

**DEVELOPMENT AND COMPARISON OF VARIOUS GRADIENT-BASED
OPTIMIZATION METHODS: APPLICATIONS IN STRUCTURAL AND
HEAT EXCHANGER PROBLEMS**

UNDERGRADUATE THESIS

Submitted as a partial fulfilment for the degree of
Sarjana Teknik from Institut Teknologi Bandung

By

Monica Indrawan

13616013

Advisor:

Dr. Lavi Rizki Zuhail

Dr. Pramudita Satria Palar



**UNDERGRADUATE PROGRAM IN AEROSPACE ENGINEERING
FACULTY OF MECHANICAL AND AEROSPACE ENGINEERING
INSTITUT TEKNOLOGI BANDUNG**

2020

DEVELOPMENT AND COMPARISON OF VARIOUS GRADIENT-BASED OPTIMIZATION METHODS: APPLICATIONS IN STRUCTURAL AND HEAT EXCHANGER PROBLEMS

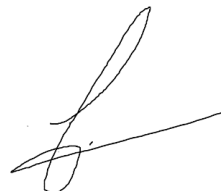
By
MONICA INDRAWAN
13616013

Undergraduate Program in Aerospace Engineering
Faculty of Mechanical and Aerospace Engineering
Bandung Institute of Technology

Approved by:
Advisor

Dr. Lavi Rizki Zuhal
19730116 200604 1 001

Co-Advisor

A handwritten signature in black ink, consisting of a large, stylized 'P' followed by a horizontal line and a small loop at the end.

Dr. Pramudita Satria Palar
19880622 201212 1 001

Abstract

DEVELOPMENT AND COMPARISON OF VARIOUS GRADIENT-BASED OPTIMIZATION METHODS: APPLICATIONS IN STRUCTURAL AND HEAT TRANSFER PROBLEMS

This research presents a benchmarking of several gradient-based optimization methods that are applied to structural and heat transfer topology optimization problems in 2D and 3D domain. The formulation of the problems is focused on the minimization of the compliance that is written in nested formulation. The Solid Isotropic Material Penalization (SIMP) material interpolation scheme is chosen to represent the topology model with the help of sensitivity filtering techniques. Minimizing the compliance leads to maximizing the global stiffness of the structure and minimizing the maximum temperature of the domain. The structural problems varied into cantilever problem, bridge problem and MBB problem whilst the heat transfer problem only considers the heat conduction case. Different optimization solvers including Optimality Criteria (OC), the Method of Moving Asymptotes (MMA), the interior point solvers in IPOPT and FMINCON and the sequential quadratic programming method in SNOPT, are benchmarked using sensitivity analysis. From this research, OC is presumed to be the most optimum method for solving the minimum compliance topology optimization problems.

Keywords: Topology optimization, SIMP approach, Optimality Criteria, Method of Moving Asymptotes, Interior Point Optimizer, Find Minimum of Constrained Nonlinear Multivariable Function, Sparse Nonlinear Optimizer

Abstrak

PENGEMBANGAN DAN PERBANDINGAN BERBAGAI METODE OPTIMISASI BERBASIS GRADIEN: APLIKASI PADA PERMASALAHAN STRUKTUR DAN PERPINDAHAN PANAS

Dalam penelitian ini, dilakukan perbandingan beberapa metode optimisasi berbasis gradien yang diaplikasikan pada permasalahan struktur dan perpindahan panas dalam domain dua dimensi dan tiga dimensi. Perumusan masalah difokuskan pada minimalisasi *compliance* yang diformulasikan dalam formulasi *nested*. Skema interpolasi dengan metode *Solid Isotropic Material Penalization (SIMP)* dipilih untuk merepresentasikan model topologi dengan bantuan teknik *sensitivity filtering*. Meminimumkan *compliance* berarti memaksimalkan kekakuan global dari struktur dan meminimumkan suhu maksimum dari domain. Permasalahan struktur yang dibahas terdiri atas permasalahan cantilever, permasalahan bridge, dan permasalahan MBB, sedangkan permasalahan perpindahan panas yang dibahas hanya permasalahan konduksi panas. Macam-macam metode optimisasi seperti *Optimality Criteria (OC)*, *Method of Moving Asymptotes (MMA)*, metode titik dalam di IPOPT dan FMINCON, serta *Sequential Quadratic Programming* di SNOPT, dibandingkan dengan menggunakan analisis sensitivitas. Berdasarkan penelitian ini, metode OC diduga sebagai metode yang paling optimal untuk permasalahan minimalisasi optimisasi topologi *compliance*.

Kata kunci: optimisasi topologi, pendekatan SIMP, *Optimality Criteria*, *Method of Moving Asymptotes*, *Interior Point Optimizer*, *Find Minimum of Constrained Nonlinear Multivariable Function*, *Sparse Nonlinear Optimizer*

Preface

This thesis is a prerequisite of getting the bachelors of engineering degree in aerospace and aeronautical engineering, faculty of mechanical and aerospace engineering, Bandung Institute of Technology. The scope of research in this undergraduate thesis is the benchmarking of gradient-based approach on structural and heat transfer topology optimization problems. This research was conducted from January 2020 to September 2020 under the guidance of the author's thesis advisor Dr. Lavi Rizki Zuhail and co-advisor Dr. Pramudita Satria Palar. The study in this thesis is the continuation on previous thesis by Nathan, Aerospace Engineering Student ITB Batch 2015 with the title of *Development of Level Set Approach for Gradient and Non-Gradient Based Structural Topology Optimization*.

Monica Indrawan

Acknowledgement

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my journey to complete the research successfully. I would like to express my sincere gratitude to my research supervisors, Dr. Lavi Rizki Zuhail, Vice Dean of Academic Affairs, Faculty of Mechanical and Aerospace Engineering, Bandung Institute of Technology and Dr. Pramudita Satria Palar for giving me the golden opportunity to conduct this research and providing invaluable guidance throughout this research. Their motivation, vision, enthusiasm and immense knowledge have deeply inspired me. They have spent their valuable time to guide me during this research despite of the outbreak of COVID pandemic. It was a great privilege and honour to work and study under their guidance. I would like to thank them for their patience, empathy and great sense of humour. Besides my supervisors, I would like to thank my thesis examiners: Dr. Ir. Bambang Kismono and Mr. Ony Arifianto Ph.D. for their encouragement, insightful comments and hard questions.

My completion of this research could not have been accomplished without the support of my fellow lab mates in Computational Aeroscience Group, Christopher Adnel for his invaluable support in every phase of this research. To Kak Nathan, Kak Kemas and Kak Ghifari, I am grateful for enlightening me the first glance of research. My sincere thanks also goes to my friend in Mathematics Major at Bandung Institute of Technology, Louis Owen for always motivating and helping me during my rough time in this research. I am overwhelmed with gratitude to acknowledge to all my friends whom I cannot mentioned each of them one by one, for helping me during my research.

Finally, to my caring, loving and supportive parents, I am deeply grateful for their love, prayers, and sacrifices for giving birth to me and preparing me for my future. Any attempt at any level cannot be satisfactorily completed without the support and guidance of my parents.

