A 55-line code for large-scale parallel topology optimization in 2D and 3D

Abhinav Gupta[a,1,*], Rajib Chowdhury[a,1,*], Anupam Chakrabarti[a,2], Timon Rabczuk[b]

[a]Department of Civil Engineering, Indian Institute of Technology Roorkee, India
[b]Institute of Structural Mechanics, Bauhaus-Universität Weimar, 99423 Weimar, Germany

Abstract

This paper presents a 55-line code written in python for 2D and 3D topology optimization (TO) based on the open-source finite element computing software (FEniCS), equipped with various finite element tools and solvers. PETSc is used as the linear algebra back-end, which results in significantly less computational time than standard python libraries. The code is designed based on the popular solid isotropic material with penalization (SIMP) methodology. Extensions to multiple load cases, different boundary conditions, and incorporation of passive elements are also presented. Thus, this implementation is the most compact implementation of SIMP based topology optimization for 3D as well as 2D problems.

Utilizing the concept of Euclidean distance matrix to vectorize the computation of the weight matrix for the filter, we have achieved a substantial reduction in the computational time and have also made it possible for the code to work with complex ground structure configurations. We have also presented the code's extension to large-scale topology optimization problems with support for parallel computations on complex structural configuration, which could help students and researchers explore novel insights into the TO problem with dense meshes. Appendix A contains the complete code, and the website: `https://github.com/iitrabhi/topo-fenics` also contains the complete code.

Keywords: Topology optimization, FEniCS, Parallel computing, Scalability, Education

1. Introduction

The primary purpose of structural topology optimization is to generate an optimal design - by removing material from the design domain - which should effectively comply with its intended design objectives and, at the same time satisfies the constraints imposed upon it. The development of high-performance structural topology optimization methods would result in low-cost, lightweight, and high-performance structures. For

---

*[*]Corresponding author

Email addresses: `agupta1[at]ce.iitr.ac.in` (Abhinav Gupta), `rajibfce[at]iitr.ac.in`,
`rajibchowdhury[at]ce.iitr.ac.in` (Rajib Chowdhury), `anupam.chakrabart[at]ce.iitr.ac.in` (Anupam Chakrabarti),
`timon.rabczuk[at]uni-weimar.de` (Timon Rabczuk)

[1]Tel.: (+91) 1332-285612

[2]Tel.: (+91) 1332-285844

the first time, Bendsøe and Kikuchi [20] presented the theory for topology optimization of continuum structures. The initial works were based on the homogenization or micro-structure approach. This approach provides criteria for topology optimization using composite materials to describe different material properties where each element is a micro-structure for a given homogeneous, isotropic material. However, the determination and evaluation of optimal micro-structures and their orientations are cumbersome [20].

In contrast, the Solid Isotropic Material with Penalization(SIMP) approach proposed by Bendsøe [18], Zhou and Rozvany [75], Mlejnek [51] suggest a relation between the design density variable and the material property based on the power-law, which results in improving the convergence of the solution. Because of the ease of implementation and straightforward description of the design density variable, this approach has grown in popularity among the research community. It is generally the preferred way to learn and understand the working philosophy of topology optimization. Gradient-based optimization algorithms have become the preferred way to solve the smooth and differentiable topology optimization problem represented by the SIMP approach[21, 63, 62, 64, 66, 65]. Topology optimization method has also been recently studied with the iso-geometric approach [37, 54, 32, 35] and mixed FEM approach [53, 52]

The classical 99-line MATLAB code for topology optimization of continuum structures [63] lead to a plethora of efficient implementations of topology optimization in various programming languages utilizing different methods. The initial code - which was based on SIMP method - has helped many researchers to develop their own implementations with SIMP [38, 67, 12, 60, 69, 43], Level Set[72, 24, 30, 57, 73, 36, 56, 58, 32, 33, 46], reaction-diffusionseongReactiondiffusionEquationBased2018a,jeongStructuralDesignConsidering2019, Bi-Directional Evolutionary Structural Optimization[77, 76, 49] and Moving Morphable Component[74]. Further Schmidt and Schulz [60], Aage et al. [5], Aage and Lazarov [1], Aage et al. [3], Amir et al. [11] have proposed efficient and parallelizable codes to carry out large scale topology optimization simulations. Recently a new generation 99 line MATLAB code was presented by Ferrari and Sigmund [29]. Readers are referred to [73] for a comprehensive review of all the educational codes available in literature. These codes have proved to be an integral part of the learning process for many researchers who wish to understand and implement topology optimization in their work.

The attractive primary property of these codes is that they are easy to understand and modify. However, to achieve readability and compactness, most of the proposed codes are limited to specific problems. Works by Zuo and Xie [77], Chen et al. [26] have helped expand the applicability of the codes to complex problems using ABAQUS and ANSYS. These works have made it possible to take advantage of the linear/nonlinear, static/dynamic FEA capacities, and meshing techniques present in the commercial package but then loose on the flexibility provided by custom implementations. Further, this limits the extension of these codes by the researches who do not have access to the commercial packages.

In this paper, we have presented a 55-line code for carrying out SIMP based topology optimization in FEniCS. The intention of this code is to stay true to the the original so that researchers and students

who are already familiar with the topic find it easy to understand an implement in their workflow. We propose the use of Euclidean distance matrices to vectorize the computation of distance matrices for filter. This results in substantial reduction in computational time for evaluation of the filter. Moreover, this method supports complex structural configurations. We also present the extension of the code to large scale complex engineering structure with support for parallelization. Thus, this compact code could help students and researchers learn and understand the concept of TO on their laptops and then extend the same to handle complex 3D structures on large high-performance computers.

This manuscript is structured as follows. Section 2 addresses the theoretical aspects of the variational formulation of the linear elasticity problem and discusses the SIMP approach. Section 3 presents the mathematical model of the topology optimization problem. We discuss the continuous and discontinuous Lagrange elements used for discretization and then present the problem's weak form. We then discuss the optimization approach, the mesh independency filter, and the Euclidean distance matrix (used to vectorize the loops for calculating the distance matrix for the mesh independency filter). In Section 4, we present a detailed step by step description of our FEniCS implementation. Section 5 includes the model extensions of the code to a 3D problem, different load cases, different boundary conditions, and inclusion of passive elements. Finally, we summarize the findings, in Section 7.

2. Topology optimization formulation

We consider an arbitrarily shaped linear elastic body Ω_{mat} which is part of a large reference domain $\Omega \subset \mathbb{R}^{\delta}, \delta = 2, 3$, having boundary Γ, as illustrated in Fig. 1. The reference domain Ω (ground structure) is chosen to allow for a definition of applied loads and boundary conditions. The displacement at a point $\boldsymbol{x} \in \Omega$ is represented by $\boldsymbol{u}(\boldsymbol{x})$. The Dirichlet and Neumann boundaries of the domain Ω are represented by Γ_D and Γ_N, respectively. Let $\boldsymbol{F_B} \subset \mathbb{R}^{\delta}, \delta = 2, 3$ represent the body force vector and $\boldsymbol{F_T} \subset \mathbb{R}^{\delta}, \delta = 2, 3$ represent the traction force on the on the traction part $\Gamma_T \subset \Gamma$, of the boundary. The ground structure could consist of zones with no material(such as holes), or zones with fixed material. FEniCS works based on the variational formulation of the problem, and thus in this section we discuss the variational approach to topology optimization problem.

2.1. Governing equations and weak form

Elastic strain energy is the energy stored at the time of deformation inside an elastic body. When an isotropic linear elastic solid is deformed under small strain condition, the strain energy density function $\psi_0(\boldsymbol{\epsilon})$ is defined as:

$$\psi_0(\boldsymbol{\epsilon}) = \frac{\lambda}{2}(tr[\boldsymbol{\epsilon}])^2 + \mu tr[\boldsymbol{\epsilon}^2] \tag{1}$$
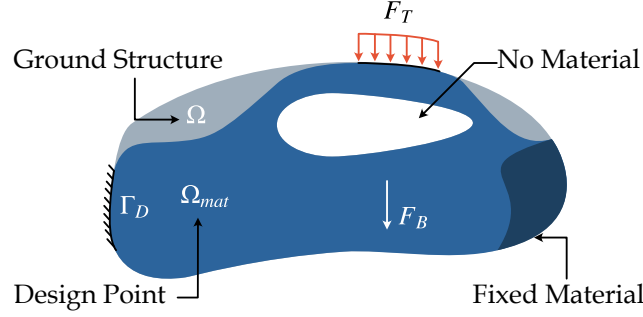
3

Figure 1: Structural topology optimization problem definition. The ground structure is chosen to allow for a definition of applied loads and boundary conditions. It could consist of zones with no material or zones with fixed material.

where $\mu, \lambda > 0$ are the Lamé parameters, $\epsilon = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) = sym(\nabla \boldsymbol{u})$, is defined as the symmetric part of the tensor $\nabla \boldsymbol{u}$. The total strain energy is thus defined as:

$$SE = \int_{\Omega} \psi_0(\boldsymbol{\epsilon}) \mathrm{d}\Omega \tag{2}$$

The work potential is defined as the potential of the external loads, which is the negative of the work done on the system. The work potential is given by

$$WP = -\int_{\Omega} \boldsymbol{u}.\boldsymbol{F_B} \mathrm{d}\Omega - \int_{\Gamma} \boldsymbol{u}.\boldsymbol{F_T} \mathrm{d}\Gamma \tag{3}$$

The potential energy of the system is defined as the sum of elastic energy and work potential. The total potential energy for a deformed body is thus given by:

$$U = \int_{\Omega} \psi_0(\boldsymbol{\epsilon}) \mathrm{d}\Omega - \int_{\Omega} \boldsymbol{u}.\boldsymbol{F_B} \mathrm{d}\Omega - \int_{\Gamma} \boldsymbol{u}.\boldsymbol{F_T} \mathrm{d}\Gamma \tag{4}$$

For calculating displacement field the principle of minimum potential energy is used. Thus, by taking the first variation of U and then applying the fundamental lemma of the calculus of variation, we get:

Governing differential equation:

$$\nabla \cdot \boldsymbol{\sigma} = -\boldsymbol{F_B} \quad in \quad \Omega \tag{5}$$

Natural boundary condition:

$$(\boldsymbol{\sigma}.\boldsymbol{n}) = \boldsymbol{F_T} \quad on \quad \Gamma \tag{6}$$

In the SIMP method, we introduce a pseudo density $d \in [0, 1]$ into Eq. (5) to model the distribution of the material within the ground structure.

