2, the BESO procedure is briefly discussed. The detailed program lines written in MATLAB are presented in Sect. 3. In Sect. 4, some examples are presented and compared to other programs' results to validate the presented software package. Finally, conclusions are made in Sect. 5.

2 The BESO algorithm formulation implemented in MATLAB

2.1 Problem statement

The solid-void minimum compliance or maximum stiffness topology optimization problem with a volume constraint is considered as follows:

$$\text{Minimize} : C = \tfrac{1}{2}\boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{U} \tag{1a}$$

$$\text{Subject to} : \sum_{j=1}^{Ne} x_j V_j \leq V_0 \tag{1b}$$

$$\boldsymbol{F} = \boldsymbol{K}\boldsymbol{U} \tag{1c}$$

$$\text{Design variable} : x_j = (0 \; or \; xmin) or \; 1 \tag{1d}$$

where C is the total strain energy (the objective function), $\boldsymbol{F}$ and $\boldsymbol{U}$ are global load and displacement vectors, $\boldsymbol{K}$ is the global stiffness matrix, $\mathbf{x}$ is elements' densities vector representing their presence or absence in the design space, V_j and V_0 indicate the jth element's volume and the prescribed volume of the final structure respectively, and N_e is the total number of elements in the design space.

2.2 The BESO algorithm

Applying the material interpolation scheme of the SIMP method in the BESO algorithm, a solid-void pattern will be assumed. This scheme is considered as follows:

$$E(x_j) = E_1 x_j^p \tag{2a}$$

$$\boldsymbol{K} = \sum_{j=1}^{N_e} x_j^p \boldsymbol{K}_{\boldsymbol{j}}^{1} \tag{2b}$$

In Eq. (2a), the elastic modulus of each element is defined as a function of the element density and solid material's Young modulus E_1, and p is the penalization parameter equal to 3. In Eq. (2b), the global stiffness matrix, $\boldsymbol{K}$, depends on the jth element density and the stiffness matrix of the solid material $\boldsymbol{K}_{\boldsymbol{J}}^{1}$.

The BESO method applies a gradient-based strategy in the sensitivity analysis step to update the design domain. So, the design variables are updated based on the elemental sensitivity number α_e as follows:

$$\alpha_e = \frac{\partial C}{\partial x_e} \tag{3a}$$

$$\alpha_e = -p x_e^{p-1} \boldsymbol{u}_{\boldsymbol{e}}^{\boldsymbol{T}} \boldsymbol{k}_1 \boldsymbol{u}_{\boldsymbol{e}} \tag{3b}$$

In Eq. (3b), $\boldsymbol{u}_{\boldsymbol{e}}$ represents the elemental displacement vector.

To overcome the mesh-dependency and checker-boarding of the design domain, a smoothing approach is applied to the sensitivity numbers of elements as follows:

$$\widehat{\alpha_e} = \frac{\sum_j w(r_{ej}) \alpha_j}{\sum_j w(r_{ej})} \tag{4a}$$

$$w(r_{ej}) = \max(0, r_{min} - r_{ej}) \tag{4b}$$

where r_{ej} is the distance between elements e and j centroids, w indicates the weight function of each element, and r_{min} is the filtering radius. To achieve convergent solutions, the BESO method uses average elemental sensitivity numbers that bring the history of the procedure into account (Eq. 5).

$$\widehat{\alpha_e} = \frac{\widehat{\alpha_e^k} + \widehat{\alpha_e^{k-1}}}{2} \tag{5}$$

For detailed information, one can refer to the book, "Evolutionary topology optimization of continuum structures, Methods and Applications", proposed by Huang and Xie (2010b).

After modifying elemental sensitivity numbers (Eq. 5), they will be sorted. The criterion used for elements addition/deletion in the BESO method is defined as the

sensitivity numbers of solid elements ($x_j = 1$) are always higher than those of soft elements ($x_j = 0/xmin$). So, depending on the elements removal approach (soft-kill or hard-kill), those with lower sensitivity numbers will be removed or labeled with *xmin* (Huang and Xie 2010b). The number of removed or *xmin*-labeled elements is determined based on the volume constraint evolutionary ratio (ER) at each step k (Eq. 6).

$$V_{k+1} = V_k(1 \pm ER)(k = 1, 2, 3, \ldots) \tag{6}$$

The add/remove procedure of elements and FEA continues in a repetitive loop until the optimization constraint is reached and the objective function is converged. Convergence of the objective function is calculated based on Eq. 7.

$$error = \frac{\left|\sum_{i=1}^{N} C_{k-i+1} - \sum_{i=1}^{N} C_{k-N-i+1}\right|}{\sum_{i=1}^{N} C_{k-i+1}} \leq \tau \tag{7}$$

The iterative procedure of the main BESO method and its implemented version in this article (the MATLAB-ABAQUS interface) is presented in Figs. 1 and 2.

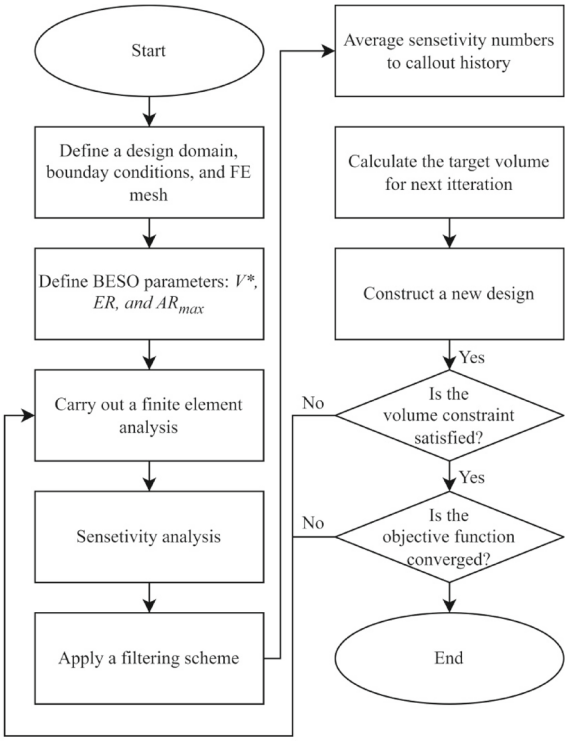

Fig. 1 Flowchart of the BESO algorithm (Huang and Xie 2010b)

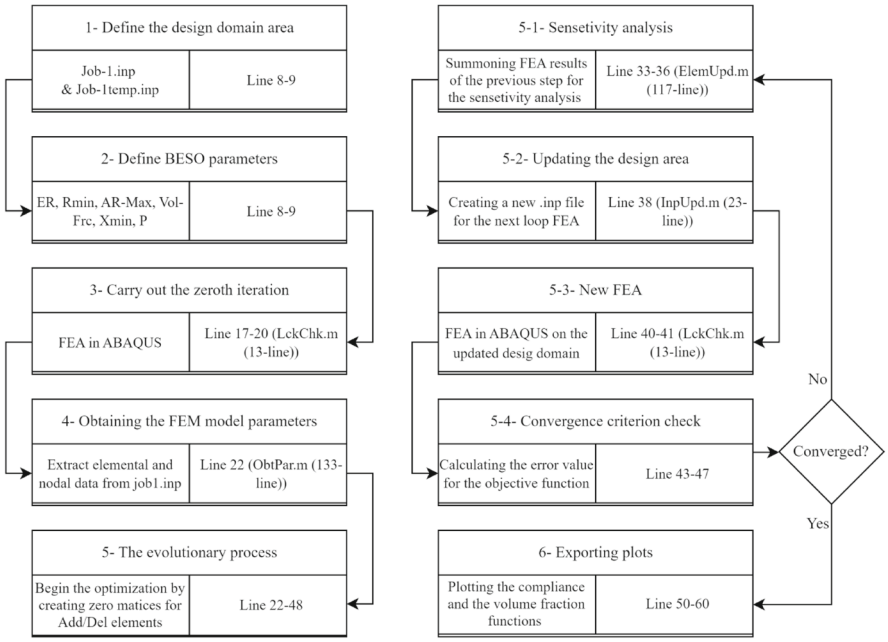

Fig. 2 Flowchart of the FEM-based BESO developed in the MATLAB-ABAQUS interface

3 MATLAB-ABAQUS interface implementation

In the following, the main optimization code (engine/source) and its functions will be explained in detail. The source code (BESO m-file) consists of 6 sections that take the user through the optimization process. These sections are presented from 3.1 to 3.6.