Jakarta, September 22, 2020

Monica Indrawan

Table of Contents

Abstract.....	i
Preface	iii
Acknowledgement	iv
List of Abbreviations	ix
CHAPTER I INTRODUCTION.....	1
1.1 Background.....	1
1.2 Problem Formulation	3
1.3 Objectives	3
1.4 Scope of Work	3
1.5 Outline.....	4
CHAPTER II FUNDAMENTAL THEORY	5
2.1. Topology Optimization Introduction	5
2.2. Solid Isotropic Material Penalization (SIMP) Method	7
2.2.1. Finite Element Method.....	8
2.3. Problem Formulation	14
2.4. Gradient-Based Optimization Methods.....	15
CHAPTER III NUMERICAL EXPERIMENT MODELLING	21
3.1. Numerical Experiment Workflow.....	21
3.2. Test Cases	27
3.3. Initialization	30
3.4. Algorithm Parameter.....	31
3.5. Stopping Criteria and Convergence Tolerance	32
CHAPTER IV RESULT AND ANALYSIS	34
4.1. Result and Analysis on 2D Optimization Problems.....	34
4.1.1 Numerical and Topology Result on 2D Optimization Problems	34
4.1.2 Benchmarking Gradient-Based Methods for 2D Problems.....	38
4.2. Result and Analysis on 3D Optimization Problems.....	40
4.2.1 Numerical and Topology Result on 3D Optimization Problems	40
4.2.2 Benchmarking on Gradient-Based Methods for 3D Problems.....	46
4.3. Topology Effect on Structural Stiffness.....	48
4.4. Topology Effect on Heat Conductivity	49
CHAPTER V CONCLUSION AND FUTURE WORKS	50
5.1. Conclusions.....	50

5.2. Future Works	52
REFERENCE	53
Appendix A.....	57
Appendix B.....	71
Appendix C.....	96
Appendix D.....	106

List of Figures

Figure 2.1 Illustration of Structural Topology Representation: SIMP (top left),.....	7
Figure 2.2 Four-node rectangular element	9
Figure 2.3 The eight-node hexahedron	11
Figure 3.1 OC Solver Workflow	22
Figure 3.2 FMINCON Solver Workflow	23
Figure 3.3 MMA Solver Workflow	24
Figure 3.4 IPOPT Solver Workflow	25
Figure 3.5 SNOPT Solver Workflow	26
Figure 3.6 Plot of compliance verses shape complexity for cantilever problem	27
Figure 3.7 2D Cantilever problem (top left), 2D Bridge Problem (top right),.....	29
Figure 3.8 3D Cantilever problem (top left), 3D bridge problem (top right),.....	30
Figure 4.1 Convergence plot of 2D cantilever problem.....	35
Figure 4.2 Convergence plot of 2D bridge problem	36
Figure 4.3 Convergence plot of 2D MBB problem.....	36
Figure 4.4 Convergence plot of 2D heat conduction problem	37
Figure 4.5 Convergence plot of 3D cantilever problem.....	43
Figure 4.6 Convergence plot of 3D bridge problem	44
Figure 4.7 Convergence plot of 3D cantilever medium-scale problem	44
Figure 4.8 Convergence plot of 3D heat conduction problem	45
Figure 4.9 The load percentage distribution on the Warren Truss Bridge.....	48
Figure 4.10 Irrelevant truss on 2D MBB problem topology result by SNOPT	49
Figure 4.11 Premature formation of supporting beam on 2D MBB problem topology result by IPOPT ...	49

List of Tables

Table 3.1 Parameters used in 2D optimization problem	29
Table 3.2 Parameters used in 3D optimization problem	30
Table 3.3 Parameter tuned in FMINCON	31
Table 3.4 Parameter tuned in IPOPT	31
Table 3.4 Parameter tuned in SNOPT	31
Table 3.6 Parameter, description and values of the tolerance criteria.....	33
Table 3.7. Parameter, description and values of termination criteria.....	33
Table 4.1 Topology layout of 2D structural optimization problem	34
Table 4.2 Topology layout of 2D heat transfer optimization problem	35
Table 4.3 Compliance, number of FE-call and CPU time of 2D cantilever problem	37
Table 4.4 Compliance, number of FE-call and CPU time of 2D bridge problem.....	37
Table 4.5 Compliance, number of FE-call and CPU time of 2D MBB problem.....	38
Table 4.6 Compliance, number of FE-call and CPU time of 2D heat conduction problem.....	38
Table 4.7 Topology layout of 3D structural optimization problem	42
Table 4.8 Topology layout of 3D heat transfer optimization problem	43
Table 4.9 Compliance, number of FE-call and CPU time of 3D cantilever problem	45
Table 4.10 Compliance, number of FE-call and CPU time of 3D bridge problem.....	45
Table 4.11 Compliance, number of FE-call and CPU time of 3D cantilever medium scale problem	46
Table 4.12 Compliance, number of FE-call and CPU time of 3D heat conduction problem.....	46

List of Abbreviations

AM	Additive Manufacturing
TO	Topology Optimization
RP	Rapid Prototyping
DDM	Direct Digital Manufacturing
SLS	Selective Laser Sintering
DMLS	Direct Metal Laser Sintering
OC	Optimality Criteria
MMA	Method of Moving Asymptotes
FMINCON	Find Minimum of Constrained Nonlinear Multivariable Function
IPOPT	Interior Point Optimizer
SNOPT	Sparse Nonlinear Optimizer
SIMP	Solid Isotropic Material with Penalization
MMC	Moving Morphable Components
RAMP	Rational Approximation Material Properties
LSF	Level Set Function
FEM	Finite Element Method
SAND	Simultaneous Analysis and Design
BFGS	Broyden-Fletcher-Goldfarb-Shanno
NLP	Non Linear Programming
NP	Non-deterministic Polynomial-time
SQP	Sequential Quadratic Programming
NPSOL	Non-linear Programming Stanford System Optimization Laboratory
QP	Quadratic Programming
FLP	Facility Location Problem

CHAPTER I

INTRODUCTION

1.1 Background

In this modern civilization, many industry uses Additive Manufacturing (AM) to build components layer upon layer using various materials. This method of manufacturing is capable of producing highly complex shape and design based on data from Computer Aided Design (CAD) software or 3D object scanners. Rather than removing the unused material that is involved in conventional manufacturing, it is more efficient to add it instead. Thus, AM is referred to as a “zero waste and efficient production”.

In addition to the efficiency, AM also brings digital flexibility as it provides a high degree of design freedom and thus escalates the level of optimization adjusted to objective design. The Additive Manufacturing process is very quick and has a wide range of application in aerospace, automotive, medical and many other field. AM enables the fabrication of parts and products that has been designed for weight reduction or better performances which makes it valuable in aerospace and automotive industries.

Additive Manufacturing itself has several subsets of technologies including 3D Printing, Rapid Prototyping (RP), Direct Digital Manufacturing (DDM), Selective Laser Sintering (SLS), layered manufacturing and additive fabrication. However, 3D Printing, RP or SLS are the three main terms used interchangeably with AM by some sources. Based on its baseline technology, AM technologies can be divided into three types. Starting with sintering in which the material is heated without being liquefied to create an object. SLS and DMLS is the examples of this technique. Another type of AM technologies is by melting the materials including direct laser metal sintering and electron beam melting. The last type of AM technologies is stereolithography (SLA), which is done by firing an ultraviolet laser into a tank of photopolymer resin.^[1]

Throughout the years, AM has received a tremendous interest and will most probably continue to prosper. AM has been proposed by number of companies and has been published in lots of scientific paper. Interest in AM also can be seen in a number of result for “Additive Manufacturing” term by website science direct.

Due to the free forms that naturally occurs, the topology result could be in a very novel and complex shape. In the past, it was often impractical to manufacture complex shape because of the limitations of traditional manufacturing methods. This has hindered the topology optimization efforts not to be fully realized. However, implementing topology optimization through AM provides the design engineers with complete freedom of design. Thus, the emergence of AM technologies could ease the procurable of highly complex shape from TO design. This could help further breakthrough in the topology optimization as a design technique to reduce the product weight if AM is considered.

In aerospace engineering, topology optimization can be used to determine the optimized wing box structure with an eye to be designed to sustain bending moment whilst dealing with aggressive weight targets.^[2] Thermal optimization problems have also mainly been discussed in several research papers during the last 21 years. TO in heat transfer problems could be used for designing combustor and turbine parts which are exposed to a very high temperature in aerospace turbine engine applications. By means of minimizing the maximum temperature, TO could improve the cooling efficiency for turbine parts hence it can increase the power output.^[3] Apart from aerospace field, this design approach is also applicable in mechanical, bio-chemical, and civil engineering.