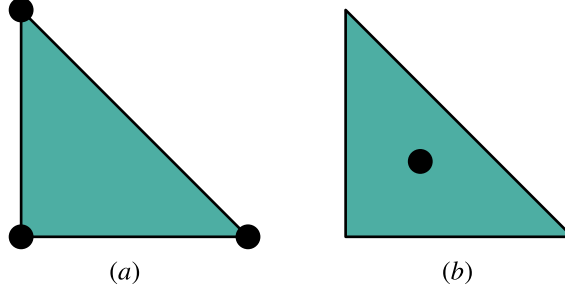$$d^p \, \nabla.\boldsymbol{\sigma} = \boldsymbol{F_B} \tag{7}$$

4

Figure 2: Finite elements - 2D. (a)three noded triangular Lagrangian element with two degrees of freedom per node used to model the displacement vector. (b) one noded discontinuous Lagrangian element used to model the density parameter.

In Eq. (7), p is the penalty parameter that helps in enforcing a $0/1$ density distribution by significantly reducing the effect of intermediate values. Based on the power-law approach we could now define the topology optimization problem, where the objective is to minimize the strain energy and the volume is constrained.

$$
\begin{aligned}
\underset{d}{\text{minimize}} \quad & f(d) = d^p U \\
\text{subject to} \quad & g(d) = \int_\Omega d.V_0 \mathrm{d}\Omega - k.V_0 \leq 0 \\
& \forall d \in \mathbb{D}, \mathbb{D} = \{d \in L^2(\Omega) : d_{min} < d \leq 1\}
\end{aligned} \tag{8}
$$

In Eq. (8), U is the strain energy of the domain and V_0 is the initial volume. k is the prescribed volume fraction.

3. Numerical Approximation

The linear elasticity model, as described by the differential equation (refer Eq. (5)), is discretized by the finite element method. We start with a discussion on weak form and the discrete form of the system. Then we discuss the optimization algorithm and the filter used in the method to avoid the checkerboard pattern.

3.1. Spatial discretization

We use first degree C^0 Lagrangian finite elements to discretize the domain and the primary field variable. The density parameter is discretized with the help of zeroth degree, C^{-1} discontinuous Galerkin elements. In the 2 dimensional cases, the three noded triangular Lagrangian finite elements have two degrees of freedom per node, whereas the discontinuous Lagrangian element of degree zero have a single value per element and are represented as shown in Fig. 2. For the 3 dimensional case, the four noded tetrahedral Lagrangian finite element have three degrees of freedom per node, whereas the discontinuous Lagrangian elements of degree zero have a single value per element and are represented as shown in Fig. 3.
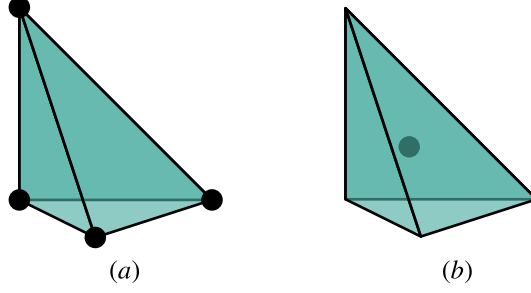
Figure 3: Finite elements - 3D. (a) four noded tetrahedral Lagrangian element with three degrees of freedom per node used to model the displacement vector. (b) one noded discontinuous Lagrangian element used to model the density parameter.

Remark. Triangular and tetrahedral elements are used for the current implementation as they are well supported in FEniCS, whereas the support for quadrilateral and hexahedral elements, even though present is somewhat limited. For simple formulations, user could switch to quadrilateral and hexahedral elements to reduce the computational expense.

3.1.1. Weak form

The weak form of the system is derived by first multiplying the governing differential equation (refer Eq. (5)) by admissible test function ($\boldsymbol{v}$) and then modification of the equations with the application of Green's theorem. The weak form of the system is defined as: Find $\boldsymbol{u} \in \boldsymbol{U}$ such that

$$\int_\Omega d^p \boldsymbol{\sigma}(\boldsymbol{u}) : \nabla \boldsymbol{v} \; d\Omega = 0 \quad \forall \boldsymbol{v} \in \widehat{\boldsymbol{U}} \tag{9}$$

In Eq. (9), one usually uses only the symmetric part of the test function (virtual strain, $\boldsymbol{\epsilon}(\boldsymbol{v}) = 1/2(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T)$ as $\int_\Omega \boldsymbol{\sigma}(\boldsymbol{u}) : (\nabla \boldsymbol{v})_{skewsym} \; d\Omega = 0$.

$$\int_\Omega d^p \boldsymbol{\sigma}(\boldsymbol{u}) : \boldsymbol{\epsilon}(\boldsymbol{v}) \; d\Omega = 0 \quad \forall \boldsymbol{v} \in \widehat{\boldsymbol{U}} \tag{10}$$

The trial ($\boldsymbol{U}$) and test ($\widehat{\boldsymbol{U}}$) spaces are defined as

$$(\boldsymbol{U}, \widehat{\boldsymbol{U}}) = \{(\boldsymbol{u}, \boldsymbol{v}) \in H^1(\Omega) : \boldsymbol{u} = \overline{\boldsymbol{u}} \; \& \; \boldsymbol{v} = 0 \; on \; \Gamma_D\} \tag{11}$$

In Eq. (11), $H^1(\Omega)$ denotes the usual Hilbert space over the domain Ω and $\overline{\boldsymbol{u}}$ is the prescribed displacement.

3.1.2. Discrete form

The weak form of the system as presented by Eq. (9) is discretized by using C^0 Lagrangian elements for displacement vector and C^{-1} discontinuous Lagrangian elements for density. The discrete form of the

problem is thus defined as: Find $(\boldsymbol{u_h}) \in \boldsymbol{U_h} \subset (\boldsymbol{U})$ such that

$$\int_\Omega d_h^p \boldsymbol{\sigma}(\boldsymbol{u_h}) : \boldsymbol{\epsilon}(\boldsymbol{v}) \; d\Omega = 0 \quad \forall \boldsymbol{v} \in \widehat{\boldsymbol{U}}_h \subset \widehat{\boldsymbol{U}} \tag{12}$$

We follow the Bubnov-Galerkin formulation, where the test and trial functions belong to the same function space. Continuous field variables are thus approximated as:

$$\boldsymbol{u}(\boldsymbol{x}) \approx \boldsymbol{u}_h(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{N}_i \boldsymbol{u}_i \tag{13}$$

where n denotes the number of nodes in the mesh, $\boldsymbol{u}_i$ and N_i are the displacement vector and basis function corresponding to i^{th} node, respectively.

3.2. Optimization algorithm

The optimization problem as described in Eq. (8) is solved using the standard optimality criteria (OC) method. The OC method states that convergence is achieved when the KKT condition is satisfied [43],

$$\frac{\partial f(d)}{\partial d_e} + \lambda \frac{\partial g(d)}{\partial d_e} = 0 \tag{14}$$

In Eq. (14) λ is the Lagrange multiplier associated with the constraint $g(d)$. This optimality condition can be expressed as $B_e = 1$, where

$$B_e = -\frac{\partial f(d)}{\partial d_e} \left(\lambda \frac{\partial g(d)}{\partial d_e} \right)^{-1} \tag{15}$$

According to [19] a heuristic updating scheme for the design variables can be formulated as:

$$d_e^{\text{new}} = \begin{cases} \max\left(d_{min}, d_e - m\right) & \text{if } d_e B_e^\eta \leq \max\left(d_{min}, d_e - m\right) \\ \min\left(1, d_e + m\right) & \text{if } d_e B_e^\eta \geq \min\left(1, d_e + m\right) \\ d_e B_e^\eta & \text{otherwise} \end{cases} \tag{16}$$

In Eq. (16), m is a positive move-limit, and η is a numerical damping coefficient. The choice of $m = 0.2$ and $\eta = 0.5$ is recommended [19, 63].

3.2.1. The bisection method

We use the bisection method to find the Lagrange multiplier λ. The Lagrange multiplier should satisfy the constraint function $g(d)$, i.e. the new value of density parameter evaluated with Eq. (16) should satisfy the condition

$$g(d(\lambda)) = 0 \tag{17}$$

Numerically, this could be achieved by the bisection method in which we assume a lower bound and upper bound on the value of λ and calculate the value of constraint function at mid-point of that interval. If the convergence is satisfactory (i.e. the length of the interval is small), then stop the iteration and return

mid-point as solution. Otherwise, examine the sign of the constraint function and update the lower-bound with mid-point if the sign is positive else update the upper-bound with mid-point value.

The sensitivity of the objective function is defined as

$$\frac{\partial f(d)}{\partial d} = -pd^{p-1}U \tag{18}$$

The derivative of the volume constraint $g(d)$ in Eq. (8) with respect to the design variable d is given

$$\frac{\partial g(d)}{\partial d} = \int_\Omega V_0 \mathrm{d}\Omega \tag{19}$$

3.2.2. Mesh-independency filter

The solution of topology optimization problems is mesh-dependent. To circumvent this issue, researchers have proposed sensitivity filters that ensure mesh-independency. The filter works by modifying the sensitivity of a specific element's objective function by taking the weighted average of the element sensitivities in a fixed neighborhood.

$$\frac{\partial(d_a E_a)}{\partial d_a} = \frac{1}{\sum_{b=1}^{N} W_b} \sum_{b=1}^{N} W_b \frac{\partial(d_b E_b)}{\partial d_b} \tag{20}$$

$$\frac{\partial(E_a)}{\partial d_a} = \frac{1}{d_a \sum_{b=1}^{N} W_b} \sum_{b=1}^{N} W_b d_b \frac{\partial(E_b)}{\partial d_b} \tag{21}$$

Here the letter $'a'$ represents the element under consideration (marked as red in Fig. 4) and the letter $'b'$ represents the elements in the neighborhood of element $'a'$ (marked with red X marks in the Fig. 4). The elements in the neighborhood are identified as the ones whose center lies inside a circle of radius r_{min} as shown in the Fig. 4. W_b is the weight factor (also known as the convolution operator) assigned to the element $'b'$, which decays linearly with increasing distance from element $'a'$ and is defined as:

$$W_b = r_{min} - dist(b,a), \ \{b \in \mathbb{N} | dist(b,a) \le r_{min}\}, \ b = 1, ..., N. \tag{22}$$

In Eq. (22), $dist(b,a)$ is the distance operator that calculates the distance between the midpoints of the element $'b'$ and element $'a'$. The weight factor is zero outside the zone defined by the circle with a radius r_{min}.