Before the main code, all texts from the Command Window are cleared; resulting in a clear screen; and then, an arbitrary working directory is defined by the user (lines 1–5). All MATLAB files, FEA input and output files, and optimization results will be created and saved in this directory. This directory will be called by *the System* command in line 6.

All codes can be found in Appendix A.

3.1 Define a design domain (lines 7–8)

The MATLAB-ABAQUS TOP interface is developed for two-dimensional problems with linear plane stress elements. The initial design domain, including geometry, loading, boundary conditions, linear plane stress elements, quad meshes (CPS4R), and a static analysis step need to be defined in the ABAQUS CAE environment. Then, a.inp job file should be exported from the ABAQUS job module to the predefined directory where MATLAB m-files exist. In this study, the exported input file is named *Job-1*. It should be opened after creation by a text editing tool of Windows, and the following texts should be added/replaced after the "** OUTPUT REQUESTS" section:

```
** OUTPUT REQUESTS
** FIELD OUTPUT: F-Output-1
**
*Output, field
*Node Output
U,
*Element Output, directions=YES
S,
*Output, history, frequency=0
*EL PRINT, POSITION=CENTROIDAL, SUMMARY=YES
S,
ELSE,
COORD,
EVOL,
*End Step
```

These changes define the outputs, including nodal displacement and coordinates, and elemental stress, strain energy, and volume values, that all are needed in a compliance optimization problem. This file data will be needed to run the zeroth iteration FEA (Sect. 3.3) on the initial design domain and extract the model parameters (elemental and nodal data of the initial design domain (Sect. 3.4)), which will be required in the sensitivity analysis.

As the optimization process begins, a dynamic ABAQUS input file, the same as the initial design domain, will be needed to be called for creating the next iteration input file (updated design domain). It should be noted that this file should include the deleted or soft elements to result in an updated design area. So, a copy of the created.inp file should be created in the same folder named *Job-1temp* with the following modifications made to it:

If a hard-kill approach is desired, the "*Elset, elset=sh_del_elm, instance=Part-1-1" string, should be added right before the "*End Assembly" line:

```
*Elset, elset=sh_del_elm, instance=Part-1-1
*End Assembly
```

The above-mentioned line defines an element set named *sh_del_elm* that includes the list of elements that should be deleted at each step. This elemental set will be removed from the design area based on the following modifications to the file:

```
** Interaction: Remove_Elements
*Model Change, remove
sh_del_elm,
** OUTPUT REQUESTS
```

If a soft-kill approach is desired, instead of the permanent deletion of elements, their initial material property should be replaced by a soft material's property. For this purpose, the following modifications in the *Job-1temp* file should be applied:

Step 1: Soft elements (equivalent to deleted elements of the hard-kill procedure) should be defined as an element set as follows:

```
*Nset, nset=Set-1, generate
  1,  9881,    1
*Elset, elset=Set-1, generate
  1,  9600,    1
*Elset, elset=sh_del_elm
```

Step 2: Mechanical properties of the soft material (Material-2), calculated based on Eq. 2a, should be added after the "**MATERIALS" headline as follows:

```
** MATERIALS
**
*Material, name=Material-1
*Elastic
200000, 0.3
*Material, name=Material-2
*Elastic
2e-4, 0.3
```

Step 3: To assign the Material-2 properties to the soft elements (*elset=sh_del_elm*), another *section* definition is needed in the ABAQUS Material property module. This section is defined as follows:

```
** Section: Section-1
*Solid Section, elset=_I1, material=Material-1
1,
** Section: Section-2
*Solid Section, elset=sh_del_elm, material=Material-2
1,
```

After applying the above-mentioned modifications, the dynamic ABAQUS input file is ready to be used during the optimization iterations. This.inp file will be summoned and read line by line to write a new input file as an updated design area (Sect. 3.5.2).

It should be noted that to use any other type of element (like 3D cubic element which is of more interest to researchers), the smoothing step of the BESO algorithm needs to be modified for 3D elements and so, the sensitivity analysis. Consequently, modifications based on a 3D element's equation order and the number of nodes should be applied to all necessary matrices' sizes in the MATLAB functions.

3.2 Define the BESO parameters (lines 9–12)

In Sect. 2, the BESO initial parameters such as evolutionary ratio (*ER*), maximum addition ratio (*AR_Max*), minimum filtering radius (*Rmin*), volume fraction (*Vol_Frc*),

minimum design variable (x_{min}), and penalization parameter (p) values are defined and entered by the user (the last two are only defined in case of using a soft kill approach). These parameters are constant during the optimization iterative process.

3.3 Zeroth iteration (lines 13–15)

In the third section, ABAQUS is called to run the first Finite Element (FE) simulation (the zeroth iteration) on the initial design domain (*Job-1.inp*). For this purpose, the *system* command is used to summon ABAQUS for running the predefined *Job-1.inp* file. Its job output is named *Iteration_0*, and its data will be used in the optimization evolutionary process initiation (5th section). A function is created named as **LckChk.m** to check the appearance and disappearance of the ABAQUS.*Lck* file and freeze the optimization process till the FEA is completed and results are extractable.

3.4 Obtaining FEM model parameters (line 17)

In the fourth section, using the second generated MATLAB function (**ObtPar.m** (98-line)), model parameters including nodal coordinates, elemental connectivity, elements centroid coordinates, nodes in an element filtering radius, elements connected to a node, number of elements (*NmEl*), and nodes (*NmNd*), initial volume (*InitialVol*), and design area volume (*DesVol*) are extracted from the model input file (*Job-1.inp*). Based on these data, the weight of nodes in an element filter radius (*Nd_weight*) and elements' weight connected to a node (*El_weight*) will be calculated. All these data will be saved to *BESO_Arrays.mat* and will be called at each iteration to be used in the sensitivity analysis (Eqs. 3, 4 and 5) and volume evolution formula (Eq. 6).

3.5 Beginning of the evolutionary process (lines 18–50)

In Sect. 5, the evolutionary procedure of the BESO method is implemented. Before the initiation of the optimization loop, some definitions need to be made (lines 18–22). Firstly, a zero matrix is created for saving elemental old (previous *While* loop) sensitivity numbers, and three zero matrices are created for saving the index (number) of dead (removed) and live elements. To count down the value of the current iteration, the *Iter* variable is defined and set to zero. The *error* variable indicating the value of the compliance convergence criteria is set to 1000.

The optimization iterative procedure is implemented in a *While* loop. To start the first iteration, the FE results of the initial design domain analysis (*Iteration_0.dat*) are saved to a variable named *DatName* (line 24). The *Iter* counter, which has been set to zero, is updated to one in line 25. It should be noted that the updated value of the *Iter* counter (e.g. one) indicates the new design domain that will be generated at the end of the current loop based on previous iteration FEA results (e.g. 0). To conduct the sensitivity analysis, these results, plus initial BESO and model parameters obtained in Sects. 3.2, 3.3 and 3.4 are needed. For this purpose, a new function is defined named as **ElemUpd.m** (line 26).