There are two basic approaches in solving topology optimization: gradient-based method and non-gradient based method. Topology optimization in this undergraduate thesis will be focuses on gradient-based method due to relatively low amount of function evaluation required compared to non-gradient based method, the gradient based method will yield higher computational efficiency.^[4] Therefore, the writer will conduct a further study on several gradient based optimization methods in topology optimization and its applications in structural and thermal problems.

In topology optimization problems, there are some methods that has been acknowledged including Optimality Criteria (OC), Method of Moving Asymptotes (MMA), Globally Convergent MMA (GCMMA), the interior point method in Interior Point Optimizer (IPOPT) and Find Minimum of Constrained Nonlinear Multivariable Function (FMINCON), and the sequential quadratic programming method in Sparse Nonlinear Optimizer (SNOPT).^[5] The purpose of this research is to provide a detailed benchmarking result among various gradient-based topology

optimization method. One way to benchmark those methods can be done by evaluating the sensitivity analysis of each method in accordance to parameter alteration.

1.2 Problem Formulation

The problems are formulated as a guideline of this research in order to obtain a specific and direct study.

1. What is the best gradient-based optimization methods for 2D and 3D structural and heat transfer topology optimization test cases?
2. What is the optimum topology design for 2D and 3D structural and heat transfer topology optimization test cases?

1.3 Objectives

Build upon the research background, the objectives of this study are also formulated below.

1. Running and implementing the gradient-based optimization methods for 2D and 3D structural and heat transfer topology optimization test cases.
2. Evaluating and determining the best gradient-based optimization methods for 2D and 3D structural and heat transfer topology optimization test cases.
3. Finding and analysing the optimum topology design for 2D and 3D structural and heat transfer topology optimization test cases.

1.4 Scope of Work

This research is focused on benchmarking different topology optimization methods within the scope of gradient-based method on the application to 2D and 3D structural and heat transfer topology optimization problems specific to compliance minimization. There are some first-order methods used in the gradient-based methods such as Optimality Criteria (OC), Method of Asymptotes (MMA) and its globally convergent version GCMMA. However, dealing with nonlinear optimization problems, second-order solvers are also effective in solving them. The primal-dual interior point methods implemented in IPOPT and FMINCON and Sequential

Quadratic Programming (SQP) solver SNOPT are three implementations of second-order solvers.^[5]

All subroutine code programs were done in MATLAB R2018a software with the help of several optimization codes and libraries i.e. Optimality Criteria (OC) method (Bendsøe and Sigmund 2003; Andreassen et al. 2011), the Method of Moving Asymptotes (MMA) (Svanberg 1987), FMINCON (Mathworks 2013), IPOPT (Wächter and Biegler 2006) and SNOPT (Gill et al. 2005). IPOPT library is a part of COIN-OR project whilst SNOPT is a TOMLAB product, modelling platform in the field of solving optimization problems. Used assumptions taken into this research are stated below.

- Fictitious fluid element implemented on void element
- Negligible body force to overcome the design dependency effect
- Continuity assumption by using Finite Element Method (FEM)
- Elastic deformation assumption
- Negligible heat convection effect

1.5 Outline

This undergraduate thesis is organized as follows:

- Chapter 1 presents the research background, problems, objectives and scope as a guideline of this study.
- Chapter 2 reviews theoretical concepts of structural and heat governing equation, finite element method formulation (FEM), topology representation model, problem formulation chosen in this study and the general concept of the five topology optimization methods.
- Chapter 3 elaborates the numerical experiment workflow, test cases of topology optimization problems and parameters used in gradient-based topology optimization methods. Initialization and some convergence criteria are also discussed in this chapter.
- Chapter 4 reports the numerical experiment results and its benchmarking analysis between five topology optimization methods.
- Chapter 5 concludes the research and provides some future works for development.

CHAPTER II

FUNDAMENTAL THEORY

2.1. Topology Optimization Introduction

Structural optimization is an example of Nonlinear Programming hence it is often called as Mathematical Optimization. Mathematical Optimization is formally the process of finding the solution of an optimization problem. The optimization problem is commonly in the form of general mathematical formulation and has a constraint function (See Equation 2.1). The constraints itself could be differentiated as an equality and inequality constraints. If no constraints are stated, then the problem is defined as an unconstrained minimization problem.

$$\min_{w.r.t. x} f(x), x = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n$$

subject to the constraints:

$$\begin{aligned} g_j(x) &\leq 0, j = 1, 2, \dots, m \\ h_j(x) &= 0, j = 1, 2, \dots, r \end{aligned} \quad (2.1)$$

where x_i of $x = [x_1, x_2, \dots, x_n]^T$ are the design variables, $f(x)$ is the objective function, $g_j(x)$ is the inequality constraint functions and $h_j(x)$ is the equality constraint functions.^[6]

There are three types of structural optimization methods encompass size, shape and topology optimization. Size optimization focuses on determining material values, dimension, thickness or other component parameters with respect to performance goals. Whilst for shape optimization, it determines the optimized overall shape of component subjected to the given design constraints. In another hand, topology optimization has a different approach of achieving optimized design in which it can attain any shape and size within design space.^[7]

Aside from structural cases, topology optimization has also been applied to heat transfer problems since the so-called homogenization design method was first proposed by Bendsøe. The steady state equation is considered in the heat transfer problem with homogeneous boundary conditions. The design-dependent load effect is dealt by utilizing fictitious fluid elements in order to provide an accurate boundary conditions.^[8]

Topology optimization (TO) optimizes the distribution of material layout while satisfying a set of constraint to achieve the goal of maximizing the performance of the design structure. When

the solver has to perform an optimization, it removes material from the design space that is not needed until it finds the best topology. In consequence, the technology method is very useful at creating strong, lightweight part with less material.

TO can be solved either using gradient or non-gradient based techniques. Gradient based optimization is a method that uses gradient to achieve minimum value of the objective function either global or local minimum.^[6] Whilst non-gradient based uses stochastic search or random variables to obtain the maximum value of the objective function.^[7] The utilization of both techniques has its own advantages. The former method can solve fine-resolution problems with thousand to millions design variables using a few hundred function evaluations. On the other hand, the latter method can solve high nonlinear, multimodal and noisy problems yet it requires thousands of finite element function evaluations for low resolution problems. Due to higher computational efficiency, gradient-based method is chosen as the structural topology optimization approach in this study.

Several alternatives to represent the structural topology model such as Solid Isotropic Material with Penalization (SIMP) or “power-law approach”, Moving Morphable Components (MMC), Rational Approximation Material Properties (RAMP) and Level Set Function (LSF). In SIMP approach, material properties within each element used to discretize the design domain are assumed to be constant, and the parameters are the relative densities of the element. The structure grid is assigned with “0” if void and “1” if the material exists.^[9] In RAMP approach, the void phase is defined to be an incompressible hydrostatic fluid transferring pressure loads without further parameterization of the surface.^[10] Whilst in MMC, components such as voids are predefined as the design variable which fewer number of designed variables are needed compared to SIMP model.^[11] Finally, Level Set Function (LSF) is created with a value at each grid points with predefined threshold in which the level set value higher than the threshold will be the topology representation model.^[12] Considering the conceptual simplicity and high computational efficiency, SIMP model is chosen as the topology representation model in this study. Moreover, this approach is a robust model thus it can be used for any combination of design constraints and penalization can be adjusted freely.