3.3. Numerical implementation

The numerical simulations are performed using the python interface of the open-source scientific computing platform FEniCS [45, 10, 44, 40, 55, 9, 39, 8, 16, 22]. To solve the algebraic equations, FEniCS is configured with PETSc [15, 17, 6] as the linear algebra backend. The output of the topology optimization problem is written to a XDMF file and Paraview [7] is used to visualize the output.

All the two dimensional analyses were performed on a Windows laptop equipped with an Intel Core i7 processor with a clock frequency of 2.2GHz having six cores and 16GB of RAM. The three dimensional
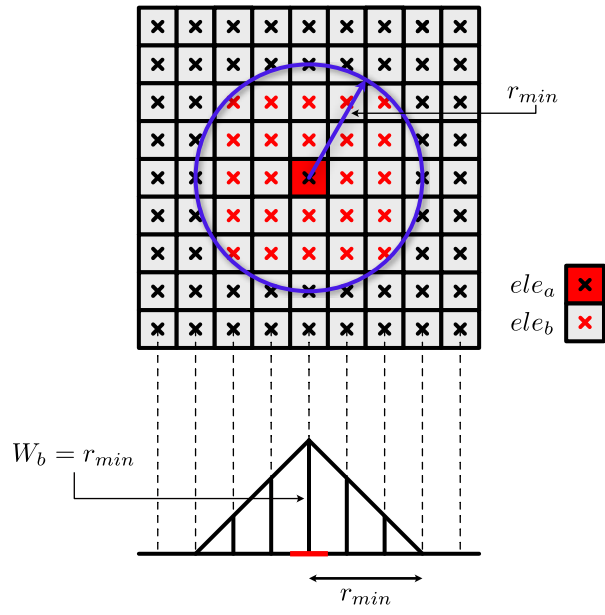
Figure 4: Mesh-independency filter. (top)Elements (ele_b) in the neighborhood of ele_a considered for the density filtering. (bottom) Variation of the weight factor (W_b) with increasing distance from ele_a. Note that the weight factor is zero outside the filter zone marked by r_{min}
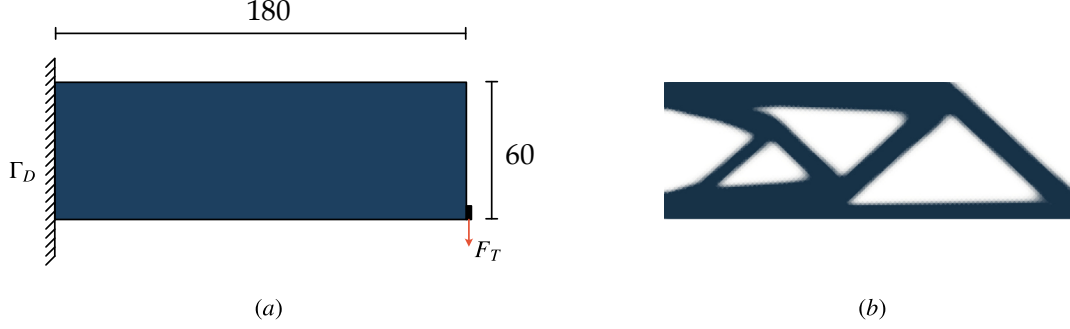
Figure 5: Cantilever beam problem. (a)The design domain, boundary conditions, and external load for the optimization of a cantilever beam. (b) Topologically optimized structure.

analysis were performed on a workstation equipped with two Intel Xeon Platinum 8260 M processor with a clock frequency of 2.4 GHz having 24 cores per processor amounting to a total of 48 cores and memory is 128 GB.

4. FEniCS implementation

In this section, the 55-line FEniCS code for the energy minimization problem is explained in detail. Several assumptions were made to simplify the code in the previous compact codes (99-line and 88-line). Thus, the codes were limited to a certain kind of problem, and changing them to 3D would require significant modification of the codebase.

The intention of the current implementation is to take advantage of the highly popular programming language python, which is known for its compactness and readability, and to create the most compact implementation of topology optimization by utilizing the open-source FEM package FEniCS. FEniCS allows us to describe the FEM problem in terms of variational principles, keeping the python implementation as close to the mathematical description as possible. This also helps us to improve the readability of the code when comparing it to the mathematical description. The code is called from the python prompt employing the following lines:

```
from topopt import main
main(nelx=180, nely=60, volfrac=0.5, penal=3.0, rmin=2.0)
```

where `nelx` and `nely` are the number of elements in the horizontal and vertical direction, respectively, `volfrac` is the prescribed volume fraction k, `penal` is the penalization power p, `rmin` is the filter radius r_{min}. Fig. 1(a) shows the domain and boundary conditions for a cantilever beam as an example that would be used to describe the code.

4.1. Header

The first line imports all the key classes and methods necessary for solving a finite element variational form in FEniCS. The dolfin library is a part of the FENiCS package, which in itself is a wrapper for the C++ library DOLFIN that provides efficient and highly scalable C++ classes for finite element computing. Thus, the package benefits from the speed and efficiency of C++ while keeping the programming for the end-user in an easy to understand and highly readable programming language (python). The second line imports the two packages necessary for performing matrix operations.

```
from dolfin import *
import numpy as np, sklearn.metrics.pairwise as sp
```

4.2. Function declaration

In FEniCS we solve a problem by defining the variational formulation with Unified Form Language (UFL). This allows us to define the formulas in python that are very similar to the mathematical description and thus makes the code highly readable. The Cauchy stress tensor and the strain energy of the system are described as

$$\sigma = \lambda \operatorname{tr}(\epsilon)I + 2\mu\epsilon \tag{23}$$

$$\psi_0(\epsilon) = \frac{\lambda}{2}(tr[\epsilon])^2 + \mu tr[\epsilon^2] \tag{24}$$

In Eqs. (23) and (24), strain ($\epsilon = \frac{1}{2}(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T) = sym(\nabla \boldsymbol{u})$), is defined as the symmetric part of the part of $\nabla \boldsymbol{u}$. Note the very close correspondence between the mathematical formulas and their equivalent UFL descriptions. This is the core strength of FEniCS that we wish to exploit to the fullest to make the most compact code of topology optimization.

```
sigma = lambda _u: 2.0 * mu * sym(grad(_u)) + lmbda * tr(sym(grad(_u))) * Identity(len(_u))
psi = lambda _u: lmbda / 2 * (tr(sym(grad(_u))) ** 2) + mu * tr(sym(grad(_u)) * sym(grad(_u)))
```

4.3. Finite element analysis

Most of the codes available in the literature follow the approach of the matrix method, wherein they derive the stiffness matrix and force vectors and then code them into the programming language. FEniCS is based on the variational approach, where the user needs to specify the variational form with UFL and then use the built-in operators and methods to generate all the required matrices and vectors automatically. This process is initiated by creating function spaces over the mesh and then defining functions in that function space. The supports and loads are then defined based on the function space and functions. Finally, a linear variational problem and a solver are defined.

11

4.3.1. Discretization the problem

We first start discretizing the problem by defining the domain under consideration with the built in function `RectangularMesh`. The mesh is made up of triangular elements. The parameters `nelx` and `nely` tell the program to divide the domain into `nelx` and `nely` unit rectangles in x and y directions respectively. Each of the unit rectangle is then divided into two triangles and the last argument decides the direction for the division of the rectangles.

```
mesh = RectangleMesh(Point(0, 0), Point(nelx, nely), nelx, nely, "right/left")
U = VectorFunctionSpace(mesh, "CG", 1)
D = FunctionSpace(mesh, "DG", 0)
u, v = TrialFunction(U), TestFunction(U)
u_new, density, density_old = Function(U), Function(D), Function(D)
```

As discussed in Section 3.1 the topology optimization problem consists of two primary variables, the displacement vector (u), and the density parameter (d) which is constant over the element and is discontinuous over the edges. Thus, to discretize the problem with finite element method we need to define two function spaces over the mesh. The first one is defined using the CG elements over the mesh and creating a vector function space of degree one. This implies that the element is a 3-noded linear triangle with two degrees of freedom at each node. For the density parameter we define a function space with DG elements. Once we have defined the function space we can define the test and trial functions over the space which are required to define the weak form of the variational problem. Then we define functions which are used to store the solution to the problem.

4.3.2. Definition of supports

The supports are first identified inside the mesh with the `CompiledSubdomain` method that takes in a condition in C++ syntax and then at compile time converts the expression into an efficient compiled C++ function. In the case of a cantilever beam fixed at x=0 the support is defined by the following command.

```
support = CompiledSubDomain("near(x[0], 0.0, tol) && on_boundary", tol=1e-14)
bcs = [DirichletBC(U, Constant((0.0, 0.0)), support)]
```

The C++ expression takes a point vector `x` as input, where the vector elements represent the x, y, and z coordinates. The support nodes are identified by a conditional statement based on the point vector. The Dirichlet boundary condition is then defined - over the nodes identified by the function `support` - by assigning a value of zero to the components of the displacement vector over the vector function space U ($u = 0$ *on* Γ_D).