3.5.1 Sensitivity analysis

The **ElemUpd.m** function (60-line) is responsible for conducting the sensitivity analysis and sorting of the elements for addition/removal purposes. Elemental strain energy values of the previous iteration (e.g. *Iter 0*) are needed to calculate new sensitivity numbers. The number of elements to be deleted will be determined based on the calculated target volume for the ongoing iteration (e.g. *Iter 1*). Input values of this function are the previous iteration's FEA results (e.g. *Iteration_0.dat*), *ER*, *NmEl*, *NmNd*, *Vol_Frc*, *ElSen_old*, *Iter*, and *sh_del_elm_old*. The outputs of this function will be the list of live and dead/soft elements (*sh_del_elm* and *sh_live_elm*), compliance and current volume values (*Compl* and *Cur_Vol*), volume adding ratio (*AR_Cur*), and updated elemental sensitivity numbers (*ElSen_new*). In the following, this function's sections are presented in detail.

Section 1: Exporting sensitivity numbers (lines 2–24)

Four zero matrices are created as follows to save elemental and nodal strain energy values (lines 2–5):

```
2-ElStat     = zeros(NmEl,3);
3-RoughElSen = zeros(NmEl,1);
4-NdSen      = zeros(NmNd,1);
5-ElSen      = zeros(NmEl,1);
```

The *ElStat* matrix is created to save the number of elements in the first column, their state of being (0 or 1) in the second column, and their corresponding sensitivity number in the third column. The *RoughElSen* matrix is created to save non-smoothed elemental strain energy values. *NdSen* and *ElSen* are created to save nodal and smoothed elemental strain energy values.

In line 6, the previous iteration FEA output file (e.g. *Iteration_0.dat*) is opened with the *fopen* command, and it is read line by line with the *fgetl* command until it reaches the "`ELEMENT FOOT- ELSE`" string where elemental strain energy data are written after (line 7–18). To export these values from the.dat file in a matrix format, the *importdata* command is used (line 19). Now, exported strain energy values are summed to calculate the compliance value of the structure (line 23), and then saved to the *RoughElSen* matrix (line 24).

Section 2: Smoothing sensitivity numbers (lines 25–37)

This section starts with calling *BESO_Arrays.mat*, which contains static saved data from the ObtPar.m function (line 25). These data are used in lines 26–31 to calculate smoothed sensitivity numbers (The *ElSen* matrix) based on the filtering scheme method (Eq. 4 and Eq. 5). For *Iter* number 1, while it's the first time that sensitivity analysis is conducted, there is no need to stabilization, and the *ElSen* matrix is directly saved to the *ElStat(:,3)* matrix (line 33). But for *Iter* numbers greater than one, the average total amount of new and old sensitivity numbers, *ElSen(:,1)* and *ElSen_old*, is calculated based on Eq. 5 and then saved to the *ElStat(:,3)* matrix (line 35 and 37).

Section 3: Determining target volume for the next iteration (lines 38–44)

Equation 6 is implemented in lines 38–44. The term *Cur_Vol* is defined as the current volume of the design domain based on the previous iteration.dat file data and then used to calculate the target volume (*TargVol*) for the next iteration.

Section 4: Determining the Elements for ADD/DEL Process (lines 45–60)

To begin the add/del step, the *sort* command is used to treat the third column of the *ElStat* matrix as a vector and sort elemental strain energy values in an ascending format. The index of the elements (indicating the number of the elements) is also considered when sorting them (lines 45–46). Then, counter *i* is defined as $i = 1$, and the variable *sum_vol* is set to 0 (lines 47–48). Using a *While* loop, the number of elements to be deleted from the design domain is counted using counter *i* until the following inequality is not satisfied (lines 49–55):

```
49-sum_vol<(DesVol-TargVol)
```

After determining the number of elements to be deleted, two matrices named *sh_del_elm* and *sh_live_elm*, are defined to save the elemental number of live and dead/soft elements (lines 56–59). Finally, these matrices, alongside with compliance value of the structure (*Compl*), current (*Cur_Vol*) and initial volume (*InitialVol*) values, and updated sensitivity numbers (*ElSen_new*) are extracted from the **ElemUpd.m** function to be used in the following lines of the evolutionary process (**BESO.m** *While* loop):

```
26-
[sh_del_elm,sh_live_elm,Compl(Iter),Cur_Vol(Iter),ElSen_new,InitialVol]=ElemU
pd(DatName,ER,NmEl,NmNd,Vol_Frc,ElSen_old,Iter,sh_del_elm_old);
```

In lines 27–29, *ElSen_new*, *sh_del_elm*, and *sh_live_elm* matrices, extracted from the previous iteration FE analysis, are saved as old values to be used in the next loop:

```
27-ElSen_old       = ElSen_new;
28-sh_live_elm_old = sh_live_elm;
29-sh_del_elm_old  = sh_del_elm;
```

3.5.2 Updating the design area by creating an input file for the next loop FEA (line 30)

As the list of live and dead/soft elements is extracted, the design domain can be updated and saved as a new input file (*iteration_1.inp*) for the next iteration. In this regard, a new function named **InpUpd.m** (20-line) is defined to create the new updated input file (line 30). The input data for the InpUpd.m function are the iteration number, *Job-1temp* file, and the list of dead/soft elements extracted from the **ElemUpd.m** function. This function opens the temporary job file *(Job-1temp*) as a reading source, and a new writable file named *Iteration_1*. The *Job-1temp* file is read line by line and written to the *Iteration_1* file, including the number of elements to be deleted or softly killed.

3.5.3 New FEA on the updated design area (lines 31–32)

As the new input file is created based on the previous iteration results, a new FEA should be conducted before the initiation of the new iteration. So, again the *system* command (line 31) is used to summon ABAQUS and run the *Iteration_1.inp* file. The FEA result report file (*Iteration_1.dat*) will be saved in the working directory, ready to be called in the second iteration.

3.5.4 Convergence criterion check (lines 33–38)

At the end of this section, the convergence of the objective function is calculated based on Eq. 7 (lines 33–38).

3.6 Exporting plots (lines 39–50)

As the convergence criterion is satisfied, the optimization *While* loop ends, and the objective function and the volume constraint evolutions can be plotted. Section 6 (lines 39–50) is written for this purpose using the MATLAB *plot* command.

4 Results

In this section, samples for the maximum stiffness problem (minimum compliance) using the MATLAB-ABAQUS TOP software package are presented. Using this software interface, the optimum material distribution patterns for the sample problems are obtained, and results are compared to the SIMP method results. Topology evolution of the example problems along with the convergence of the objective function and the volume constraint indicate the validity and efficient performance of the developed MATLAB-ABAQUS TOP interface.

4.1 A short cantilever problem

A design domain of 80 mm in length, 55 mm in height, and 1 mm in thickness (2D planar modeling space) is defined in the ABAQUS part module, as shown in Fig. 3. A Young's modulus of 100 GPa, and a Poisson's ratio of 0.3 are defined in the ABAQUS material module. A concentrated downward force of 100 N is applied for a static analysis step. Then, the FE model is generated using linear quad plane-stress (CPS4R) elements (mesh grid of 100×80). In the job module, a new job is defined named *Job-1* and an input file is generated using the *Write Input* option. A copy of this file is created and renamed to *Job-1temp*, and changes are made to these files as explained in Sect. 3.1.

The BESO parameters for the short cantilever problem are as presented in Table 2:

Running the interface for the above mentioned inputs, the optimization results are achieved as indicated in Fig. 4 and 5.

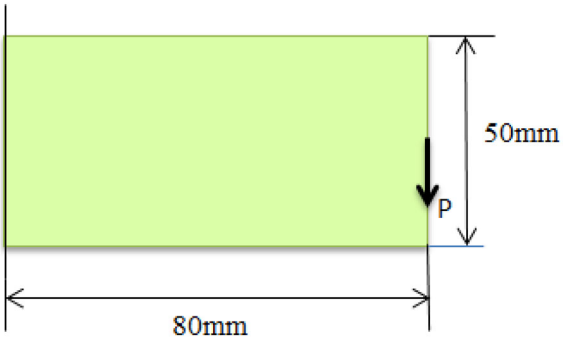

Fig. 3 The design domain, loading, and boundary conditions of a short cantilever

Table 2 The BESO parameters for the short cantilever problem

ER	AR_{max}	r_{min}	τ	x_{min}(Only soft-kill approach)	p(Only soft-kill approach)
2%	5%	3 mm	0.01%	0.001	3

As can be seen in Fig. 4, while the volume fraction constraint reaches 0.5, the mean compliance value converges to 1.88 (N.mm). Topology convergence occurs after iteration 45 (Fig. 5). To verify the validity of the developed MATLAB-ABAQUS TOP computer interface, the final topology is compared to the SIMP method result, as depicted in Fig. 6.