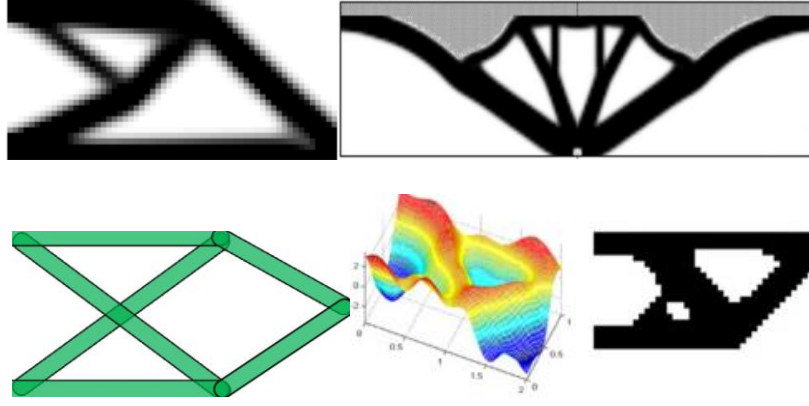


Figure 2.1 Illustration of Structural Topology Representation: SIMP (top left)^[9], RAMP (top right)^[10], MMC (bottom left)^[11], LSF (bottom right)^[12]

2.2. Solid Isotropic Material Penalization (SIMP) Method

The material distribution is formed to represent the topology layout in certain design domain with given boundary condition. Material interpolation scheme has to penalize intermediate densities values (solid or void design) into a nonlinear continuous problem thereupon a good design could be obtained. The Solid Isotropic Material Penalization (SIMP) material interpolation scheme or so-called “power-law approach” is the chosen approach to model the topology layout. Here, the power ($p \geq 3$) of density by certain constants multiplied with the material Young modulus. This power p diminishes intermediate densities in order to generate solid and void layout on the topology design.^[13]

$$E(\rho_e) = E_{min} + (E_0 - E_{min})\rho_e^p$$

$$0 \leq \rho_e \leq 1 \quad (2.2)$$

However, this approach must be combined with filtering techniques to avoid checker-board patterns in the final design. The checker-board patterns are well known as the areas where the material density jumps from 0 to 1 between neighbouring elements.^[14] The filter works by weighting the element sensitivities according to the difference between a chosen radius r as stated in Equation 2.3.

$$\frac{\widehat{\partial c}}{\partial x_e} = \frac{1}{\max(\gamma, x_e) \sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_i \frac{\partial c}{\partial x_i} \quad (2.3)$$

The weight factor H_{ei} is written as:

$$H_{ei} = \max(0, r_{min} - \Delta(e, i)) \quad (2.4)$$

where N_e is the set of element i , $\Delta(e, i)$ is the distance between centre of element e and i . The term γ ($= 10^{-3}$) is a small positive number added to prevent zero division.

2.2.1. Finite Element Method

The Finite Element Method (FEM) as the optimization basis is assign to evaluate the design performance of the problem. FEM discretized the geometry model into smaller domains. Each of this smaller domains are used to solve the partial differential governing equations to form the solution describing the whole system. This section presents the finite element formulation for partial differential equation.

Structural Finite Element Formulation

Several constitutive equations including displacement approximation, strain approximation, stress approximation and element stiffness matrix for each element are stated in Equation 2.5, Equation 2.6, Equation 2.7 and Equation 2.8, respectively.

Displacement approximation in terms of shape function for each element:

$$\{u\} = [N]\{d\} \quad (2.5)$$

Strain approximation in terms of strain-displacement matrix for each element:

$$\{\varepsilon\} = [B]\{d\} \quad (2.6)$$

Stress approximation for each element:

$$\{\sigma\} = [D][B]\{d\} \quad (2.7)$$

Hooke's Law

$$\begin{aligned} \{\sigma\} &= [D]\{\varepsilon\} \\ \begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} &= \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix} \end{aligned} \quad (2.8)$$

where E is the Young's modulus of the material, ν is the Poisson's ratio of the material, σ τ and ε are the normal stress and normal strain, respectively. Whilst, τ_{xy} and γ_{xy} are the shear stress and shear strain, respectively. Element stiffness matrix for each element:

$$[k] = \int_{V^e} [B][D][B] dV \quad (2.9)$$

Four Node Rectangular Element

4-noded rectangular element has four nodes per element with two degree of freedoms per node (x- and y- direction). Its nodes are located at the edges which parallel to the coordinate axes.^[15] Displacement d in x- and y- direction for each node is expressed in Equation 2.10.

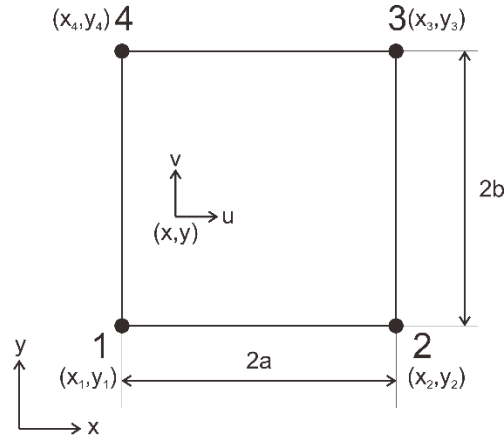


Figure 2.2 Four-node rectangular element

$$\{d\} = \{u_1 \ v_1 \ u_2 \ v_2 \ u_3 \ v_3 \ u_4 \ v_4\}^T \quad (2.10)$$

$$u(x, y) \approx \sum_{i=1}^4 N_i(x, y) u_i$$

$$v(x, y) \approx \sum_{i=1}^4 N_i(x, y) v_i \quad (2.11)$$

The shape function N_1, N_2, N_3 and N_4 are bilinear functions of x and y and zero along element boundaries not containing the nodal point.

$$N_1 = \frac{1}{4ab} (x - x_2)(y - y_4)$$

$$N_2 = -\frac{1}{4ab} (x - x_1)(y - y_3)$$

$$N_3 = \frac{1}{4ab} (x - x_4)(y - y_2)$$

$$N_4 = -\frac{1}{4ab} (x - x_3)(y - y_1) \quad (2.12)$$

By considering the stress-strain relation in Hooke's law (See Equation 2.8), \underline{B} matrix (strain displacement) corresponding to each element is obtained in Equation 2.13.

$$[B] = \begin{bmatrix} \frac{\partial N_1(x,y)}{\partial x} & 0 & \frac{\partial N_2(x,y)}{\partial x} & 0 & \frac{\partial N_3(x,y)}{\partial x} & 0 & \frac{\partial N_4(x,y)}{\partial x} & 0 \\ 0 & \frac{\partial N_1(x,y)}{\partial y} & 0 & \frac{\partial N_2(x,y)}{\partial y} & 0 & \frac{\partial N_3(x,y)}{\partial y} & 0 & \frac{\partial N_4(x,y)}{\partial y} \\ \frac{\partial N_1(x,y)}{\partial y} & \frac{\partial N_1(x,y)}{\partial x} & \frac{\partial N_2(x,y)}{\partial y} & \frac{\partial N_2(x,y)}{\partial x} & \frac{\partial N_3(x,y)}{\partial y} & \frac{\partial N_3(x,y)}{\partial x} & \frac{\partial N_4(x,y)}{\partial y} & \frac{\partial N_4(x,y)}{\partial x} \end{bmatrix}$$

$$[B] = \frac{1}{A} \begin{bmatrix} y-y_4 & 0 & y_3-y & 0 & y-y_2 & 0 & y-y_1 & 0 \\ 0 & x-x_2 & 0 & x_1-x & 0 & x-x_4 & 0 & x_3-x \\ x-x_2 & y-y_4 & x_1-x & y_3-y & x-x_4 & y-y_2 & x_3-x & y-y_1 \end{bmatrix} \quad (2.13)$$

Hence, the element stiffness matrix is stated in Equation 2.14.