4.3.3. Definition of loads

The load boundary is identified with the help of `load_marker`. This is achieved in FEniCS with the help of `MeshFunction` method. We mark the mesh entities with integer markers. We then use those integers in the measures (`dx` and `ds`) to identify the region in our variational form. In UFL, `dx` represents the integral over the whole cell whereas `ds` represents the integral over the exterior facet of the cell. In our case suppose the load is applied over the bottom two elements of the mesh. This is achieved by first identifying the bottom two elements with the load marker and then updating the measure `ds` to account for the markers. After that we define the right hand $\boldsymbol{f}.\boldsymbol{v}\ ds$ with the command.

```
load_marker = MeshFunction("size_t", mesh, 1)
CompiledSubDomain("x[0]==l && x[1]<=2", l=nelx).mark(load_marker, 1)
ds = Measure("ds")(subdomain_data=load_marker)
F = dot(v, Constant((0.0, -1.0))) * ds(1)
```

4.3.4. Defining the variational problem and solver

The bilinear form of the systems as described by Eq. (12) is defined first and then we define the `LinearVariationalProblem` and `LinearVariationalSolver` objects. Defining the problem and solver objects allows the program to automatically update the primary field variable and thus inturn the discrete form of the system. Thus, we can solve the system inside a loop by simply calling the `solve()` method over the solver object. The call to `solve()` will create the stiffness matrix and the force vector and will not dispose of them immediately after execution, which allows the program to reuse them within the loop.

```
K = inner(density ** penal * sigma(u), grad(v)) * dx
problem= LinearVariationalProblem(K, F, u_new, bcs)
solver = LinearVariationalSolver(problem)
# FE-ANALYSIS -------------------------------------------------
solver_disp.solve()
```

FEniCS utilizes PETSc as its linear algebra backend and by default uses the sparse LU decomposition ("lu") method, and the actual software that is called is then the sparse LU solver from UMFPACK (which PETSc has an interface to). We could easily change the solver parameters and design the solver as per our requirements by accessing the method parameters over the solver object.

```
solver.parameters["linear_solver"] = "gmres"
solver.parameters["preconditioner"] = "ilu"
```

4.4. Optimality criteria based optimizer

We initialize the density vector with `volfrac` as the initial density. Note that we use `as_vector()` method over the density function to access the underlying array of values. The iterations start with the

finite element analysis of the linear variational problem. The solution to the problem is stored in the variable `u_sol`, which is then used to evaluate the objective function and sensitivity over each element of the mesh.

```
solver.solve()
objective = project(density ** penal * psi(u_sol), D).vector()[:]
sensitivity = -penal * (density.vector()[:]) ** (penal - 1) * project(psi(u_sol), D).vector()[:]
```

The optimality criteria based update of the density vector is done on lines 46 - 51. The bounds for the calculation of Lagrange multiplier by the bisection method are initialized on line 46 and the actual loop starts on line 47.

```
l1, l2, move = 0, 100000, 0.2
while l2 - l1 > 1e-4:
    l_mid = 0.5 * (l2 + l1)
    density_new = np.maximum(0.001,np.maximum(density.vector()[:] - move, np.minimum(1.0, np.minimum(density.
        vector()[:] + move, density.vector()[:] * np.sqrt(-sensitivity / l_mid)))))
    l1, l2 = (l_mid, l2) if sum(density_new) - volfrac * mesh.num_cells() > 0 else (l1, l_mid)
density.vector()[:] = density_new
```

4.5. Mesh-independency filtering

The modification of the sensitivities by the application of the sensitivity filter described by Eq. (21) is a linear operation and this could be converted to a matrix product as defined by Eq. (26). The matrix W is the weight matrix. The elements of the matrix satisfy the condition presented by Eq. (22) and Fig. 4 . First we retrieve the midpoints of all the elements within the mesh and then evaluate the Euclidean distance matrix for the midpoint vector. In the case of topology optimization, suppose we have a collection of vectors $\{\boldsymbol{x}_i \in \mathbb{R}^d : i \in \{1, ..., N\}\}$ containing the coordinates of midpoints $(\boldsymbol{x}_i)$ of all the elements within the mesh. The Euclidean distance matrix is a $N \times N$ matrix that contains the pairwise distance between all the elements of a vector of length N, and is defined as:

$$\boldsymbol{D}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j) = \|\mathbf{x}_i\|_2^2 - 2\mathbf{x}_i^T \mathbf{x}_j + \|\mathbf{x}_j\|_2^2 \tag{25}$$

This gives us the pairwise distance between all the domain elements, or more specifically a single element $\boldsymbol{D}_{ab}$ represents the distance of ele_a from ele_b. Then we apply the condition that the elements outside the zone of influence should have a weight equal to zero. i.e., the elements whose distance > rmin should have a weight equal to zero.

Remark. By evaluating the weight matrix based on vector based calculations we reduced a lot of computational cost that is associated with evaluation of the same in a nested loop. The euclidean distance matrix is a symmetric matrix and thus there exists multiple algorithms to further reduce the computational cost

14

associated with their evaluation on parallel system. The interested reader is referred to Angeletti et al. [13], Li et al. [42] for the same.

$$
\left\{\begin{array}{c} \frac{\partial E_1}{\partial d_1} \\ \vdots \\ \frac{\partial E_n}{\partial d_n} \end{array}\right\} = \left(\left[\begin{array}{ccc} W_{11} & \cdots & W_{1n} \\ \vdots & \ddots & \vdots \\ W_{n1} & \cdots & W_{nn} \end{array}\right] \cdot \left(\left\{\begin{array}{c} d_1 \\ \vdots \\ d_n \end{array}\right\} \odot \left\{\begin{array}{c} \frac{\partial E_1}{\partial d_1} \\ \vdots \\ \frac{\partial E_n}{\partial d_n} \end{array}\right\}\right)\right) \odot \left(\left\{\begin{array}{c} \frac{1}{d_1} \\ \vdots \\ \frac{1}{d_n} \end{array}\right\} \odot \left\{\begin{array}{c} \frac{1}{\sum_{b=1}^{n} W_{1b}} \\ \vdots \\ \frac{1}{\sum_{b=1}^{n} W_{nb}} \end{array}\right\}\right) \quad (26)
$$

```python
# PREPARE DISTANCE MATRICES FOR FILTER ----------------------------
midpoint = [cell.midpoint().array()[:] for cell in cells(mesh)]
distance_mat = rmin - sp.euclidean_distances(midpoint, midpoint)
distance_mat[distance_mat < 0] = 0
distance_sum = distance_mat.sum(1)  # sum the row
# FILTERING/MODIFICATION OF SENSITIVITIES ----------------------
sensitivity = np.divide(distance_mat @ np.multiply(density.vector()[:], sensitivity), np.multiply(density.
    vector()[:], distance_sum))
```

Once we have the weight matrix, we evaluate the vector of the sum of weight for each element. The modification in sensitivity is then evaluated by just performing the matrix operations described by Eq. (26). Here $\odot$ represents the Hadamard product, and (.) represents the dot product. The distance matrix (weight matrix) and the distance sum vector do not vary during the iteration process, and thus they are evaluated and stored outside of the primary iteration loop.

5. Model Extensions

The original 99-line code by Sigmund [63] and the 88-line code by Andreassen et al. [12], the authors described how to extend their code to account for passive elements, other boundary conditions, and multiple load cases. In this section, we also describe these extensions based on our 55 lines of code. Besides, we show that the extension of the code to 3D problems still keeps the number of lines limited to 55 and requires a minimal change of the code.

5.1. Working with 3D problems

One of the amazing advantages of working with FEniCS is that it allows us to write a unified simulation code that can operate in $\mathbb{R}^d, d = 1, 2, 3$, since the variational formulation is independent of dimension. By making a few modifications to the code we can use the same 55-lines of code to run 3D simulations. This is hugely beneficial to the previous implementation as to work with 3D they required significant modification of the code base [12]. To change the code into a 3D cantilever problem, we just need to update the mesh and add the third dimension values to the boundary condition and load definitions. Thus, only the following change is required:
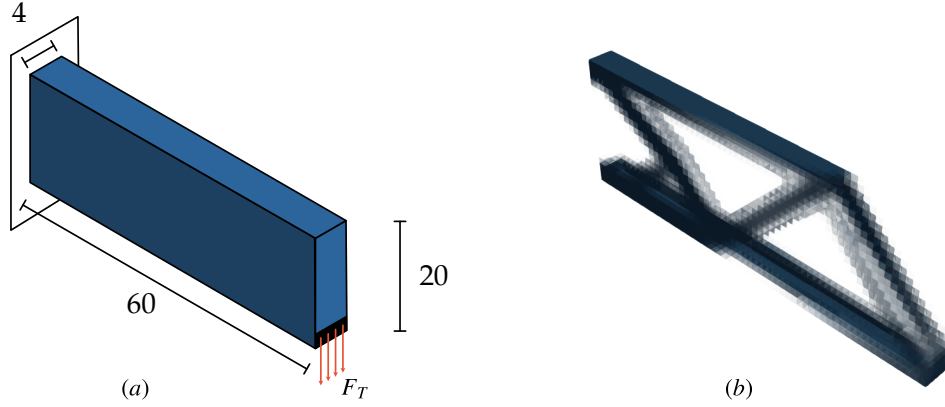
Figure 6: 3D Cantilever beam problem. (a)The ground structure with the applied external loads and boundary conditions for the optimization of a 3D cantilever beam. (b) The topologically optimized design.

```python
def main(nelx, nely, nelz, volfrac, penal, rmin):
    mesh = BoxMesh(Point(0.0, 0.0, 0.0), Point(nelx, nely, nelz), nelx, nely, nelz)
    bcs = [DirichletBC(U, Constant((0.0, 0.0, 0.0)), support)]
    F = dot(v, Constant((0.0, -1.0, 0.0))) * ds(1)
```

We need to add the number of element required in z-direction to the parameters of function definition. Changing the mesh constructor to `BoxMesh()`, creates a box partioned with (`nelx`, `nely`, `nelz`) elements. Fig. 6 shows the ground structure with the applied external loads and boundary conditions for the optimization of a 3D cantilever beam along with the optimized geometry achieved by running the command:

```python
from topopt import main
main(nelx=60, nely=20, nelz=4, volfrac=0.3, penal=3.0, rmin=1.5)
```

5.2. Other boundary conditions

The support elements are identified based on the conditional statements written in the C++ syntax. To add a prop to the other end of the cantilever beam we just need to identify the elements and then assign a Dirichlet boundary condition to it. This is achieved by adding the following lines of code:

```python
prop_support = CompiledSubDomain("near(x[0], l,tol) && near(x[1], 0, tol)", tol=1e-14, l=nelx)
prop_bc = DirichletBC(U.sub(1), Constant(0.0), prop_support, method="pointwise")
bcs.append(prop_bc)
```