It should be noted that in the SIMP method, a continuum design domain including gray elements is achieved, resulting in a higher strain energy value. However, the BESO approach is a discrete-based (0–1) method meaning that no gray elements are allowed to remain in the design domain; so, resulting in a faster FEA process.

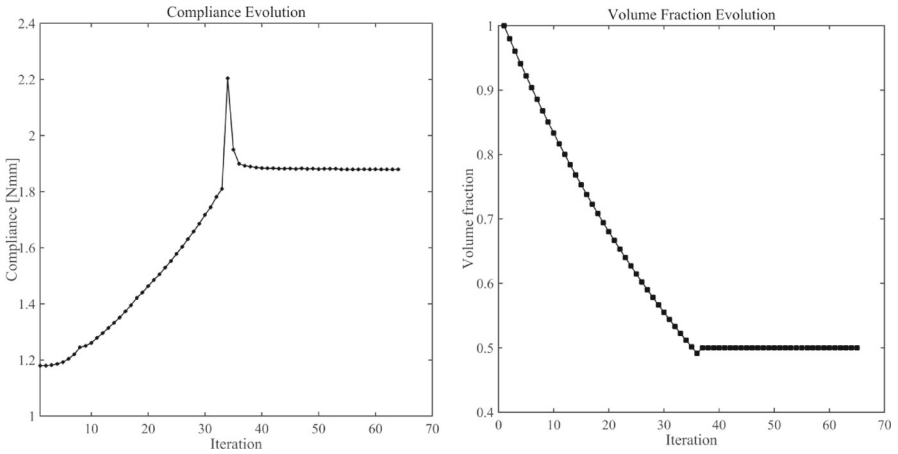

Fig. 4 Evolution histories of the objective function (strain energy) and the volume constraint

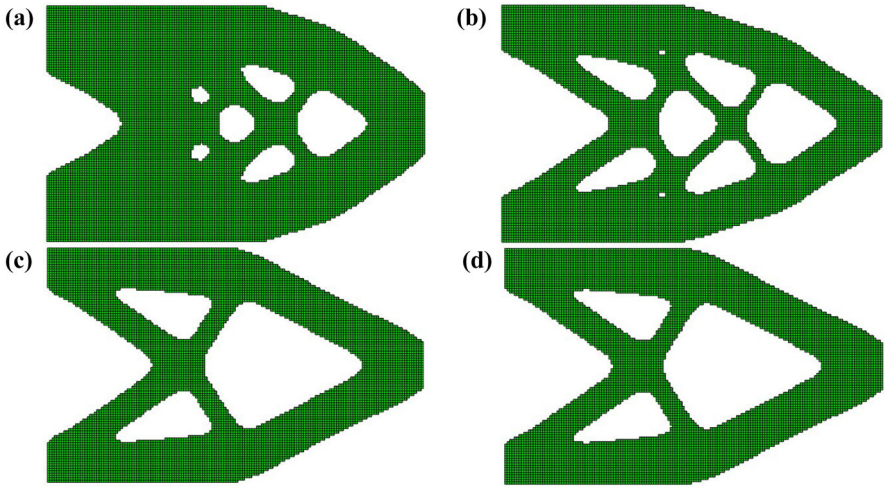

Fig. 5 Evolution histories of the short cantilever topology: **a** iteration 15; **b** iteration 30; **c** iteration 45; **d** iteration 65 (final)

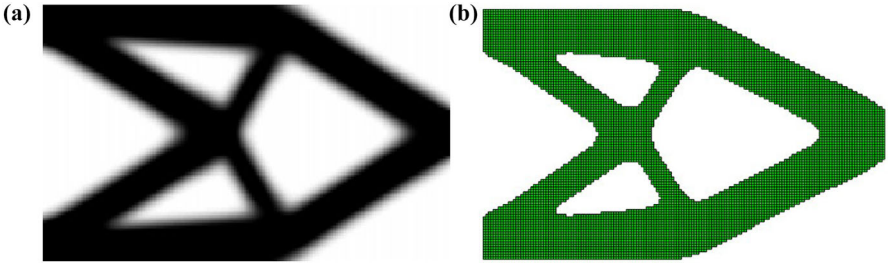

Fig. 6 Comparing results of the SIMP **a** and the BESO **b** methods for the short cantilever problem

4.2 An MBB beam problem

In this example, An MBB (Fig. 7) beam with a design domain of 240 mm in length, 40 mm in height, and 1 mm in thickness is defined in the ABAQUS part module. A concentrated downward force of 100 N is applied for a static analysis step, and a Young's modulus of 200 GPa and a Poisson's ratio of 0.3 are considered as the material properties. The same FE modeling procedure is applied to discretize the

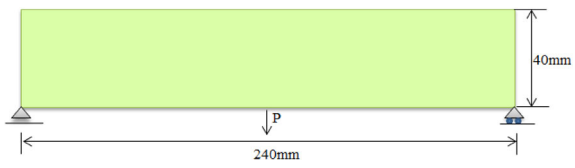

Fig. 7 The design domain, loading, and boundary conditions of an MBB beam

Table 3 The BESO parameters for the MBB beam problem

ER	AR_{max}	r_{min}	τ	x_{min}(Only soft-kill approach)	p(Only soft-kill approach)
5%	5%	6 mm	0.01%	0.001	3

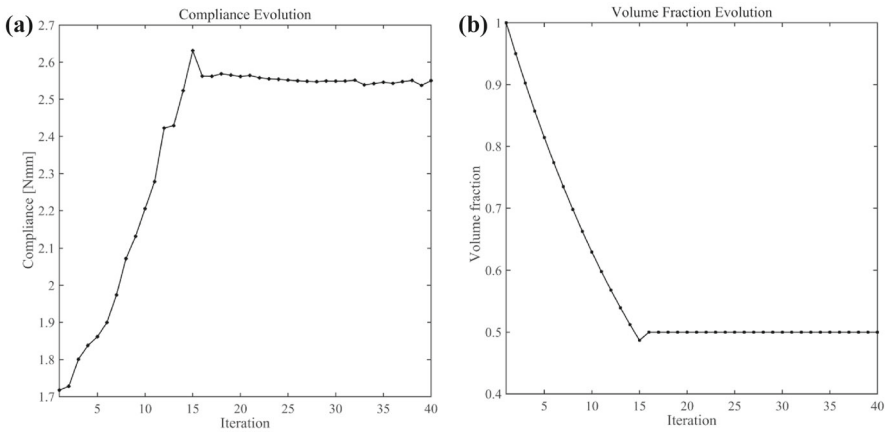

Fig. 8 Evolution histories of **a** the mean compliance and **b** the volume fraction for the classic MBB beam

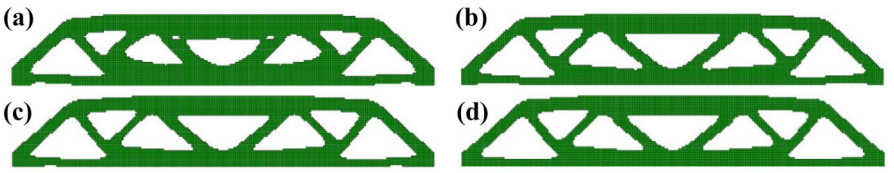

Fig. 9 Evolution histories of the MBB beam topology: **a** iteration 10, **b** iteration 20, **c** iteration 30, and **d** iteration 40 (final)

design area into 240 × 40 CPS4R elements. *Job-1* and its copy, *Job-1temp*, are also created.

