

Highlights

**SOPTX: A High-Performance Multi-Backend Framework for Topology Optimization**

Liang He,

- 
- 
-

# SOPTX: A High-Performance Multi-Backend Framework for Topology Optimization<sup>\*</sup>

Liang He<sup>a,\*,1</sup>, b,2

<sup>a</sup>, , , , ,  
<sup>b</sup>, , , , ,

## ARTICLE INFO

**Keywords:**

## ABSTRACT

In recent years, topology optimization (TO) has gained widespread attention in both industry and academia as an ideal structural design method. However, its application has high barriers to entry due to the deep expertise and extensive development work typically required. Traditional numerical methods for TO are tightly coupled with computational mechanics methods such as finite element analysis (FEA), making the algorithms intrusive and requiring comprehensive understanding of the entire system. This paper presents SOPTX, a TO package based on FEALPy, which implements a modular architecture that decouples analysis from optimization, supports multiple computational backends (NumPy/PyTorch/JAX), and achieves a non-intrusive design paradigm.


The main innovations of SOPTX include: (1) A cross-platform, multi-backend support system compatible with various computational backends such as NumPy, PyTorch, and JAX, enabling efficient algorithm execution on CPUs and flexible acceleration using GPUs, as well as efficient sensitivity computation for objective and constraint functions via automatic differentiation (AD); (2) A fast matrix assembly technique, overcoming the performance bottleneck of traditional numerical integration methods and significantly enhancing computational efficiency; (3) A modular framework designed to support TO problems for arbitrary dimensions and meshes, complemented by a rich library of composable components, including diverse filters and optimization methods, enabling users to flexibly configure and extend optimization workflows according to specific needs.


Taking the density-based method as an example, this paper elaborates the architecture, computational workflow, and usage of SOPTX through the classical compliance minimization problem with volume constraints. Numerical examples demonstrate that SOPTX significantly outperforms existing open-source packages in terms of computational efficiency and memory usage, especially exhibiting superior performance in large-scale problems. The modular design of the software not only enhances the flexibility and extensibility of the code but also provides opportunities for exploring novel TO problems, offering robust support for education, research, and engineering applications in TO.

## 1. Introduction

Topology optimization (TO) is an essential class of structural optimization techniques aimed at improving structural performance by optimizing material distribution within a design domain. In fields such as aerospace, automotive, and civil engineering, TO addresses critical design challenges through efficient material utilization and mechanical performance optimization. Among various approaches, density-based methods are particularly popular due to their intuitiveness and practicality. In these methods, the distribution of material and void within the design domain is optimized, and the relative density of each finite element is treated as a design variable. The most widely adopted density-based method is the Solid Isotropic Material with Penalization (SIMP) approach. This approach promotes binary (0 – 1) solutions

<sup>\*</sup>  
\*Corresponding author

 (L. He); ()

 (L. He); ()

ORCID(s):

1

by penalizing intermediate densities, and due to its simplicity and seamless integration with finite element analysis (FEA), it has been extensively employed since its introduction by Bendsoe and Sigmund (2004).

However, TO problems inherently involve large-scale computations. This is due to the tight coupling between structural analysis and element-level optimization of design variables: at each iteration, it is necessary not only to solve boundary value problems to obtain structural responses but also to calculate derivatives of the objective and constraint functions with respect to design variables, supporting gradient-based optimization algorithms. For large-scale problems, this implies solving large linear systems and performing sensitivity analysis at every iteration, placing significant demands on computing resources and performance. Therefore, improving computational efficiency and scalability while ensuring accuracy has become a major challenge in TO research and applications.

To lower the entry barrier and promote widespread adoption of TO, researchers have published numerous educational studies and literature. A pioneering example is the 99-line MATLAB code by Sigmund (2001), which demonstrated the fundamentals of a two-dimensional SIMP algorithm in a concise and self-contained manner, profoundly impacting both TO education and research. Subsequently, improved versions of this educational code emerged continuously, including the more efficient 88-line version proposed by Andreassen, Clausen, Schevenels, Lazarov and Sigmund (2011) and the extended 3D implementation by Liu and Tovar (2014). These educational codes convey practical knowledge of TO in the simplest possible form and provide self-contained examples of basic numerical algorithms.

Meanwhile, efforts have also been made to leverage the advantages of open-source software development for addressing TO problems. For example, Chung, Hwang, Gray and Kim (2019) proposed a modular TO approach based on OpenMDAO (Gray, Hwang, Martins, Moore and Naylor, 2019), an open-source multi-disciplinary design optimization framework. They decomposed the TO problem into multiple components, where users provide forward computations and analytic partial derivatives, while OpenMDAO automatically assembles and computes total derivatives, enhancing code flexibility and extensibility. Subsequently, Gupta, Chowdhury, Chakrabarti and Rabczuk (2020) developed a parallel-enabled TO implementation based on the open-source finite element software FEniCS (Alnæs, Blechta, Hake, Johansson, Kehlet, Logg, Richardson, Ring, Rognes and Wells, 2015), demonstrating its potential for handling large-scale problems. More recently, Ferro and Pavanello (2023) advanced this direction by introducing a concise and efficient TO implementation in just 51 lines of code. Their work utilized FEniCS for modeling and FEA, Dolfin Adjoint for automatic sensitivity analysis, and Interior Point OPTimizer for optimization, greatly simplifying the implementation process. These projects provided standardized interfaces enabling convenient integration with existing FEA tools, thereby lowering implementation complexity. However, despite improvements in modularity achieved by these open-source software packages, challenges in terms of functionality extension and flexibility remain when dealing with complex engineering applications.