The solution presented in Fig. 7 is then obtained by the following:

```python
from topopt import main
main(nelx=180, nely=60, volfrac=0.5, penal=3.0, rmin=3)
```
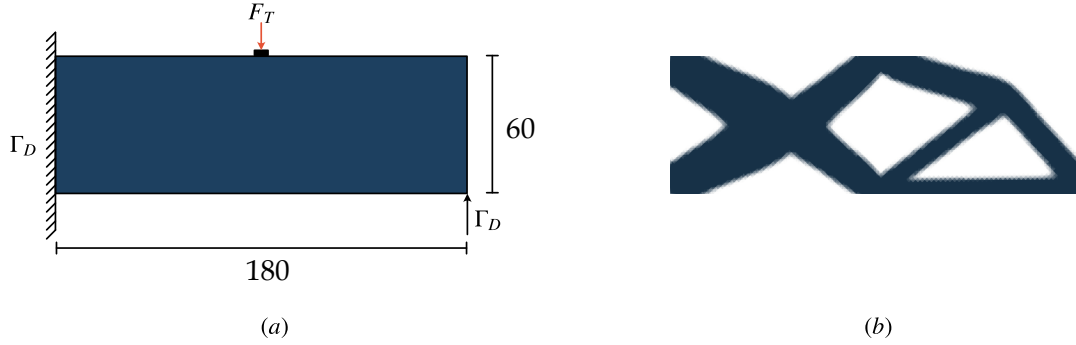
Figure 7: Propped cantilever beam problem. (a)The ground structure with the applied external loads and boundary conditions for the optimization of a propped cantilever beam. (b) Topologically optimized structure.
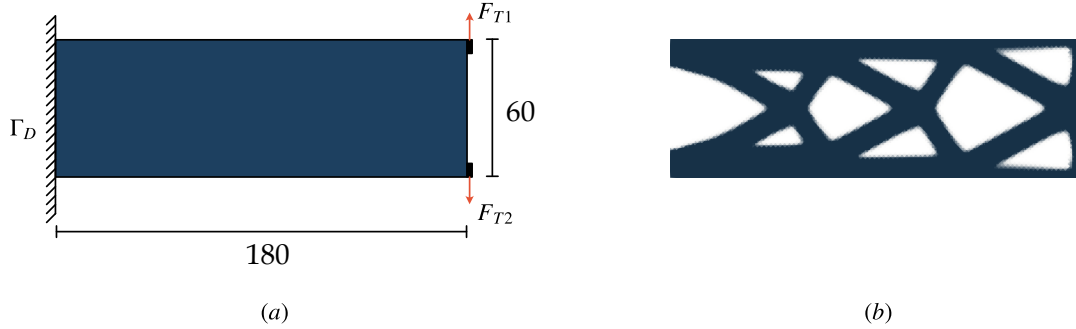

Figure 8: Cantilever beam with multiple load case. (a)The ground structure with the applied external loads and boundary conditions for the optimization of a cantilever beam with multiple load case. (b) Topologically optimized structure.

5.3. Multiple load cases

Adding multiple load cases to the problem is also achieved with less than 60 lines of code. In the two load case the change is made in line number 21 where we need to add a new marker for the second load. This is identified with the help of load marker 2. The information is then passed to the measure in a similar manner as before. We could then define loads on certain boundaries by using there specific markers.

```
CompiledSubDomain("x[0]==l && x[1]<=2", l=nelx).mark(load_marker, 1)
CompiledSubDomain("x[0]==l && x[1]>=h-2", l=nelx, h=nely).mark(load_marker, 2)
ds = Measure("ds")(subdomain_data=load_marker)
F1 = dot(v, Constant((0.0, -1.0))) * ds(1)
F2 = dot(v, Constant((0.0, +1.0))) * ds(2)
problem = [LinearVariationalProblem(K, F1, u_sol, bcs),LinearVariationalProblem(K, F2, u_sol, bcs)]
```

The objective function and the sensitivity is calculated based on the displacement field by solving the two variational problems and summing them up for further calculations. This is done by replacing the lines

17

38-42 with the following.

```python
objective, sensitivity = np.zeros(mesh.num_cells()), np.zeros(mesh.num_cells())
for i in range(2):
    # FE-ANALYSIS -----------------------------------------------
    solver = LinearVariationalSolver(problem[i])
    solver.solve()
    # OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS ------------------
    objective += project(density ** penal * psi(u_sol), D).vector()[:]
    sensitivity += -penal * (density.vector()[:]) ** (penal - 1) * project(psi(u_sol), D).vector()[:]
```

The solution presented in Fig. 8 is then obtained by the following:

```python
from topopt import main
main(nelx=180, nely=60, volfrac=0.5, penal=3.0, rmin=3)
```

5.4. Passive elements

To incorporate zones of fixed density, such as zones with no material (holes) or zones with fixed material, we define passive elements that have a fixed density throughout the iteration process. This is achieved with just additional five lines of code including the identification of the passive elements. The zone is identified and marked with the help of the Meshfunction class of FEniCS by adding the following lines of code.

```python
# DEFINE PASSIVE ELEMENTS -------------------------------------
circle = CompiledSubDomain('(x[0]-x0)*(x[0]-x0) + (x[1]-x1)*(x[1]-x1) <= r*r',x0=nelx/3, x1=nely/2, r=nely/4)
circle_marker = MeshFunction("size_t", mesh, mesh.topology().dim())
circle.mark(circle_marker, 1)
```

This information is then passed to the optimality criteria loop by adding the following line and then, explicitly setting their density value equal to minimum density in case of a hole.

```python
density_new[circle_marker.where_equal(1)] = 0.001
```

The solution presented in Fig. 9 is then obtained by the following:

```python
from topopt import main
main(nelx=180, nely=60, volfrac=0.5, penal=3.0, rmin=3)
```

5.5. Alternative optimizers

The optimality criteria based optimizer have been widely used in the literature because of its easy to understand formulation and straight forward implementation, but suffers from the lack of support for problems with multiple constrains. Another common optimizer that has been used successfully by many researches to solve multi-constraint topology optimization problems is based on the method of moving asymptotes(MMA)
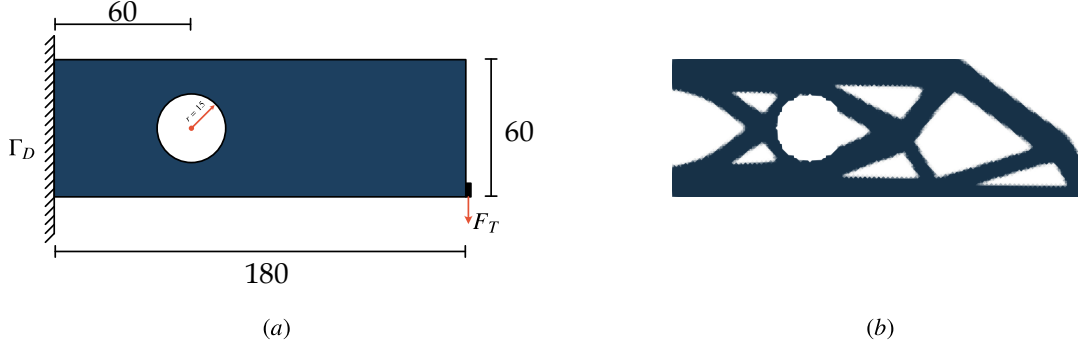
Figure 9: Cantilever beam with passive elements. (a)The ground structure with the applied external loads and boundary conditions for the optimization of a cantilever beam with a hole. (b) Topologically optimized structure.

algorithm[68]. The MMA-algorithm is a mathematical programming algorithm, which is in nature similar to methods like Sequential Linear Programming (SLP) and Sequential Quadratic Programming (SQP) for solving smooth, non-linear optimization problems [21]. The call to MMA requires definition of several input arguments along with the first and second derivatives of the objective as well as the constraint functions. In the case of TO the second order derivatives are assumed to be zero. One can find the definition of all the auxiliary variables in the documentation of code files which one can obtain from Prof. Krister Svanberg, KTH, Sweden. The MMA code is called by first defining the auxiliary variables as per the documentation and then replacing the optimality criteria block with the following:

```
xmma,ymma,zmma,lam,xsi,eta,aaa,zet,s,low,upp = \
        mmasub(m,n,loop,xval,xmin,xmax,xold1,xold2,f0val,df0dx,fval,dfdx,low,upp,a0,a,cMMa,d,move)
```

Since the code allows for direct access of the underlying matrices and vectors, one could also use the optimizers available in the open-source python libraries such a SciPy or PyOpt directly.

6. Extension to practical applications

This code is capable of running large scale simulations on systems ranging from high performance workstations to computational clusters. The readability of the code allows for easy modifications and the just in time compilation provides us with speed and efficiency. With a few modifications, the code can handle complex structural configuration with support for parallel computation. In this section we will discuss few challenges related to application of TO to large scale engineering problems with the help of a 3D bridge example and its implementation with the 55-line code. Fig. 10 shows the ground structure with the applied external loads and boundary conditions for the optimization of a 3D bridge with a non-designable deck. An opening of size $20m \times 25m$ is placed over the non-designable deck and a uniformly distributed load is
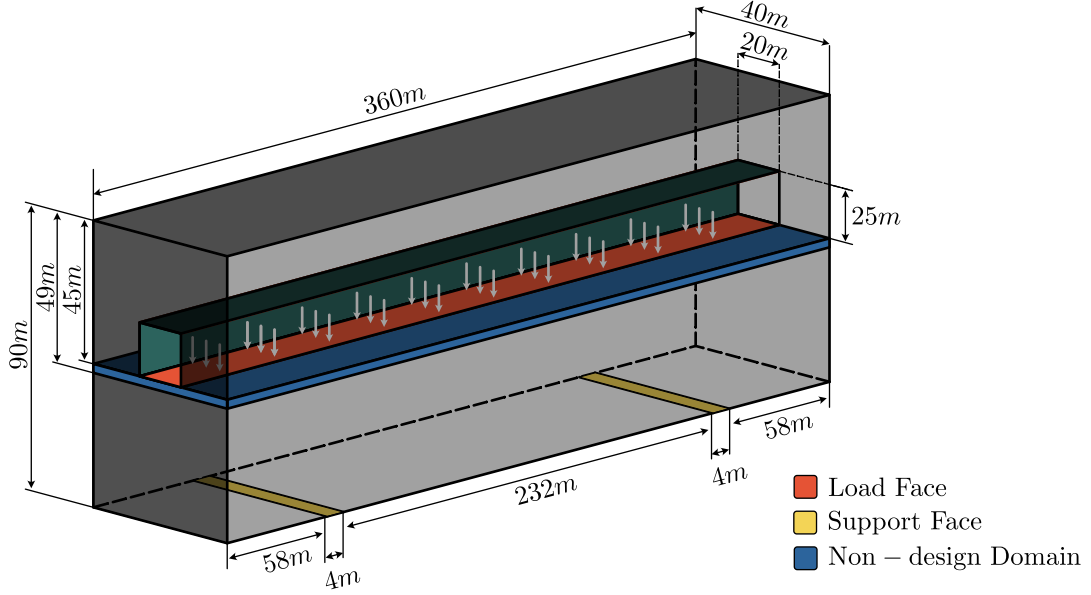
Figure 10: 3D bridge problem. The ground structure is 360m long with an opening for traffic. It further consist of a non-designable deck and 4m wide support faces. The structure is meshed with 6 million elements.

applied over the deck inside the opening. The structure is meshed with approximately six-million tetrahedral elements with three-million DoFs.