The BESO parameters for this case study are considered as indicated in Table 3.

Figure 8 indicates that as the volume fraction constraint reaches 0.5, the mean compliance value converges to 2.55 (N.mm). The topology convergence, as shown in Fig. 9, occurs after iteration 15. To verify the validity of the result, the final topology is compared to the SIMP method result (Fig. 10).

4.3 A Bridge-type structure

This example, as indicated in Fig. 11, includes a rectangular design domain of 240 mm in length, 40 mm in height, and 1 mm in thickness. A uniformly-distributed load of

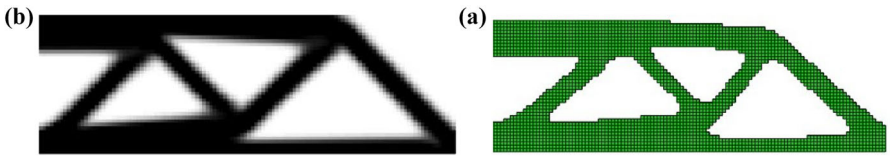

Fig. 10 Comparing SIMP **a** and BESO **b** methods' results for the MBB beam problem

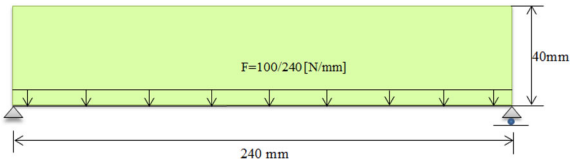

Fig. 11 The design domain, loading, and boundary conditions of a bridge-type structure

Table 4 The BESO parameters for the bridge-type structure

ER	AR_{max}	r_{min}	τ	x_{min}(Only soft-kill approach)	p(Only soft-kill approach)
2%	5%	6 mm	0.1%	0.001	3

Fig. 12 Design area ■ and non-design area ■ in the bridge-type structure

100/240 N/mm is applied to the bottom deck of 240 mm × 1 mm, and its two corners are constrained. The design area is discretized into 240 × 40 CPS4R elements.

The BESO parameters for the bridge-type structure are considered as indicated in Table 4.

To stop the elements of the deck area from deletion, the design domain is divided into two sub-domains of design and frozen areas, as shown in Fig. 12. In this regard, elements of the frozen area are excluded from the list of *sh_del_elm*.

Running the interface for the above mentioned inputs, the optimization results are achieved as as indicated in Figs. 13 and 14.

As can be seen in Figs. 13 and 14, convergence occurs after iteration 35 for the objective function, the volume constraint, and the structure topology.

4.4 A Multiple-load case problem

In this example, a multiple-load case, which is one of the basic TO problems, is considered. A rectangular design domain of 240 × 40 × 1 mm^3, as shown in Fig. 15, is loaded in two different points and its two corners are constrained. The design area is discretized into 240 × 40 CPS4R elements .

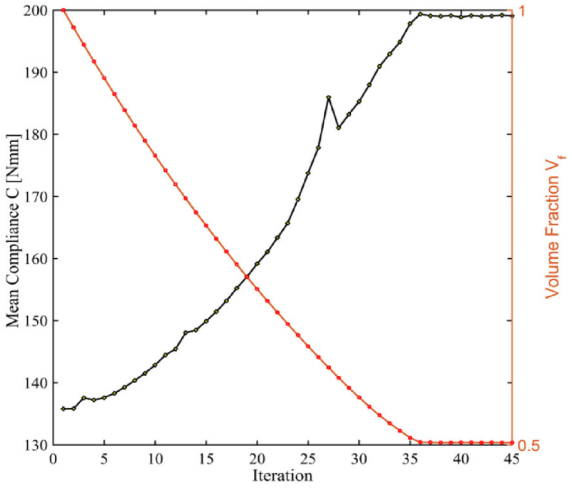

Fig. 13 Evolution histories of the mean compliance and the volume fraction for the bridge-type structure

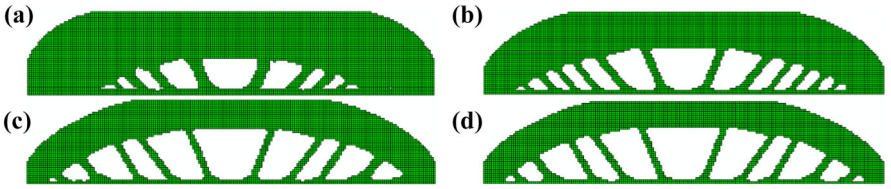

Fig. 14 Evolution histories of the bridge-type structure topology: **a** iteration 10, **b** iteration 20, **c** iteration 30, and **d** iteration 45 (final)

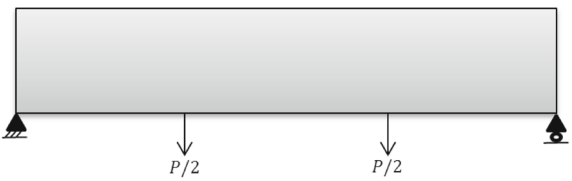

Fig. 15 Design domain, loading, and boundary conditions for a multiple-load case problem

The BESO parameters are considered as indicated in Table 5.

Running the interface for the above mentioned inputs, the optimization results are achieved as illustrated in Figs. 16 and 17.

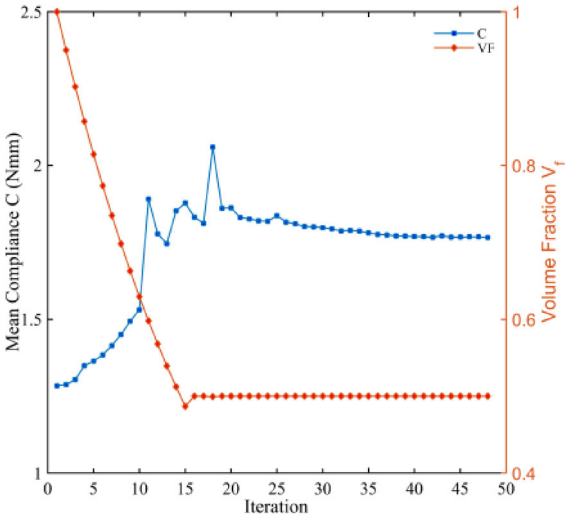

Fig. 16 Evolution histories of the mean compliance and the volume fraction for the multiple-load case problem

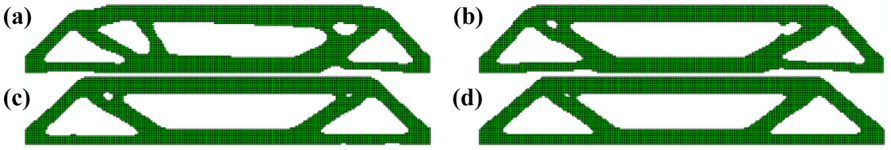

Fig. 17 Evolution histories of the multiple-load case problem's topology: **a** iteration 10, **b** iteration 20, **c** iteration 30, and **d** iteration 47 (final)

Table 5 The BESO parameters for the multiple-load case problem

ER	AR_{max}	r_{min}	τ	x_{min}(Only soft-kill approach)	p(Only soft-kill approach)
5%	5%	6 mm	0.01%	0.001	3

5 Conclusion

In this article, a BESO-based MATLAB-ABAQUS interface is presented for solving 2D compliance problems. In this regard, the BESO method, developed by Huang and Xie (2010b), is implemented by defining different MATLAB functions. For the FEA part, ABAQUS software is considered which is summoned by the optimization source code when required. The format of the ABAQUS input and output files suitable for the MATLAB optimization code is discussed in detail. The MATLAB functions, each one being responsible for a step of the BESO method, are explained and all codes for connecting MATLAB and ABAQUS are presented. Different examples of the compliance problem are solved and results are compared to the SIMP method outputs to validate and verify the developed interface. This code is presented for educational and engineering practice purposes, which can be considered as a computer program asset for students new to the topology optimization field. This work is aimed at developing a multipurpose efficient optimization software interface that could be easily understood, used, and developed by students and researchers all around the world. Using ABAQUS as the finite element solver, many issues arising from FEM codes are resolved. So, geometry and loading are not considered design challenges anymore. This interface has the potential to be developed for other topology optimization problems and 3D geometries. All codes are presented in the Appendix section.