To accelerate the development of TO, automating sensitivity analysis has become a critical step. This involves automatically computing derivatives of objectives, constraints, material models, projections, filters, and other components with respect to the design variables. Currently, the common practice involves manually calculating sensitivities, which, despite not being theoretically complex, can be tedious and error-prone, often becoming a bottleneck in the development of new TO modules and exploratory research. Automatic differentiation (AD) provides an efficient and accurate approach for evaluating derivatives of numerical functions (Griewank and Walther, 2008). By decomposing complex functions into a series of elementary operations (such as addition and multiplication), AD accurately computes derivatives of arbitrary differentiable functions. In TO, the Jacobian matrices represent sensitivities of objective functions and constraints with respect to design variables, and software can automate this process, relieving developers from manually deriving and implementing sensitivity calculations. With its capability of easily obtaining accurate derivative information, AD offers significant advantages in design optimization, particularly for highly nonlinear problems.

In recent years, the use of AD in TO has gradually increased. For instance, Nørgaard, Sagebaum, Gauger and Lazarov (2017) employed the AD tools CoDiPack and Taped to achieve automatic sensitivity analysis in unsteady flow TO, significantly enhancing computational efficiency. Chandrasekhar, Sridhara and Suresh (2021) leveraged JAX (Bradbury, Frostig, Hawkins, Johnson, Leary, Maclaurin, Necula, Paszke, VanderPlas, Wanderman-Milne et al., 2018), a high-performance Python library, to implement AD within density-based TO, efficiently solving classical topology optimization problems such as compliance minimization.

Although the programs described above—including early educational codes, open-source software implementations, and initial frameworks incorporating AD—have significantly promoted the adoption and development of TO methods, they typically employ a procedural programming paradigm. This approach divides the numerical computation process into multiple interdependent subroutines, leading to a tightly coupled relationship between analysis modules (e.g., FEA) and optimization modules. Such tightly coupled architectures limit code extensibility and reusability: on one hand, the strong interdependencies between subroutines mean that even adding a new objective function or constraint often requires invasive modifications across multiple modules, increasing development time and potentially introducing new errors; on the other hand, this coupled architectural pattern makes it challenging to integrate topology optimization programs as standalone modules within multidisciplinary design optimization (MDO) frameworks or system-level engineering processes. For instance, in aerospace applications, TO modules need seamless integration with other disciplines (such as fluid mechanics or thermodynamics), yet tightly coupled architectures typically result in complicated and inefficient integration procedures, hindering the application of TO in more complex scenarios. Consequently, designing an architecture that decouples analysis from optimization, thereby enhancing extensibility and reusability without sacrificing algorithmic accuracy, has become an important challenge that urgently needs addressing in the TO community.

## 2.

Figure 1:

Table 1

3.

## CRedit authorship contribution statement

Liang He: . : .

## References

- Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N., 2015. The fenics project version 1.5. Archive of numerical software 3.
- Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B.S., Sigmund, O., 2011. Efficient topology optimization in matlab using 88 lines of code. Structural and Multidisciplinary Optimization 43, 1–16.
- Bendsøe, M.P., Sigmund, O., 2004. Topology optimization by distribution of isotropic material. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 1–69. URL: [https://doi.org/10.1007/978-3-662-05086-6\\_1](https://doi.org/10.1007/978-3-662-05086-6_1), doi:10.1007/978-3-662-05086-6\_1.
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al., 2018. Jax: composable transformations of python+ numpy programs .
- Chandrasekhar, A., Sridhara, S., Suresh, K., 2021. Auto: a framework for automatic differentiation in topology optimization. Structural and Multidisciplinary Optimization 64, 4355–4365.
- Chung, H., Hwang, J.T., Gray, J.S., Kim, H.A., 2019. Topology optimization in openmdao. Structural and multidisciplinary optimization 59, 1385–1400.
- Ferro, R.M., Pavanello, R., 2023. A simple and efficient structural topology optimization implementation using open-source software for all steps of the algorithm: Modeling, sensitivity analysis and optimization. CMES-Computer Modeling in Engineering & Sciences 136.
- Gray, J.S., Hwang, J.T., Martins, J.R., Moore, K.T., Naylor, B.A., 2019. Openmdao: An open-source framework for multidisciplinary design, analysis, and optimization. Structural and Multidisciplinary Optimization 59, 1075–1104.
- Griewank, A., Walther, A., 2008. Evaluating derivatives: principles and techniques of algorithmic differentiation. SIAM.
- Gupta, A., Chowdhury, R., Chakrabarti, A., Rabczuk, T., 2020. A 55-line code for large-scale parallel topology optimization in 2d and 3d. arXiv preprint arXiv:2012.08208 .
- Liu, K., Tovar, A., 2014. An efficient 3d topology optimization code written in matlab. Structural and multidisciplinary optimization 50, 1175–1196.
- Nørgaard, S.A., Sagebaum, M., Gauger, N.R., Lazarov, B.S., 2017. Applications of automatic differentiation in topology optimization. Structural and Multidisciplinary Optimization 56, 1135–1146.
- Sigmund, O., 2001. A 99 line topology optimization code written in matlab. Structural and multidisciplinary optimization 21, 120–127.