6.1. Complex structural configuration

The ground structure for the 3D-bridge as presented in Fig. 10 consists of three sub-domains: the non-designable deck, the load face and the support face. Such kind of structural configuration is difficult to handle in the MATLAB codes [63, 12, 29] but are comparatively easy to handle in commercial packages in which we can have an explicit representation of the complex structural configuration [77]. FEniCS supports marked computational mesh generated by 'gmsh'[31]; which allows us to explicitly mark different sub-domains in 'gmsh' with numerical labels and then read those directly into FEniCS. This extends the capability of the 55-line code to complex structural configurations. For complex geometries one needs to follow the mesh processing pipeline, and, thus create the mesh in 'gmsh' and mark different sub-domains, convert the mesh to XDMF with 'meshio'[59] and then finally read the mesh directly into the program with the commands:

```
mesh = Mesh()
with XDMFFile("domain.xdmf") as infile:
    infile.read(mesh)
```

The supports and loads are defined by reading the numerical labels of the mesh and defining 'Measures'(dx, ds) over the domain based on these labels. This is achieved by replacing lines $17 - 22$ with the following:

20

Figure 11: 3D bridge problem. Smoothened topologically optimized design.

```
mvc = MeshValueCollection("size_t", mesh, 2)
with XDMFFile("mesh/surface.xdmf") as infile:
        infile.read(mvc, "surface_tag")
mf = cpp.mesh.MeshFunctionSizet(mesh, mvc)
ds = Measure("ds")(subdomain_data=mf)
bcs = [DirichletBC(U, Constant((0.0, 0.0, 0.0)), mf, support_tag)]
F = dot(v, Constant((0.0, -1.0, 0.0))) * ds(load_tag)
```

The imported mesh will work directly with code as the computation of the filter is based on the actual centre-points of the elements (refer Section 4.5) as opposed to the assumptions made in the 88 or 99 line codes where the filter is evaluated based on the assumption of uniform mesh with unit cells. The optimized geometry presented in Fig. 11 is achieved by running the command:

```
from topopt import main
main(volfrac=0.15, penal=3.0, rmin=10.0)
```

6.2. Handling large 3D problems

Solving large scale 3D problems require access to systems with large amount of RAM and computational power. Even with powerful computational systems the implementation could be slow because of inefficient coding. The computational cost associated with TO can be attributed to three major operations: computation of the filter, evaluation of the displacement solution of the problem, and evaluation of the optimal solution of the optimization problem. Since all of these are dependent on the size of the mesh, the computational cost of the algorithm is directly related to the mesh.

In the original 99-line code and further it's evolution into the 88-line code, loops were used to evaluate the weight-matrices for the filter. Loops are known to be extremely slow as compared to vector operations,
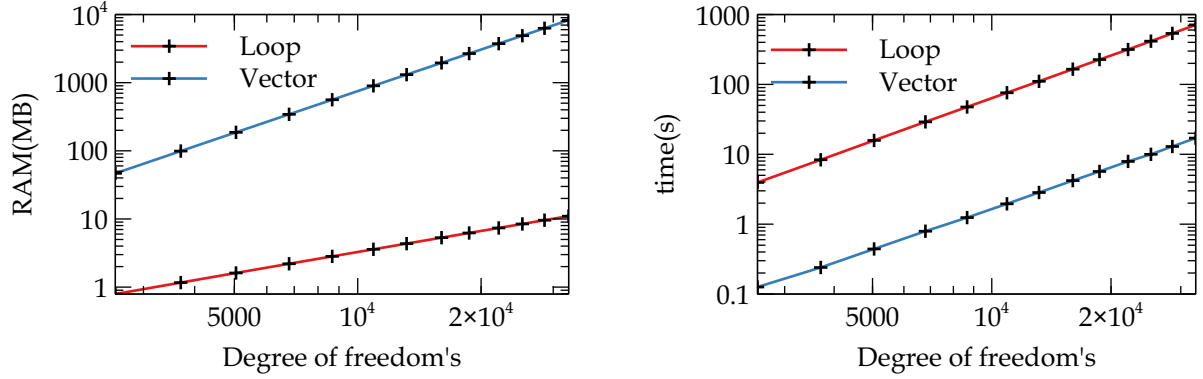
21

Figure 12: Comparison of looping versus vector operation for computation of filter. With increasing number of degrees of freedom's looping approach takes substantially less amount of RAM as compared to vector approach but require more time.

but, vector operations require a RAM size capable of storing the full matrices required to perform the computation. In Fig. 12, we can see that in terms of RAM, loops perform much better as compared to the vector approach, but, in terms of computational time the vector approach outperforms loop. For 33000 DoF's the RAM required by the looping approach is a mere 11MB as compared to 8GB required by vector approach, but, vector approach completes the computation in 17 seconds as opposed to 710 seconds taken by loops. To take benefit of both the approaches it is advised to use the chunking technique as described in Li et al. [42] by looping over large chunks of the mid-point vector which could be stored in the RAM for computation.

In the 88-line MATLAB code [12] substantial reduction in computational expense as compared to the original 99-line code [63] was achieved by vectorization of the loops and the new 99-line code [29] further provides a reduction in computational expense by smart use of sparse matrices and using the 'fsparse' function from the `stenglib` library that is written in 'C'. The use of sparse solvers along with vectorization has lead to substantial reduction in the computational expense of the evaluation of the displacement solution of the problem and also evaluation of the optimal solution of the optimization problem. The latest 99-line code achieves the assembly of the stiffness matrix of a six million DoF's system in under 10 seconds. Since we are using PetSc as the backend, our code benefits from the capabilities of the library such as support for parallelization, sparse matrices and iterative solvers. This results in under 10 second assembly of stiffness matrix for the same six million DoF's system, and with parallelization we can achieve further speedups.

6.3. Parallel implementation of topology optimization

Topology optimization algorithm requires computation of both the displacement solution of the systems and the optimization solution of the system in each iteration. To solve large scale engineering structures researchers in the past have parallelized the TO algorithm [2, 23, 25, 28, 61, 70, 71, 4, 50, 14, 48, 47]. Many of these works are extremely capable in solving the TO problem efficiently on parallel HPC systems, but are

too complex for beginners to understand and implement. In this section our aim is to present the parallel performance of our code and help beginners take benefit of multi-core simulations on there laptops or HPC workstations. This implementation by no means is as optimized as the 2589-line code by Schmidt and Schulz [61] or the 6300-lines code of Aage et al. [4], but serves as a proof-of-concept for the parallel capability of the 55-line code.

The modifications necessary to make the code compatible for parallel computation lies only in the modification of filter. Mesh-independency filter as described in Section 3.2.2 requires information of the co-ordinates of the center-points of elements in a zone around the element under consideration. Since, parallelization in FEniCS works by splitting the mesh into 'n' parts - where 'n' is the number of processors - and then passing only a portion of the mesh to a single processor, the information required for computation of the filter is not available with each processor. Thus, to make the code compatible for parallel computation we have to change the filter to the Helmholtz filter as described in [41, 27]. Parallel computation can then be performed on 16 cores by running the command:

```
mpirun -np 16 python3 main.py
```

In the above command `main.py` file contains the following code:

```python
from topopt import main
main(volfrac=0.15, penal=3.0, rmin=10.0)
```

It is to be noted that, our implementation supports complex structural configuration on parallel systems. For the problem presented in Fig. 10 with six million elements, the code took 10 minutes and 50 seconds per iteration on a single core, and, 1 minute and 40 seconds per iteration on 16 cores. This proves the code's scalability, and with further improvements in solver design and the use of better optimization algorithms, the code is capable of substantial computational gains.

7. Conclusion

This paper provides a simple, compact, and easy-to-understand implementation of topology optimization using the solid isotropic material with penalization methodology for 2D and 3D structures via an open-source finite element python package (FEniCS). By revisiting the original 99-line code by Sigmund [63] and its further modification to 88-lines by Andreassen et al. [12] and the new 99-line by Ferrari and Sigmund [29], we present a brief step-by-step overview of the topology optimization problem's mathematical model and a detailed description of the FEniCS implementation.

The 99-line code laid the foundation and has helped many researchers enter the field of topology optimization, emphasizing the 2D plane stress problem. The 88-line code significantly reduced the computational

time by vectorizing the loops, while also reducing the number of lines to 88. The new-99-line code further improved the efficiency of the original code by updating the code to latest developments in the sparse computation of matrices in MATLAB. This implementation's key strength is the generalized and compact approach enabled by FEniCS, which allows us to make a unified code-base for 2D and 3D problems with easy to understand UFL expressions that make the code extremely readable in just 55-lines.

Compactness in the code is also achieved by vectorizing most of the operations. By utilizing the Euclidean distance matrix concept, we have also vectorized the weight matrix computation for mesh-independency filters which has lead to substantial savings in the computational time for evaluation of the filter. This also allows us to evaluate the filter on complex structural configurations without any change in the code-base. We also demonstrate the capability of the code to handle complex large scale engineering structures with support for parallel computation on high performance workstations.

Since the expressions written in the python code are automatically converted into compiled C++ functions, the implementation is computationally efficient. The whole code-base is based on open-source packages, and thus, it is accessible and ready-to-use by the entire research community. Considering FEniCS is platform-independent [34], the code works on Windows, MAC and Linux systems alike. FEniCS is also equipped with powerful tools that allow straightforward extension of the current code into different areas, such as nonlinearity, hyper-elasticity, compliant mechanisms, and multi-scale modeling with the same base code.

8. Declarations

8.1. Funding

8.2. Conflict of interest

The authors declare that they have no known competing financial or personal relationship with other people or organizations that could inappropriately influence or bias the content of the paper.