6 Supplementary materials

A folder named 2DBESO-MATLABAQUS, including MATLAB m-files and ABAQUS input job files is presented for a sample problem (2D MMB Beam). Instructions for running the software package are given in a text file saved into the mentioned folder.

Appendix

In this appendix, the 2D-BESO MATLAB functions developed for compliance topology optimization problems are presented. All functions' m-files should be saved in one folder along with the ABAQUS job input files.

BESO.m (Optimization engine (source code))

```
%% Setting the working directory of the program
clc;
clear all;
tic;
WorDir='Copy & paste the folder address where m-files and .inp files exist';
AbqDir=['cd ' WorDir];
system(AbqDir); %Make WorDir ABAQUS working directory
%% 1- Define a design domain
InpName='Job-1';
InpBase='Job-1temp';
%% 2- Define the BESO parameters
ER=0.05;
AR_Max=0.05;
Rmin=6;
Vol Frc=0.5;
%% 3- Zeroth iteration (Itration_0)
Itr0Cmd=['abq6142 job=Iteration_0 input=' InpName];
system(Itr0Cmd);
Iter=0;
LckChk(Iter)
%% 4- Obtaining FEM model parameter for element add/deletion process
[NmEl,NmNd]=ObtPar(InpName,Rmin);
%% 5- Beginning of the Evolutionary Process
ElSen old=zeros(NmEl,1);
sh_del_elm=[];
sh_del_elm_old  = [];
sh live elm old = [];
error=1000;
while error>0.0001
DatName=['Iteration_' num2str(Iter) '.dat'];
Iter=Iter+1;
   %% 5-1 Sensitivity analysis (ElemUpd.m function)
   [sh del elm,sh live elm,Compl(Iter),Cur Vol(Iter),ElSen new,InitialVol]=ElemUpd(DatName,ER,NmEl,NmNd,Vol_Frc,ElSen_old,Iter,sh_del_elm_old);
       ElSen_old = ElSen_new;
       sh live elm old = sh live elm;
       sh del elm old  = sh del elm;
   %% 5-2 Creating the input file for the next loop/Updating design domain (InpUpd.m function)
       InpUpd(Iter,InpBase,sh_del_elm)
   %% 5-3 FEA on the new input file/New FEA
       system(['abq6142 job=Iteration_' num2str(Iter) ' input=Iteration_' num2str(Iter)]);
       LckChk(Iter)
   %% 5-4 Convergence criterion check
       if Iter>11
           error=abs(sum(Compl(Iter-5:Iter))-sum(Compl(Iter-10:Iter-5)))/sum(Compl(Iter-5:Iter));
       else
           error=1000;
       end
   end
%% 6- Exporting Plots
iter=1:Iter;
plot(iter,Compl);
title('Compliance Evolution');
xlabel('Iteration');
ylabel('Compliance');
figure;
iter=1:Iter;
plot(iter,Cur_Vol/sum(InitialVol(:,2)));
xlabel('Iteration');
ylabel('Volume fraction');
title('Volume Fraction Evolution');
time=toc;
```

1. ObtPar.m (Obtaining FEM model parameters)

```
function [NmEl,NmNd]=ObtPar(InpName,Rmin)

%% Nodal Coordinates
fid=fopen([InpName '.inp'],'r');
line_counter=1;
tline = fgetl(fid);
while ischar(tline)
    tline = fgetl(fid);
    line_counter=line_counter+1;
    header=strfind(tline,'*Node');

       if ~isempty(header)
           fclose('all');
           break;
       end
   end
   NodeData=importdata([InpName '.inp'],',',line_counter);
   NodeCoord=NodeData.data(:,2:3);

%% Elemental Connectivity
   fid=fopen([InpName '.inp'],'r');
   line_counter=1;
   tline = fgetl(fid);
   while ischar(tline)
       tline = fgetl(fid);
       line_counter=line_counter+1;
       header=strfind(tline,'*Element');
       if ~isempty(header)
           fclose('all');
           break;
       End
   end
   ElemData1=importdata([InpName '.inp'],',',line_counter);
   Connectivity=ElemData1.data(:,2:5);

%% Element Centroid Coordinates
   fid=fopen('Iteration_0.dat','r');
   line_counter=1;
   tline = fgetl(fid);
   while ischar(tline)
       tline = fgetl(fid);
       line_counter=line_counter+1;
       header=strfind(tline,'ELEMENT  FOOT-   COORD1      COORD2');
       if ~isempty(header)
           Header_Line=line_counter+2;
           fclose('all');
           break;
       End
   end
   ElemData2=importdata('Iteration_0.dat',' ',Header_Line);
   ElemCoord=ElemData2.data(:,2:3);

%% Nodes that are in an Element filter
   seed=((NodeCoord(1,1)-NodeCoord(2,1))^2+(NodeCoord(1,2)-
NodeCoord(2,2))^2)^(0.5);
   size=100;
   ElemFilter=zeros(length(ElemCoord(:,1)),size);
   for i=1:length(ElemCoord(:,1))
       temp=(((NodeCoord(:,1)-ElemCoord(i,1)).^2+(NodeCoord(:,2)-
ElemCoord(i,2)).^2).^(0.5)<Rmin);
       temp=(find(temp==1))';
       ElemFilter(i,1:length(temp))=temp;
   end
```

```
%% Computing elements that are connected to a node
    NodalConnect=zeros(length(NodeCoord(:,1)),4);
    NmNd=length(NodeCoord(:,1));
    NmEl=length(ElemCoord(:,1));
    for i=1:length(ElemCoord(:,1))
        for j=1:4
           NodalConnect(Connectivity(i,j),j)=i;
        End
    end
    Conncted El=zeros(length(NodeCoord(:,1)),4);

    for i=1:length(NodalConnect(:,1))
Conncted_El(i,1:sum(NodalConnect(i,:)~=0))=NodalConnect(i,NodalConnect(i,:
)~=0);
    End

%% Computing the weights of elements connected to a node
    El_weight=zeros(length(NodeCoord(:,1)),4);
    for i=1:length(Conncted_El(:,1))
        M=sum(Conncted El(i,:)~=0);
        temp=Conncted_El(i,Conncted_El(i,:)~=0);
        if M==1
            El weight(i,1)=1;
        else
            El_weight(i,1:M)=(1-(((ElemCoord(temp,1)-
NodeCoord(i,1)).^2+(ElemCoord(temp,2)-
NodeCoord(i,2)).^2).^(0.5))/sum(((ElemCoord(temp,1)-
NodeCoord(i,1)).^2+(ElemCoord(temp,2)-NodeCoord(i,2)).^2).^(0.5)))/(M-1);
        end
    end

%% Computing the weights of the nodes that are in an elements filter
    Nd_weight=zeros(length(ElemFilter(:,1)),size);
    for i=1:length(ElemFilter(:,1))
        M=sum(ElemFilter(i,:)~=0);
        temp=ElemFilter(i,ElemFilter(i,:)~=0);
        Nd_weight(i,1:M)=Rmin-((NodeCoord(temp,1)-
ElemCoord(i,1)).^2+(NodeCoord(temp,2)-ElemCoord(i,2)).^2).^(0.5);
    end

    fid=fopen('Iteration_0.dat','r');
    line_counter=1;
    tline = fgetl(fid);
    while ischar(tline)
        tline = fgetl(fid);
        line counter=line counter+1;
        header=strfind(tline,'ELEMENT  FOOT-   EVOL');
        if ~isempty(header)
            header_line=line_counter+2;
            fclose('all');
            break;
        end
    end
    Vol_Data=importdata('Iteration_0.dat',' ',header_line);
    InitialVol=Vol_Data.data(:,:);
    DesVol=sum(InitialVol(:,2));

    save('BESO_Arrays.mat','ElemFilter','NodalConnect','Nd_weight','El_weig
ht','InitialVol','DesVol');
```