$$[k] = \frac{Eh}{1-\nu^2} \begin{bmatrix} \frac{3-\nu}{6} & \frac{1+\nu}{8} & \frac{-3-\nu}{12} & \frac{-1+3\nu}{8} & \frac{-3+\nu}{12} & \frac{-1-\nu}{8} & \frac{\nu}{6} & \frac{1-3\nu}{8} \\ \frac{1+\nu}{8} & \frac{3-\nu}{8} & \frac{1-3\nu}{12} & \frac{\nu}{8} & \frac{-1-\nu}{12} & \frac{-3+\nu}{8} & \frac{-1+3\nu}{6} & \frac{-3-\nu}{8} \\ \frac{8}{-3+\nu} & \frac{6}{1-3\nu} & \frac{8}{3-\nu} & \frac{6}{-1-\nu} & \frac{8}{\nu} & \frac{12}{-1+3\nu} & \frac{8}{-3+\nu} & \frac{12}{1+\nu} \\ \frac{12}{-1+3\nu} & \frac{8}{\nu} & \frac{6}{-1-\nu} & \frac{8}{3-\nu} & \frac{6}{1-3\nu} & \frac{12}{-3-\nu} & \frac{12}{1+\nu} & \frac{8}{-3+\nu} \\ \frac{8}{-3+\nu} & \frac{6}{-1-\nu} & \frac{8}{\nu} & \frac{6}{1-3\nu} & \frac{8}{3-\nu} & \frac{12}{1+\nu} & \frac{8}{-3-\nu} & \frac{12}{-1+3\nu} \\ \frac{12}{-1-\nu} & \frac{8}{-3+\nu} & \frac{6}{-1+3\nu} & \frac{8}{-3-\nu} & \frac{6}{1+\nu} & \frac{8}{3-\nu} & \frac{12}{1-3\nu} & \frac{8}{\nu} \\ \frac{8}{\nu} & \frac{12}{-1+3\nu} & \frac{8}{-3+\nu} & \frac{12}{1+\nu} & \frac{8}{-3-\nu} & \frac{6}{1-3\nu} & \frac{8}{3-\nu} & \frac{6}{-1-\nu} \\ \frac{6}{1-3\nu} & \frac{8}{-3-\nu} & \frac{12}{1+\nu} & \frac{8}{-3+\nu} & \frac{12}{-1+3\nu} & \frac{8}{\nu} & \frac{6}{-1-\nu} & \frac{8}{3-\nu} \end{bmatrix} \quad (2.14)$$

Eight Node Hexahedral Element

During the 3D Finite Element simulation, it is very effective to use a hexahedron element considering the reduced number of element and the increase of element distortion resistance thereupon yields to the computational time will be reduced. Hexahedral element has a topology equivalent to a cube with eight corner nodes and three degrees of freedom per node. The element is evaluated as an isotropic element in natural coordinates namely ξ .^[16]

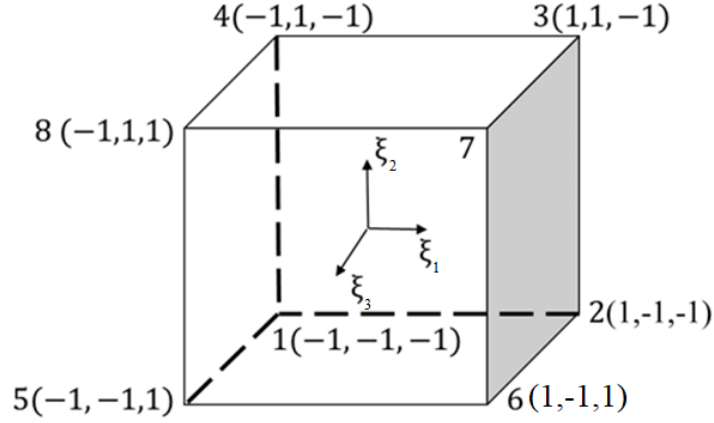


Figure 2.3 The eight-node hexahedron^[16]

The interpolation function of 3D constitutive matrix for an element is expressed in Equation 2.15.

$$C_i(\tilde{x}_i) = E_i(\tilde{x}_i)C_i^0$$

where

$$C_i^0 = \frac{1}{(1+\nu)(1-2\nu)} \times \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & (1-2\nu)/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1-2\nu)/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (1-2\nu)/2 \end{bmatrix} \quad (2.15)$$

The element stiffness matrix can be obtained by calculating the volume integral of the element constitutive matrix $C_i(\tilde{x}_i)$ and the strain-displacement matrix \underline{B} . The \underline{B} matrix relates the strain approximation as stated in Equation 2.6.

$$k_i(\tilde{x}_i) = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} B^T C_i(\tilde{x}_i) B d\xi_1 d\xi_2 d\xi_3 \quad (2.16)$$

Afterwards, the element stiffness matrix is also interpolated same as the constitutive matrix.

$$k_i(\tilde{x}_i) = E_i(\tilde{x}_i)k_i^0$$

where

$$k_i^0 = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} B^T C^0 B d\xi_1 d\xi_2 d\xi_3 \quad (2.17)$$

Hence, the element stiffness matrix k_i^0 for 8-node hexahedral element can be expressed as:

$$k_i^0 = \frac{1}{(1+\nu)(1-2\nu)} \begin{bmatrix} k_1 & k_2 & k_3 & k_4 \\ k_2^T & k_5 & k_6 & k_4^T \\ k_3^T & k_6 & k_5 & k_2^T \\ k_4 & k_3 & k_2 & k_1^T \end{bmatrix} \quad (2.18)$$

By accumulating the element stiffness matrix, the global version can be expressed as:

$$K(\tilde{x}) = \mathcal{A}_{i=1}^n k_i(\tilde{x}_i) = \mathcal{A}_{i=1E_i(\tilde{x}_i)}^n k_i^0 \quad (2.19)$$

Heat Conduction Finite Element Formulation

Consider T is the temperature and g is the heat generation, the heat governing equations and its derivation in Cartesian coordinate system can be written as:

$$\begin{aligned} \nabla^2 T &= g \\ \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} &= g(x, y) \end{aligned} \quad (2.20)$$

Deriving the equation in Cartesian coordinate system, the boundary integration of weighted residual yields:

$$\begin{aligned} I &= \int_{xy} N \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} - g(x, y) \right) dydx - \int_{\Gamma_e} N \frac{\partial T}{\partial n} d\Gamma \\ I &= - \int_{xy} \frac{\partial N}{\partial x} \frac{\partial T}{\partial x} dx dy + \oint_{\Gamma_1} N \frac{\partial T}{\partial x} n_x d\Gamma - \oint_{\Gamma_2} N \frac{\partial T}{\partial x} n_x d\Gamma \\ I_1 &= - \int_{xy} \frac{\partial N}{\partial x} \frac{\partial T}{\partial x} dx dy + \oint_{\Gamma} N \frac{\partial T}{\partial x} n_x d\Gamma \\ I_2 &= - \int_{xy} \frac{\partial N}{\partial y} \frac{\partial T}{\partial y} dx dy + \oint_{\Gamma} N \frac{\partial T}{\partial y} n_y d\Gamma \end{aligned} \quad (2.21)$$

where N is the shape function and Γ_e is the element boundary.

Finally, the equation 2.21 can be written as stated in Equation 2.22.

$$I = - \int_{xy} \left(\frac{\partial N}{\partial x} \frac{\partial T}{\partial x} + \frac{\partial N}{\partial y} \frac{\partial T}{\partial y} \right) dx dy - \int_{xy} N g(x, y) dx dy + \int_{\Gamma} N \frac{\partial T}{\partial n} d\Gamma \quad (2.22)$$

Four Node Rectangular Element

Applying the bilinear shape function in Equation 2.12, the element matrix for heat conductivity can be computed as:

$$[K^e] = \int_{xy} \left(\begin{Bmatrix} \frac{\partial N_1}{\partial x} \\ \frac{\partial N_2}{\partial x} \\ \frac{\partial N_3}{\partial x} \\ \frac{\partial N_4}{\partial x} \end{Bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \end{bmatrix} + \begin{Bmatrix} \frac{\partial N_1}{\partial y} \\ \frac{\partial N_2}{\partial y} \\ \frac{\partial N_3}{\partial y} \\ \frac{\partial N_4}{\partial y} \end{Bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \end{bmatrix} \right) dxdy \quad (2.23)$$

Therefore, the following element matrix for bilinear rectangular element can be generated by performing the integrations for all terms resulting the symmetrical matrix such that explained below.^[17]

$$[K^e] = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} \quad (2.24)$$

$$\begin{aligned} k_{22} &= k_{11} \\ k_{23} &= k_{14} = k_{41} \\ k_{24} &= k_{13} = k_{42} \\ k_{33} &= k_{11} \\ k_{34} &= k_{12} = k_{43} = k_{21} \\ k_{44} &= k_{11} \end{aligned} \quad (2.25)$$