Appendix A. FEniCS Code

```
1  from dolfin import *
2  import numpy as np, sklearn.metrics.pairwise as sp
3  # A 55 LINE TOPOLOGY OPTIMIZATION CODE --------------------------------
4  def main(nelx, nely, volfrac, penal, rmin):
5      sigma = lambda _u: 2.0 * mu * sym(grad(_u)) + lmbda * tr(sym(grad(_u))) * Identity(len(_u))
```

```
6      psi = lambda _u: lmbda / 2 * (tr(sym(grad(_u))) ** 2) + mu * tr(sym(grad(_u)) * sym(grad(_u)))
7      xdmf = XDMFFile("output/density.xdmf")
8      mu, lmbda = Constant(0.4), Constant(0.6)
9      # PREPARE FINITE ELEMENT ANALYSIS ----------------------------------
10     mesh = RectangleMesh(Point(0, 0), Point(nelx, nely), nelx, nely, "right/left")
11     U = VectorFunctionSpace(mesh, "P", 1)
12     D = FunctionSpace(mesh, "DG", 0)
13     u, v = TrialFunction(U), TestFunction(U)
14     u_sol, density_old, density = Function(U), Function(D), Function(D, name="density")
15     density.vector()[:] = volfrac
16     # DEFINE SUPPORT -------------------------------------------------
17     support = CompiledSubDomain("near(x[0], 0.0, tol) && on_boundary", tol=1e-14)
18     bcs = [DirichletBC(U, Constant((0.0, 0.0)), support)]
19     # DEFINE LOAD ----------------------------------------------------
20     load_marker = MeshFunction("size_t", mesh, mesh.topology().dim() - 1)
21     CompiledSubDomain("x[0]==l && x[1]<=2", l=nelx).mark(load_marker, 1)
22     ds = Measure("ds")(subdomain_data=load_marker)
23     F = dot(v, Constant((0.0, -1.0))) * ds(1)
24     # SET UP THE VARIATIONAL PROBLEM AND SOLVER ----------------------
25     K = inner(density ** penal * sigma(u), grad(v)) * dx
26     problem = LinearVariationalProblem(K, F, u_sol, bcs)
27     solver = LinearVariationalSolver(problem)
28     # PREPARE DISTANCE MATRICES FOR FILTER ---------------------------
29     midpoint = [cell.midpoint().array()[:] for cell in cells(mesh)]
30     distance_mat = rmin - sp.euclidean_distances(midpoint, midpoint)
31     distance_mat[distance_mat < 0] = 0
32     distance_sum = distance_mat.sum(1)
33     # START ITERATION ------------------------------------------------
34     loop, change = 0, 1
35     while change > 0.01 and loop < 2000:
36         loop = loop + 1
37         density_old.assign(density)
38         # FE-ANALYSIS ------------------------------------------------
39         solver.solve()
40         # OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS ----------------
41         objective = project(density ** penal * psi(u_sol), D).vector()[:]
42         sensitivity = -penal * (density.vector()[:]) ** (penal - 1) * project(psi(u_sol), D).vector()[:]
43         # FILTERING/MODIFICATION OF SENSITIVITIES --------------------
44         sensitivity = np.divide(distance_mat @ np.multiply(density.vector()[:], sensitivity), np.multiply(
                 density.vector()[:], distance_sum))
45         # DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD ------------
46         l1, l2, move = 0, 100000, 0.2
47         while l2 - l1 > 1e-4:
```

```
48          l_mid = 0.5 * (l2 + l1)
49          density_new = np.maximum(0.001,np.maximum(density.vector()[:] - move, np.minimum(1.0, np.minimum
                (density.vector()[:] + move, density.vector()[:] * np.sqrt(-sensitivity / l_mid)))))
50          l1, l2 = (l_mid, l2) if sum(density_new) - volfrac * mesh.num_cells() > 0 else (l1, l_mid)
51      density.vector()[:] = density_new
52      # PRINT RESULTS ------------------------------------------------
53      change = norm(density.vector()-density_old.vector(), norm_type="linf", mesh=mesh)
54      print("it.: {0} , obj.: {1:.3f} Vol.: {2:.3f}, ch.: {3:.3f}".format(loop, sum(objective), sum(
                density.vector()[:]) / mesh.num_cells(), change))
55      xdmf.write(density, loop)
```

References

[1] N. Aage and B. S. Lazarov. Parallel framework for topology optimization using the method of moving asymptotes. *Structural and multidisciplinary optimization*, 47(4):493–505, 2013. 2

[2] N. Aage and B. S. Lazarov. Parallel framework for topology optimization using the method of moving asymptotes. *Structural and Multidisciplinary Optimization*, 47(4):493–505, Apr. 2013. ISSN 1615-1488. doi: 10.1007/s00158-012-0869-2. 22

[3] N. Aage, E. Andreassen, and B. S. Lazarov. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572, Mar. 2015. doi: 10.1007/s00158-014-1157-0. 2

[4] N. Aage, E. Andreassen, and B. S. Lazarov. Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572, Mar. 2015. ISSN 1615-1488. doi: 10.1007/s00158-014-1157-0. 22, 23

[5] N. Aage, E. Andreassen, B. S. Lazarov, and O. Sigmund. Giga-voxel computational morphogenesis for structural design. *Nature*, 550(7674):84–86, 2017. 2

[6] S. Abhyankar, J. Brown, E. M. Constantinescu, D. Ghosh, B. F. Smith, and H. Zhang. Petsc/ts: A modern scalable ode/dae solver library. *arXiv preprint arXiv:1806.01437*, 2018. 8

[7] G. B. L. C. Ahrens, James. *ParaView: An End-User Tool for Large Data Visualization, Visualization Handbook*, Aug. 2005. 8

[8] M. S. Alnæs, A. Logg, K.-A. Mardal, O. Skavhaug, and H. P. Langtangen. Unified framework for finite element assembly. *International Journal of Computational Science and Engineering*, 4(4):231–244, 2009. doi: 10.1504/IJCSE.2009.029160. 8

[9] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software*, 40(2), 2014. doi: 10.1145/2566630. 8

[10] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015. doi: 10.11588/ans.2015.100.20553. 8

[11] O. Amir, N. Aage, and B. S. Lazarov. On multigrid-CG for efficient topology optimization. *Structural and Multidisciplinary Optimization*, 49(5):815–829, 2014. 2

[12] E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, Jan. 2011. doi: 10.1007/s00158-010-0594-7. 2, 15, 20, 22, 23

[13] M. Angeletti, J.-M. Bonny, and J. Koko. Parallel Euclidean distance matrix computation on big datasets *. Feb. 2019. 15

[14] J. Baiges, J. Martínez-Frutos, D. Herrero-Pérez, F. Otero, and A. Ferrer. Large-scale stochastic topology optimization using adaptive mesh refinement and coarsening through a two-level parallelization scheme. *Computer Methods in Applied Mechanics and Engineering*, 343:186–206, 2019. 22

[15] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997. 8

[16] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. Mcinnes, B. Smith, and H. Zhang. PETSc Users Manual. *ReVision*, (ANL-95/11 - Revision 3.12), 2010. doi: 10.2172/1178104. URL https://www.mcs.anl.gov/petsc. 8

[17] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Web page. https://www.mcs.anl.gov/petsc, 2019. URL https://www.mcs.anl.gov/petsc. 8

[18] M. P. Bendsøe. Optimal shape design as a material distribution problem. *Structural optimization*, 1(4):193–202, Dec. 1989. doi: 10.1007/BF01650949. 2

[19] M. P. Bendsøe. *Optimization of Structural Topology, Shape, and Material*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995. ISBN 978-3-662-03117-9 978-3-662-03115-5. doi: 10.1007/978-3-662-03115-5. 7

[20] M. P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71(2):197–224, Nov. 1988. doi: 10.1016/0045-7825(88)90086-2. 2

[21] M. P. Bendsoe and O. Sigmund. *Topology Optimization: Theory, Methods, and Applications*. Springer Science & Business Media, 2013. 2, 19

[22] J. Bleyer. Numerical tours of computational mechanics with FEniCS. 2018. doi: 10.5281/zenodo.1287832. 8

[23] T. Borrvall and J. Petersson. Large-scale topology optimization in 3D using parallel computing. *Computer Methods in Applied Mechanics and Engineering*, 190(46):6201–6229, Sept. 2001. ISSN 0045-7825. doi: 10.1016/S0045-7825(01)00216-X. 22

[24] V. J. Challis. A discrete level-set topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 41(3):453–464, Apr. 2010. doi: 10.1007/s00158-009-0430-0. 2

[25] V. J. Challis, A. P. Roberts, and J. F. Grotowski. High resolution topology optimization using graphics processing units (GPUs). *Structural and Multidisciplinary Optimization*, 49(2):315–325, Feb. 2014. ISSN 1615-1488. doi: 10.1007/s00158-013-0980-z. 22

[26] Q. Chen, X. Zhang, and B. Zhu. A 213-line topology optimization code for geometrically nonlinear structures. *Structural and Multidisciplinary Optimization*, 59(5):1863–1879, May 2019. doi: 10.1007/s00158-018-2138-5. 2

[27] E. M. de Souza and E. C. N. Silva. Topology optimization applied to the design of actuators driven by pressure loads. *Structural and Multidisciplinary Optimization*, 61(5):1763–1786, May 2020. ISSN 1615-147X, 1615-1488. doi: 10.1007/s00158-019-02421-5. 23

[28] A. Evgrafov, C. J. Rupp, K. Maute, and M. L. Dunn. Large-scale parallel topology optimization using a dual-primal substructuring solver. *Structural and Multidisciplinary Optimization*, 36(4):329–345, Oct. 2008. ISSN 1615-1488. doi: 10.1007/s00158-007-0190-7. 22

[29] F. Ferrari and O. Sigmund. A new generation 99 line Matlab code for compliance Topology Optimization and its extension to 3D. *arXiv:2005.05436 [cs, math]*, July 2020. 2, 20, 22, 23

[30] A. G. Allaire G (2009) A 2-d Scilab Code for shape and topology optimization by the level set method.