2. Lck.Chk.m (ABAQUS running)

```
function LckChk(Iter)
A=0;
while A==0
    A=exist(['Iteration_' num2str(Iter) '.lck'],'file');
end
A=2;
while A==2
    A=exist(['Iteration_' num2str(Iter) '.lck'],'file');
end
```

3. Elem.Upd.m (Sensitivity analysis)

```
function
[AR Cur,sh del elm,sh live elm,Compl,Cur Vol,ElSen new,InitialVol]=ElemUpd
(DatName,ER,NmEl,NmNd,Vol Frc,ElSen old,Iter,sh del elm old)

%% 1-Exporting sensitivity numbers from previous iteration FEA results
ElStat     = zeros(NmEl,3);  %number, state, sensitivity
RoughElSen = zeros(NmEl,1);
NdSen      = zeros(NmNd,1);
ElSen      = zeros(NmEl,1);

fid = fopen(DatName,'r');
line counter=1;
tline = fgetl(fid);
while ischar(tline)
       tline = fgetl(fid);
       line_counter=line_counter+1;
       header=strfind(tline,'ELEMENT  FOOT-   ELSE');
       if ~isempty(header)
           header line=line counter+2;
           fclose('all');
           break;
       end
   end
   Enrg Data = importdata(DatName,' ',header line);
   ELSE      = Enrg Data.data(:,:);
   ElStat(:,1)=1:NmEl;
   ElStat(ELSE(:,1),2) = 1;   %alive elements
   Compl = sum(ELSE(:,2));
   RoughElSen(ELSE(:,1),1) = ELSE(:,2);

%% 2- Smoothing sensitivity numbers
   load BESO Arrays.mat
   for i=1:NmNd
NdSen(i)=sum(RoughElSen(NodalConnect(i,NodalConnect(i,:)~=0),1).*El_weight
(i,1:sum(NodalConnect(i,:)~=0))');
   end
   for i=1:NmEl
ElSen(i)=sum(NdSen(ElemFilter(i,ElemFilter(i,:)~=0),1).*Nd weight(i,1:sum(
ElemFilter(i,:)~=0))');
   end
   if Iter==1
       ElStat(:,3)=ElSen(:,1);
   else
       ElStat(:,3)=(ElSen(:,1)+ElSen old)/2;
   end
   ElSen new=ElStat(:,3);

%% 3-Determining target volume for the next iteration
   Cur Vol=sum(InitialVol(ELSE(:,1),2));
   if Cur Vol>Vol Frc*DesVol
       TargVol=(1-ER)*Cur Vol;
   end
   if Cur Vol<=Vol Frc*DesVol
           TargVol=Vol Frc*DesVol;
   End

%% 4-Determining elements to be deleted or added
   [Y,I]=sort(ElStat(:,3));  % "I" gives the index of element in column;
"Y" is the sorted columns of matrix
   Stat_Sorted=ElStat(I,:);
   i=1;
   sum_vol=0;
   while sum vol<(DesVol-TargVol)
         sum_vol=sum_vol+InitialVol(Stat_Sorted(i,1),2);
         i=i+1;
   end
   if mod(i,2)==1
       i=i-1;
   end
   AllElements = 1:NmEl;
   AllElements = AllElements(:);
   sh del elm  = Stat Sorted(1:i,1);
   sh live elm = setdiff(AllElements,sh del elm);

   end
```

4. Inp.Upd.m (Creating new input file for next iteration)

```
function InpUpd(Iter,InpBase,sh del elm)
fidin=fopen([InpBase '.inp'],'r');
fidout=fopen(['Iteration_' num2str(Iter) '.inp'],'w+');
line_counter=1;
tline = fgetl(fidin);
while ischar(tline)
    header=strfind(tline,'*Elset, elset=sh del elm, instance=Part-1-1');
    if ~isempty(header)
        fprintf(fidout,'%s\n',tline);
        fprintf(fidout,'%d, %d, %d,\n', sh_del_elm);
        if mod(length(sh_del_elm),3)==1 || mod(length(sh_del_elm),3)==2
            fprintf(fidout,'\n');
        end
        tline = fgetl(fidin);
        continue;
    end
    fprintf(fidout,'%s\n',tline);
    tline = fgetl(fidin);
end
fclose('all');
```

References

Allaire G, Jouve F, Toader A-M (2002) A level-set method for shape optimization. CR Math 334(12):1125–1130

Andreassen E et al (2011) Efficient topology optimization in MATLAB using 88 lines of code. Struct Multidiscip Optim 43(1):1–16

Ansola Loyola R et al (2018) A sequential element rejection and admission (SERA) topology optimization code written in Matlab. Struct Multidiscip Optim 58(3):1297–1310

Asgari M (2015) Material distribution optimization of 2D heterogeneous cylinder under thermo-mechanical loading. Struct Eng Mech Int J 53(4):703–723

Asgari M (2016) Material optimization of functionally graded heterogeneous cylinder for wave propagation. J Compos Mater 50(25):3525–3528

Banh TT et al (2021a) Multiple bi-directional FGMs topology optimization approach with a preconditioned conjugate gradient multigrid. Steel Compos Struct 41(3):385–402

Banh TT, Luu NG, Lee D (2021b) A non-homogeneous multi-material topology optimization approach for functionally graded structures with cracks. Compos Struct 273:114230

Banh TT et al (2023) A robust dynamic unified multi-material topology optimization method for functionally graded structures. Struct Multidiscip Optim 66(4):75

Bendsøe MP (1989) Optimal shape design as a material distribution problem. Struct Optim 1(4):193–202

Bendsøe MP et al (1995) Optimal design of material properties and material distribution for multiple loading conditions. Int J Numer Methods Eng 38(7):1149–1170

Bendsøe MP et al. (1994) On the prediction of extremal material properties and optimal material distribution for multiple loading conditions. In: International design engineering technical conferences and computers and information in engineering conference. American Society of Mechanical Engineers

Bourdin B, Chambolle A (2003) Design-dependent loads in topology optimization. ESAIM Control Optim Calcul Var 9:19–48

Challis VJ (2010) A discrete level-set topology optimization code written in Matlab. Struct Multidiscip Optim 41(3):453–464

Chen Q, Zhang X, Zhu B (2019) A 213-line topology optimization code for geometrically nonlinear structures. Struct Multidiscip Optim 59(5):1863–1879

Doan QH, Lee D (2017) Optimum topology design of multi-material structures with non-spurious buckling constraints. Adv Eng Softw 114:110–120

Dong G et al (2020) Design and optimization of solid lattice hybrid structures fabricated by additive manufacturing. Addit Manuf 33:101116