Eight Node Hexahedral Element

Here a steady-state heat conduction equation in finite element formulation is written as:

$$K(k_i^0)U(k_i^0) = F \quad (2.26)$$

where $U(k_i^0)$ is the global temperature vector, F is the global thermal generation and $K(k_i^0)$ is the global thermal conductivity matrix. $K(k_i^0)$ is generated by the accumulation of element thermal conductivity matrices $k_i(\tilde{x}_i)$. The element conductivity matrix k_i can be formulated based on the following equation.^[16]

$$k_i(\tilde{x}_i) = [k_{min} + (k_0 - k_{min})\tilde{x}_i^p]k_i^0 \quad (2.27)$$

where k_0 represent the good thermal conduction whilst k_{min} represent the other poor conductor. Then, $K(k_i^0)$ can be expressed as:

$$K = \begin{bmatrix} 1/3 & 0 & -1/12 & 0 & 0 & -1/12 & -1/12 & -1/12 \\ 0 & 1/3 & 0 & -1/12 & -1/12 & 0 & -1/12 & -1/12 \\ -1/12 & 0 & 1/3 & 0 & -1/12 & -1/12 & 0 & -1/12 \\ 0 & -1/12 & 0 & 1/3 & -1/12 & -1/12 & -1/12 & 0 \\ 0 & -1/12 & -1/12 & -1/12 & 1/3 & 0 & -1/12 & 0 \\ -1/12 & 0 & -1/12 & -1/12 & 0 & 1/3 & 0 & -1/12 \\ -1/12 & -1/12 & 0 & -1/12 & -1/12 & 0 & 1/3 & 0 \\ -1/12 & -1/12 & -1/12 & 0 & 0 & -1/12 & 0 & 1/3 \end{bmatrix} \quad (2.28)$$

2.3. Problem Formulation

There are three types of topology optimization problems particularly minimum compliance, minimum volume, and compliant mechanism design problems. The chosen problem formulation is minimizing the compliance. Compliance minimization is the simplest objective to implement and it is known as the classical formulation of the problem. Compliance itself could be interpreted as flexibility under the prescribed support and loading condition or heat dissipation efficiency. Minimizing the compliance leads to maximizing the stiffness of the structure or minimizing the maximum temperature of the heat exchanger. The equilibrium equation to model the stiffness could be obtained by applying the finite element method.

$$K(x)u - F = 0 \quad (2.29)$$

where u is the state variable (nodal displacement), x is the design variable (density), F is the external load and $K(x)$ is the stiffness matrix or conductivity matrix.

Formulating the optimization problem could be attained using several ways. In Simultaneous Analysis and Design (SAND) approach, both displacement and design variable are considered as independent variables which are used to minimize the compliance. Computing gradient and Hessians of the objective and constraints function in SAND approach are easily computed. However, the number of the variables could be reduced such that only the design variable is used to formulate the problem in nested approach. The displacement caused by the force is then determined by Equation 2.30.

$$u(x) = K^{-1}(x)F \quad (2.30)$$

Therefore, nested formulations have reduced size advantage, at the cost of more complex sensitivity analysis. Equation 2.31 shows the minimum compliance problem written in nested formulation. The formulation has linear equality constraints with a non-linear objective function.

$$\begin{aligned}
& \underset{x}{\text{minimize:}} \quad c(x) = U^T K U = \sum_{e=1}^N (x_e)^p u_e^T k_0 u_e \\
& \text{Subject to:} \quad \frac{V(x)}{V_0} = v_{frac} \\
& \quad \quad \quad : KU = F \\
& \quad \quad \quad : 0 < x_{min} \leq x \leq 1
\end{aligned} \tag{2.31}$$

where U is the global displacement, F is the external force, K is the global stiffness matrix, u_e and k_e are the element displacement vector and stiffness matrix, respectively, x is vector of the design variables and N is the number of elements. ^[18]

Sensitivity analysis is then performed in each iteration through the optimization process in order to evaluate the impact of system's stiffness or conductivity caused by the variation of material densities. The sensitivity analysis is equivalent to the derivative of the objective function with respect to the design variable. Moreover, the second order derivative of the objective function is also computed in order to reduce the amount of computational memory although it can be approximated by the solver using limited-memory BFGS or other approaches. ^[19]

$$\frac{\partial c}{\partial x_e} = -p x_e^{p-1} (E_0 - E_{min}) u_e^T k_0 u_e \tag{2.32}$$

$$\frac{\partial^2 c}{\partial \tilde{x}_i \partial \tilde{x}_j} = \begin{cases} 0 & i \neq j, \\ 2[p \tilde{x}_i^{p-1} (E_0 - E_{min})]^2 & i = j. \\ [E_{min} + \tilde{x}_i^p (E_0 - E_{min})]^{-1} u_i^T k_i^0 u_i, & i = j. \end{cases} \tag{2.33}$$

2.4. Gradient-Based Optimization Methods

Topology optimization problems could be solved using several different gradient-based methods. These methods use the gradient information of the objective function in order to find the optimum solution. One of the advantages underlying the implementation of gradient based methods is the high computational efficiency as a result of their rapid convergence. However, these methods are not effective in finding global optima due to the presence of multiple local optima. Therefore, the final solution tends to be reliant to the starting points. ^[20]

Optimality Criteria (OC)

In Optimality Criteria (OC) methods, the optimization problem is solved by a heuristic updating scheme.

$$x_e^{new} = \begin{cases} \max(0, x_e - m) & \text{if } x_e B_e^\eta \leq \max(0, x_e - m) \\ \min(1, x_e + m) & \text{if } x_e B_e^\eta \geq \min(1, x_e + m) \\ x_e B_e^\eta & \text{otherwise} \end{cases} \quad (2.34)$$

where *move* ($m = 0.2$) is a positive move limit, $\eta (= 1/2)$ is a numerical damping coefficient and B_e is found from the optimality condition.

$$B_e = \frac{-\frac{\partial c}{\partial x_e}}{\lambda \frac{\partial V}{\partial x_e}} \quad (2.35)$$

where λ is a Lagrangian multiplier that could be found using bisection algorithm.^[21]

$$\|x^{new} - x\|_\infty \leq \epsilon \quad (2.36)$$

Method of Moving Asymptotes (MMA)

The other approach of method discussed in this research is Method of Moving Asymptotes (MMA) which was introduced by Prof. Svanberg in 1987. MMA works by approximating the constraint functions $f_i(x)$ at certain iteration point into certain convex functions $\tilde{f}_i^{(k)}(x)$.

$$\tilde{f}_i^{(k)}(\mathbf{x}) = \sum_{j=1}^n \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j} + \frac{q_{ij}^{(k)}}{x_j - l_j^{(k)}} \right) + r_i^{(k)}, i = 0, 1, \dots, m$$

where

$$\begin{aligned} p_{ij}^{(k)} &= (u_j^{(k)} - x_j^{(k)})^2 \left(1.001 \left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^+ + 0.001 \left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^- + \frac{10^{-5}}{x_j^{max} - x_j^{min}} \right) \\ q_{ij}^{(k)} &= (x_j^{(k)} - l_j^{(k)})^2 \left(0.001 \left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^+ + 1.001 \left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^- + \frac{10^{-5}}{x_j^{max} - x_j^{min}} \right) \\ r_i^{(k)} &= f_i(\mathbf{x}^{(k)}) - \sum_{j=1}^n \left(\frac{p_{ij}^{(k)}}{u_j^{(k)} - x_j^{(k)}} + \frac{q_{ij}^{(k)}}{x_j^{(k)} - l_j^{(k)}} \right) \end{aligned}$$

Here, $\left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^+$ denotes the largest of the two numbers $\frac{\partial f_i}{\partial x_j}(x^{(k)})$ and 0,

while $\left(\frac{\partial f_i}{\partial x_j}(x^{(k)}) \right)^-$ denotes the largest of the two numbers $-\frac{\partial f_i}{\partial x_j}(x^{(k)})$ and 0. (2.37)

The selection of corresponding functions is based primarily on gradient information at the current iteration point, but also on some parameters $u_j^{(k)}$ and $l_j^{(k)}$ (moving asymptotes). The moving asymptotes are modified using the information from previous iteration points. These approximating functions are so-called sub-problem that is generated at each iteration point.