27

http://www.cmap.polytechnique.fr/∼allaire/levelset_en.html. 2009. 2

[31] C. Geuzaine and J.-F. Remacle. *Gmsh Reference Manual*. http://www.geuz.org/gmsh, 1.12 edition, Aug. 2003. 20

[32] H. Ghasemi, H. S. Park, and T. Rabczuk. A level-set based IGA formulation for topology optimization of flexoelectric materials. *Computer Methods in Applied Mechanics and Engineering*, 313:239–258, 2017. 2

[33] H. Ghasemi, H. S. Park, and T. Rabczuk. A multi-material level set-based topology optimization of flexoelectric composites. *Computer Methods in Applied Mechanics and Engineering*, 332:47–62, 2018. 2

[34] J. S. Hale, L. Li, C. N. Richardson, and G. N. Wells. Containers for Portable, Productive, and Performant Scientific Computing. *Computing in Science Engineering*, 19(6):40–50, Nov. 2017. ISSN 1558-366X. doi: 10.1109/MCSE.2017. 2421459. 24

[35] K. M. Hamdia, H. Ghasemi, Y. Bazi, H. AlHichri, N. Alajlan, and T. Rabczuk. A novel deep learning based method for the computational material design of flexoelectric nanostructures with topology optimization. *Finite Elements in Analysis and Design*, 165:21–30, 2019. 2

[36] D. Herrero, J. Martínez, and P. Martí. An implementation of level set based topology optimization using GPU. In *10th World Congress on Structural and Multidisciplinary Optimization, Orlando, Florida, USA*, pages 1–10, 2013. 2

[37] P. Kang and S.-K. Youn. Isogeometric topology optimization of shell structures using trimmed NURBS surfaces. *Finite Elements in Analysis and Design*, 120:18–40, Nov. 2016. doi: 10.1016/j.finel.2016.06.003. 2

[38] G. Kharmanda, N. Olhoff, A. Mohamed, and M. Lemaire. Reliability-based topology optimization. *Structural and Multidisciplinary Optimization*, 26(5):295–307, Mar. 2004. doi: 10.1007/s00158-003-0322-7. 2

[39] R. C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, 2004. doi: 10.1145/1039813.1039820. 8

[40] R. C. Kirby and A. Logg. A compiler for variational forms. *ACM Transactions on Mathematical Software*, 32(3), 2006. doi: 10.1145/1163641.1163644. 8

[41] B. S. Lazarov and O. Sigmund. Filters in topology optimization based on Helmholtz-type differential equations. *International Journal for Numerical Methods in Engineering*, 86(6):765–781, May 2011. ISSN 00295981. doi: 10.1002/nme.3072. 23

[42] Q. Li, V. Kecman, and R. Salman. A Chunking Method for Euclidean Distance Matrix Calculation on Large Dataset Using Multi-GPU. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 208–213, Washington, DC, USA, Dec. 2010. IEEE. ISBN 978-1-4244-9211-4. doi: 10.1109/ICMLA.2010.38. 15, 22

[43] K. Liu and A. Tovar. An efficient 3D topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 50(6):1175–1196, Dec. 2014. doi: 10.1007/s00158-014-1107-x. 2, 7

[44] A. Logg and G. N. Wells. Dolfin: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2), 2010. doi: 10.1145/1731022.1731030. 8

[45] A. Logg, K.-A. Mardal, G. N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012. ISBN 978-3-642-23098-1. doi: 10.1007/978-3-642-23099-8. 8

[46] L. Lu, T. Yamamoto, M. Otomori, T. Yamada, K. Izui, and S. Nishiwaki. Topology optimization of an acoustic metamaterial with negative bulk modulus using local resonance. *Finite Elements in Analysis and Design*, 72:1–12, Sept. 2013. doi: 10.1016/j.finel.2013.04.005. 2

[47] J. Martínez-Frutos and D. Herrero-Pérez. Large-scale robust topology optimization using multi-GPU systems. *Computer Methods in Applied Mechanics and Engineering*, 311:393–414, 2016. 22

[48] J. Martínez-Frutos and D. Herrero-Pérez. GPU acceleration for evolutionary topology optimization of continuum structures using isosurfaces. *Computers & Structures*, 182:119–136, 2017. 22

[49] J. Martínez-Frutos and D. Herrero-Pérez. Evolutionary topology optimization of continuum structures under uncertainty using sensitivity analysis and smooth boundary representation. *Computers & Structures*, 205:15–27, 2018. 2

[50] J. Martínez-Frutos, P. J. Martínez-Castejón, and D. Herrero-Pérez. Efficient topology optimization using GPU computing with multilevel granularity. *Advances in Engineering Software*, 106:47–62, Apr. 2017. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2017.01.009. 22

[51] H. P. Mlejnek. Some aspects of the genesis of structures. *Structural optimization*, 5(1):64–69, Mar. 1992. doi: 10.1007/BF01744697. 2

[52] S. S. Nanthakumar, T. Lahmer, X. Zhuang, H. S. Park, and T. Rabczuk. Topology optimization of piezoelectric nanostructures. *Journal of the Mechanics and Physics of Solids*, 94:316–335, 2016. 2

[53] S. S. Nanthakumar, X. Zhuang, H. S. Park, and T. Rabczuk. Topology optimization of flexoelectric structures. *Journal of the Mechanics and Physics of Solids*, 105:217–234, 2017. 2

[54] C. Nguyen, X. Zhuang, L. Chamoin, X. Zhao, H. Nguyen-Xuan, and T. Rabczuk. Three-dimensional topology optimization of auxetic metamaterial using isogeometric analysis and model order reduction. *Computer Methods in Applied Mechanics and Engineering*, 371:113306, 2020. 2

[55] K. B. Ølgaard and G. N. Wells. Optimisations for quadrature representations of finite element tensors through automated code generation. *ACM Transactions on Mathematical Software*, 37, 2010. doi: 10.1145/1644001.1644009. 8

[56] R. Ortigosa, J. Martínez-Frutos, A. J. Gil, and D. Herrero-Pérez. A new stabilisation approach for level-set based topology optimisation of hyperelastic materials. *Structural and Multidisciplinary Optimization*, 60(6):2343–2371, 2019. 2

[57] M. Otomori, T. Yamada, K. Izui, and S. Nishiwaki. MATLAB code for a level set-based topology optimization method using a reaction diffusion equation. *Struct Multidiscip Optim*, 51, 2014. doi: 10.1007/s00158-014-1190-z. 2

[58] D. H. Pérez. *Level Set Method Applied to Topology Optimization*. 2012. 2

[59] N. Schlömer, G. McBain, T. Li, K. Luu, christos, V. M. Ferrándiz, C. Barnes, eolianoe, L. Dalcin, nilswagner, A. Gupta, S. Müller, L. Schwarz, J. Blechta, C. Coutinho, D. Beurle, s1291, B. Shrimali, A. Cervone, T. Heister, T. Langlois, S. Peak, S. Sharma, M. Bussonnier, lgiraldi, G. Jacquenot, G. A. Vaillant, C. Wilson, A. U. Gudchenko, and A. Croucher. nschloe/meshio 3.3.1, Dec. 2019. 20

[60] S. Schmidt and V. Schulz. A 2589 line topology optimization code written for the graphics card. *Computing and Visualization in Science*, 14(6):249–256, Aug. 2011. doi: 10.1007/s00791-012-0180-1. 2

[61] S. Schmidt and V. Schulz. A 2589 line topology optimization code written for the graphics card. *Computing and Visualization in Science*, 14(6):249–256, Aug. 2011. ISSN 1433-0369. doi: 10.1007/s00791-012-0180-1. 22, 23

[62] O. Sigmund. On the design of compliant mechanisms using topology optimization. *Journal of Structural Mechanics*, 25(4):493–524, 1997. 2

[63] O. Sigmund. A 99 line topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, Apr. 2001. doi: 10.1007/s001580050176. 2, 7, 15, 20, 22, 23

[64] O. Sigmund. Morphology-based black and white filters for topology optimization. *Structural and Multidisciplinary Optimization*, 33(4-5):401–424, 2007. 2

[65] O. Sigmund and K. Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, 2013. 2

[66] O. Sigmund and J. Petersson. Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. *Structural optimization*, 16(1):68–75, 1998. 2

[67] K. Suresh. A 199-line Matlab code for Pareto-optimal tracing in topology optimization. *Structural and Multidisciplinary Optimization*, 42(5):665–679, Nov. 2010. doi: 10.1007/s00158-010-0534-6. 2

[68] K. Svanberg. The method of moving asymptotes—a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24(2):359–373, 1987. doi: 10.1002/nme.1620240207. 19

[69] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes. Polytop: A MATLAB implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Struct Multidiscip Optim*, 45, 2012. doi:

10.1007/s00158-011-0696-x. 2

[70] K. Vemaganti and W. E. Lawrence. Parallel methods for optimality criteria-based topology optimization. *Computer Methods in Applied Mechanics and Engineering*, 194(34):3637–3667, Sept. 2005. ISSN 0045-7825. doi: 10.1016/j.cma. 2004.08.008. 22

[71] E. Wadbro and M. Berggren. Megapixel Topology Optimization on a Graphics Processing Unit. *SIAM Review*, 51(4): 707–721, Nov. 2009. ISSN 0036-1445. doi: 10.1137/070699822. 22

[72] M. Y. Wang, X. Wang, and D. Guo. A level set method for structural topology optimization. *Comput Methods Appl Mech Eng*, 192, 2003. doi: 10.1016/S0045-7825(02)00559-5. 2

[73] P. Wei, Z. Li, X. Li, and M. Y. Wang. An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. *Structural and Multidisciplinary Optimization*, 58(2):831–849, Aug. 2018. doi: 10.1007/s00158-018-1904-8. 2

[74] W. Zhang, J. Yuan, J. Zhang, and X. Guo. A new topology optimization approach based on moving Morphable components (MMC) and the ersatz material model. *Struct Multidiscip Optim*, 53, 2016. doi: 10.1007/s00158-015-1372-3. 2

[75] M. Zhou and G. Rozvany. The COC algorithm, Part II: Topological, geometrical and generalized shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 89(1-3):309–336, Aug. 1991. doi: 10.1016/0045-7825(91) 90046-9. 2

[76] S. Zhou, J. Cadman, Y. Chen, W. Li, Y. M. Xie, and X. Huang. Design and fabrication of biphasic cellular materials with transport properties – a modified bidirectional evolutionary structural optimization procedure and MATLAB program. *Int J Heat Mass Transf*, 55, 2012. doi: 10.1016/j.ijheatmasstransfer.2012.08.028. 2

[77] Z. H. Zuo and Y. M. Xie. A simple and compact Python code for complex 3D topology optimization. *Advances in Engineering Software*, July 2015. doi: 10.1016/j.advengsoft.2015.02.006. 2, 20