Eschenauer HA, Kobelev VV, Schumacher A (1994) Bubble method for topology and shape optimization of structures. Struct Optim 8(1):42–51
Ferrari F, Sigmund O (2020) A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D. Struct Multidiscip Optim 62(4):2211–2228
Gao J et al (2021) IgaTop: an implementation of topology optimization for structures using IGA in MATLAB. Struct Multidiscip Optim 64(3):1669–1700
Han Y, Xu B, Liu Y (2021) An efficient 137-line MATLAB code for geometrically nonlinear topology optimization using bi-directional evolutionary structural optimization method. Struct Multidiscip Optim 63(5):2571–2588
Huang X, Xie Y (2007a) Convergent and mesh-independent solutions for the bi-directional evolutionary structural optimization method. Finite Elem Anal Des 43(14):1039–1049
Huang X, Xie Y (2007b) Bidirectional evolutionary topology optimization for structures with geometrical and material nonlinearities. AIAA J 45(1):308–313
Huang X, Xie Y (2008a) Topology optimization of nonlinear structures under displacement loading. Eng Struct 30(7):2057–2068
Huang X, Xie Y (2008b) Optimal design of periodic structures using evolutionary topology optimization. Struct Multidiscip Optim 36(6):597–606
Huang X, Xie YM (2009) Bi-directional evolutionary topology optimization of continuum structures with one or multiple materials. Comput Mech 43(3):393–401
Huang X, Xie Y (2010a) Evolutionary topology optimization of continuum structures with an additional displacement constraint. Struct Multidiscip Optim 40(1):409–416
Huang X, Xie M (2010b) Evolutionary topology optimization of continuum structures: methods and applications. Wiley, Heidellberg
Huang X, Xie YM, Lu G (2007) Topology optimization of energy-absorbing structures. Int J Crashworthiness 12(6):663–675
Huang X, Zuo Z, Xie Y (2010) Evolutionary topological optimization of vibrating continuum structures for natural frequencies. Comput Struct 88(5–6):357–364
Lagaros ND, Vasileiou N, Kazakis G (2019) AC# code for solving 3D topology optimization problems using SAP2000. Optim Eng 20(1):1–35
Laurain A (2018) A level set-based structural optimization code using FEniCS. Struct Multidiscip Optim 58(3):1311–1334
Li Q et al (1999) Shape and topology design for heat conduction by evolutionary structural optimization. Int J Heat Mass Transf 42(17):3361–3371
Liang Y, Cheng G (2020) Further elaborations on topology optimization via sequential integer programming and Canonical relaxation algorithm and 128-line MATLAB code. Struct Multidiscip Optim 61(1):411–431
Liu K, Tovar A (2014) An efficient 3D topology optimization code written in Matlab. Struct Multidiscip Optim 50(6):1175–1196
Luo Z et al (2008) A level set-based parameterization method for structural shape and topology optimization. Int J Numer Methods Eng 76(1):1–26
Meng Z et al (2020) New hybrid reliability-based topology optimization method combining fuzzy and probabilistic models for handling epistemic and aleatory uncertainties. Comput Methods Appl Mech Eng 363:112886
Meng Z et al (2021) Robust topology optimization methodology for continuum structures under probabilistic and fuzzy uncertainties. Int J Numer Methods Eng 122(8):2095–2111
Meng Z et al (2022) A fidelity equivalence computation method for topology optimization of geometrically nonlinear structures. Eng Optim. https://doi.org/10.1080/0305215X.2022.2146684
Michell AGM (1904) LVIII. The limits of economy of material in frame-structures. Lond Edinb Dublin Philos Mag J Sci 8(47):589–597
Ngoc NM, Hoang V-N, Lee D (2022) Concurrent topology optimization of coated structure for non-homogeneous materials under buckling criteria. Eng Comput 38(6):5635–5656
Picelli R, Sivapuram R, Xie YM (2021) A 101-line MATLAB code for topology optimization using binary variables and integer programming. Struct Multidiscip Optim 63(2):935–954
Querin OM, Steven GP, Xie YM (1998) Evolutionary structural optimisation (ESO) using a bidirectional algorithm. Eng Comput 15(8):1031–1048
Querin O, Steven G, Xie Y (2000a) Evolutionary structural optimisation using an additive algorithm. Finite Elem Anal Des 34(3):291–308

Querin O et al (2000b) Computational efficiency and validation of bi-directional evolutionary structural optimisation. Comput Methods Appl Mech Eng 189(2):559–573
Salgarello M, Visconti G, Barone-Adesi L (2013) Interlocking circumareolar suture with undyed polyamide thread: a personal experience. Aesthetic Plast Surg 37(5):1061–1062
Sethian JA, Wiegmann A (2000) Structural boundary design via level set and immersed interface methods. J Comput Phys 163(2):489–528
Sigmund O (2001) A 99 line topology optimization code written in Matlab. Struct Multidiscip Optim 21(2):120–127
Sokolowski J, Zochowski A (1999) On the topological derivative in shape optimization. SIAM J Control Optim 37(4):1251–1272
Talischi C et al (2012) PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Struct Multidiscip Optim 45(3):329–357
Tavakoli R, Mohseni SM (2014) Alternating active-phase algorithm for multimaterial topology optimization problems: a 115-line MATLAB implementation. Struct Multidiscip Optim 49(4):621–642
Teimouri M, Asgari M (2019) Multi-objective BESO topology optimization for stiffness and frequency of continuum structures. Struct Eng Mech 72(2):181–190
Teimouri M, Asgari M (2020) Developing a bidirectional evolutionary topology algorithm for continuum structures with the objective functions of stiffness and fundamental frequency with geometrical symmetry constraint. Amirkabir J Mech Eng 52(1):249–264
Teimouri M, Asgari M (2021) Mechanical performance of additively manufactured uniform and graded porous structures based on topology-optimized unit cells. Proc Inst Mech Eng C J Mech Eng Sci 235(9):1593–1618
Teimouri M, Mahbod M, Asgari M (2021) Topology-optimized hybrid solid-lattice structures for efficient mechanical performance. In: Structures. Elsevier
Wang MY, Wang X, Guo D (2003) A level set method for structural topology optimization. Comput Methods Appl Mech Eng 192(1):227–246
Wei P et al (2018) An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. Struct Multidiscip Optim 58(2):831–849
Xia L, Breitkopf P (2015) Design of materials using topology optimization and energy-based homogenization approach in Matlab. Struct Multidiscip Optim 52(6):1229–1241
Xia Q, Shi T (2016a) Topology optimization of compliant mechanism and its support through a level set method. Comput Methods Appl Mech Eng 305:359–375
Xia Q, Shi T (2016b) Optimization of structures with thin-layer functional device on its surface through a level set based multiple-type boundary method. Comput Methods Appl Mech Eng 311:56–70
Xia Q et al (2006) Semi-Lagrange method for level-set-based structural topology and shape optimization. Struct Multidiscip Optim 31(6):419–429
Xia Q, Shi T, Wang MY (2011) A level set based shape and topology optimization method for maximizing the simple or repeated first eigenvalue of structure vibration. Struct Multidiscip Optim 43(4):473–485
Xia Q et al (2012) A level set solution to the stress-based structural shape and topology optimization. Comput Struct 90:55–64
Xia Q, Wang MY, Shi T (2014) A level set method for shape and topology optimization of both structure and support of continuum structures. Comput Methods Appl Mech Eng 272:340–353
Xia L et al (2018) Bi-directional evolutionary structural optimization on advanced structures and materials: a comprehensive review. Arch Comput Methods Eng 25(2):437–478
Xie Y, Steven GP (1992) Shape and layout optimization via an evolutionary procedure. In: Proceedings of the international conference on computational engineering science
Xie YM, Steven GP (1993) A simple evolutionary procedure for structural optimization. Comput Struct 49(5):885–896
Yaghmaei M, Ghoddosian A, Khatibi MM (2020) A filter-based level set topology optimization method using a 62-line MATLAB code. Struct Multidiscip Optim 62(2):1001–1018
Zhang W et al (2016) A new topology optimization approach based on Moving Morphable Components (MMC) and the ersatz material model. Struct Multidiscip Optim 53(6):1243–1260
Zhao C et al (1998) A generalized evolutionary method for numerical topology optimization of structures under static loading conditions. Struct Optim 15(3):251–260
Zhou M, Rozvany G (1991) The COC algorithm, Part II: topological, geometrical and generalized shape optimization. Comput Methods Appl Mech Eng 89(1–3):309–336

Zhu B et al (2020) Design of compliant mechanisms using continuum topology optimization: a review. Mech Mach Theory 143:103622
Zhu B et al (2021) An 89-line code for geometrically nonlinear topology optimization written in FreeFEM. Struct Multidiscip Optim 63(2):1015–1027
Zuo ZH, Xie YM (2015) A simple and compact Python code for complex 3D topology optimization. Adv Eng Softw 85:1–11