For $k = 1$ and $k = 2$,

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - 0.5(x_j^{max} - x_j^{min}), \\ u_j^{(k)} &= x_j^{(k)} + 0.5(x_j^{max} - x_j^{min}). \end{aligned} \quad (2.38)$$

For $k \geq 3$,

$$\begin{aligned} l_j^{(k)} &= x_j^{(k)} - \gamma_j^{(k)} (x_j^{(k-1)} - l_j^{(k-1)}), \\ u_j^{(k)} &= x_j^{(k)} + \gamma_j^{(k)} (u_j^{(k-1)} - x_j^{(k-1)}) \end{aligned} \quad (2.39)$$

where

$$\gamma_j^{(k)} = \begin{cases} 0.7 & \text{if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) < 0, \\ 1.2 & \text{if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) > 0, \\ 1 & \text{if } (x_j^{(k)} - x_j^{(k-1)})(x_j^{(k-1)} - x_j^{(k-2)}) = 0, \end{cases} \quad (2.40)$$

Two approaches for solving the sub-problem in MMA i.e dual approach and primal-dual interior-point approach. Newton's method is used to solve set of KKT conditions.

$$\begin{aligned} \text{minimize} \quad & g_0(x) + a_0 z + \sum_{i=1}^m \left(c_i y_i + \frac{1}{2} d_i y_i^2 \right) \\ \text{subject to} \quad & g_i(x) - a_i z - y_i \leq b_i, & i = 1, \dots, m \\ & \alpha_j \leq x_j \leq \beta_j, & j = 1, \dots, n \\ & z \geq 0, y_i \geq 0, & i = 1, \dots, m \end{aligned} \quad (2.41)$$

where

$$g_i(x) = \sum_{j=1}^n \left(\frac{p_{ij}}{u_j - x_j} \frac{q_{ij}}{x_j - l_j} \right), \quad i = 0, 1, \dots, m \quad (2.42)$$

where g_0 is the objective function, g_i is the constraint function, x_j is the design variables where x_j^{min} and x_j^{max} are the lower and upper bounds of x_j and it should satisfy $l_j < \alpha_j < \beta_j < u_j$ for all j .^[22]

Find Minimum of Constrained Nonlinear Multivariable Function (FMINCON)

FMINCON covers set of nonlinear optimization algorithms such as trust region methods, active set algorithms, SQP algorithms, interior point algorithms and others. The interior point approach used in FMINCON to solve minimization problem is approximated as follows:

$$\begin{aligned} \min_{x,s} f_\mu(x, s) &= \min_{x,s} f_\mu(x) - \mu \sum_i \ln(s_i), \\ \text{subject to } h(x) &= 0 \text{ and } g(x) + s = 0 \end{aligned} \quad (2.43)$$

where $h(x)$ and $g(x)$ is the equality and inequality constraint function, respectively. The number of slack s_i is the same as the number of inequality constraints $g(x)$ and it should be kept a positive number. The minimum of f_μ would converge the minimum of f during the reduction of μ to zero value. The algorithm uses conjugate gradient (CG) approach to determine the step along the optimization proses. In this approach, the quadratic approximation in thrust region is minimized by subjecting to the linearized constraint

The algorithm decreases a merit function at each iteration as follows.

$$f_\mu(x, s) + \nu \|h(x), g(x) + s\| \quad (2.44)$$

As well as iteration number increases, the parameter ν will also increase in order to obtain the solution at feasibility convergence. If the *merit* function is not reduced at certain step, the algorithm will attempt a new step.^[23]

Interior Point Optimizer (IPOPT)

IPOPT is a software library that uses line searches based on filter methods in order to find a local solution of NLP. IPOPT uses a limited-memory quasi-Newton with BFGS update formula to ensure memory availability in the Hessian approximation.^[19] Similar with FMINCON, the problem formulation is simplified using the primal-dual barrier approach as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \varphi_\mu(x) &:= f(x) - \mu \sum_{i=1}^n \ln(x^{(i)}), \\ \text{subject to } c(x) &= 0 \end{aligned} \quad (2.45)$$

for a decreasing sequence of barrier parameters μ converging to zero. Homotopy method is applied to the primal-dual equations as follows:

$$\begin{aligned}
\nabla f(x) + \nabla c(x)\lambda - z &= 0 \\
c(x) &= 0 \\
XZe - \mu e &= 0
\end{aligned} \tag{2.46}$$

With the homotopy parameter μ which is driven to zero and λ is the lagrangian multipliers for the equality constraints and bound constraints and Equation 2.46 with $\mu = 0$ and " $x, z \geq 0$ " are the Karush-Khun-Tucker (KKT) conditions.

Consider k is the iteration counter, an iterate (x_k, λ_k, z_k) with $x_k, z_k > 0$ and $(d_k^x, d_k^\lambda, d_k^z)$ to denote the search directions, the following symmetric linear system equivalent to the the primal-dual equation (2.46) is obtained as follows:

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & A_k \\ A_k^T & -\delta_c I \end{bmatrix} \begin{pmatrix} d_k^x \\ d_k^\lambda \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_{\mu j}(x_k) + A_k \lambda_k \\ c(x_k) \end{pmatrix} \tag{2.47}$$

In this implementation, the linear system is solved by using a damped Newton's method.^[24]

Sparse Nonlinear Optimizer (SNOPT)

Consider the following NP problem. Let $f(x)$ is a linear or nonlinear objective function, $c(x)$ is a vector of nonlinear functions $c_i(x)$ with sparse derivatives, A is a sparse matrix, and l and u are vectors of lower and upper bounds. All the nonlinear functions are considered smooth and have their first derivatives.

$$\begin{aligned}
&\min_{x \in \mathbb{R}^n} f(x) \\
&\text{subject to } l \leq \begin{pmatrix} x \\ c(x) \\ Ax \end{pmatrix} \leq u
\end{aligned} \tag{2.48}$$

SNOPT (sparse nonlinear optimizer) is a SQP method which applied the theoretical properties of the NPSOL algorithm for wider large problems by utilizing sparsity in the constraint Jacobian and also uses limited-memory quasi-Newton to approximate Hessian matrix H_k . This method updates H_k in the presence of negative curvature. An inertia-controlling reduced-Hessian active-set method is used to solve the QP sub-problems, which allows the variables to be linear in the objective and constraint function.

To solve the problem subject to infeasible constraints, SNOPT exploits the use of l_1 penalty function. Here, infeasible linear constraints are detected by solving the below FLP problem. Let e be a vector of ones and v and w are handled implicitly.

$$\begin{aligned} \min_{x,v,w} e^T(v + w) \\ \text{subject to } l \leq \begin{pmatrix} x \\ Ax - v + w \end{pmatrix} \leq u, v \geq 0, w \geq 0 \end{aligned} \quad (2.49)$$

The expression above is in the form of a general linear constraint optimization problem with 3 variables x , v , and w . The three variables is also simply bounded by zero, u , and l . All subsequent iterates satisfy the linear constraints if $v = 0$ and $w = 0$. Otherwise, SNOPT terminates which causes the nonlinear functions not being computed.

SNOPT then solves the given (NP) by exploiting QP subproblems based on linearizations of the nonlinear constraints. If a QP subproblem is found infeasible, SNOPT will solve the following $NP(\gamma)$ problem. Let $\gamma \geq 0$ be a penalty parameter which may take a finite sequence of increasing values.

$$\begin{aligned} \min_{x,v,w} f(x) + e^T(v + w) \\ \text{subject to } l \leq \begin{pmatrix} x \\ c(x) - v + w \\ Ax \end{pmatrix} \leq u, v \geq 0, w \geq 0, \end{aligned} \quad (2.50)$$

where $f(x) + \gamma e^T(v + w)$ is called a *composite objective*. If γ is large enough and (NP) has a feasible solution, then (NP) and $(NP(\gamma))$ outputs an identical solution. However, if (NP) has no feasible solution and given that γ is sufficiently large, $(NP(\gamma))$ will output a good infeasible point.^[25]

CHAPTER III

NUMERICAL EXPERIMENT MODELLING

3.1. Numerical Experiment Workflow

Figure 3.1, Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5 presents the numerical experiment workflow by gradient-based methods i.e OC, FMINCON, MMA, IPOPT and SNOPT, respectively. The workflow starts with the input of geometry model dimension, material properties, and other configuration parameters. The equality constraint of the volume fraction and the starting point of the final solution is defined. After the boundary and loading conditions are specified, the code programs prepare the finite element by generating element stiffness matrix. Several configuration regarding the solver should be determined based on how each system perform to find the optimum solution. Then, the solvers will try to solve the optimization problem iteratively until it reaches prescribed convergence criteria. The design variables are computed by calculating finite element analysis, objective function and sensitivity analysis with the help of sensitivity filtering methods. Furthermore, the use of Jacobian and Hessian functions is also necessary to second-order solvers. Finally, several data such as compliance value, CPU time required and topology layout could be extracted after the solvers terminate.

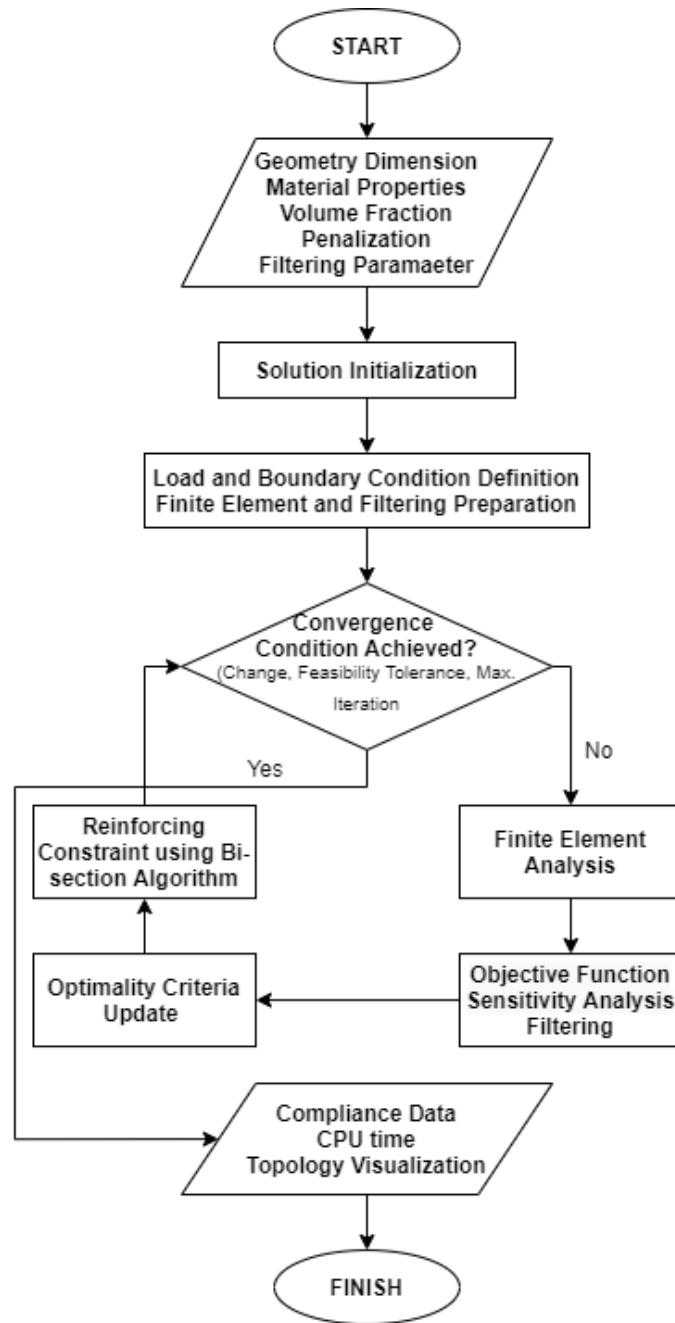


Figure 3.1 OC Solver Workflow

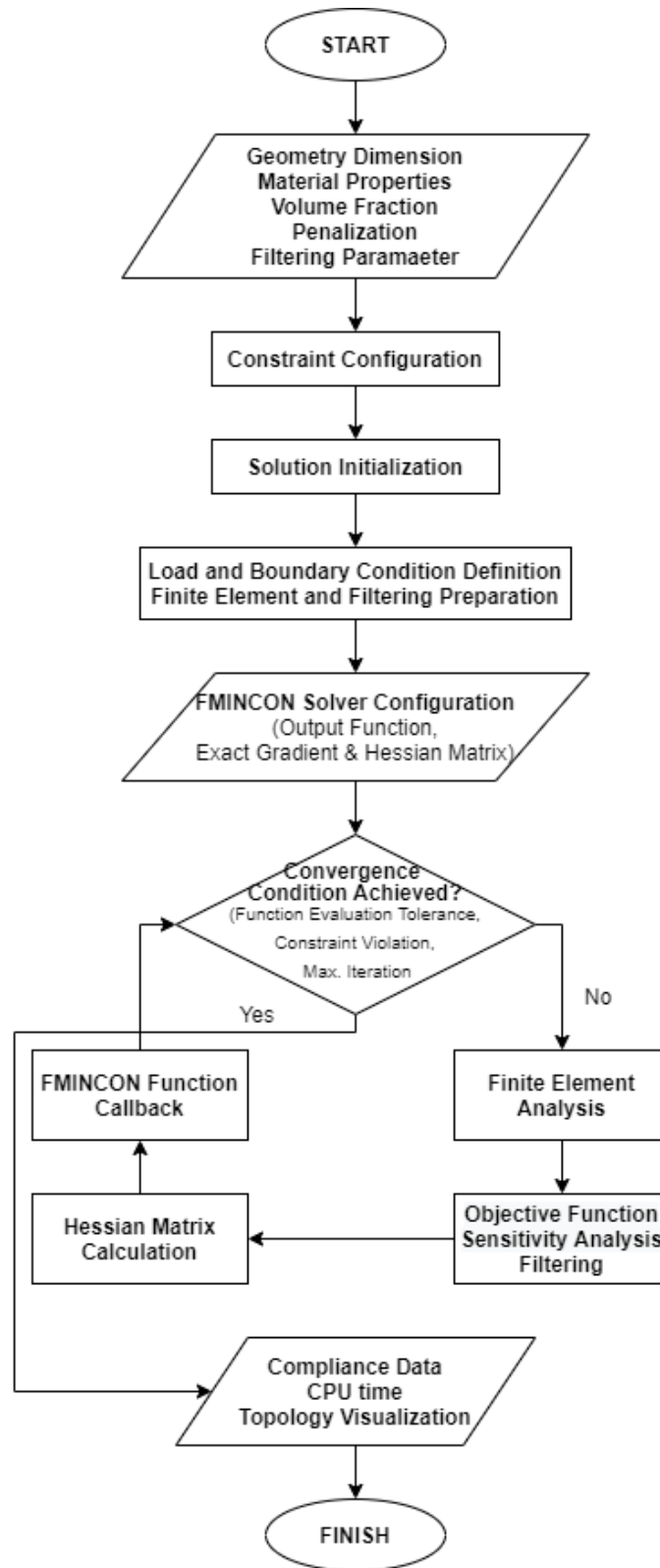


Figure 3.2 FMINCON Solver Workflow

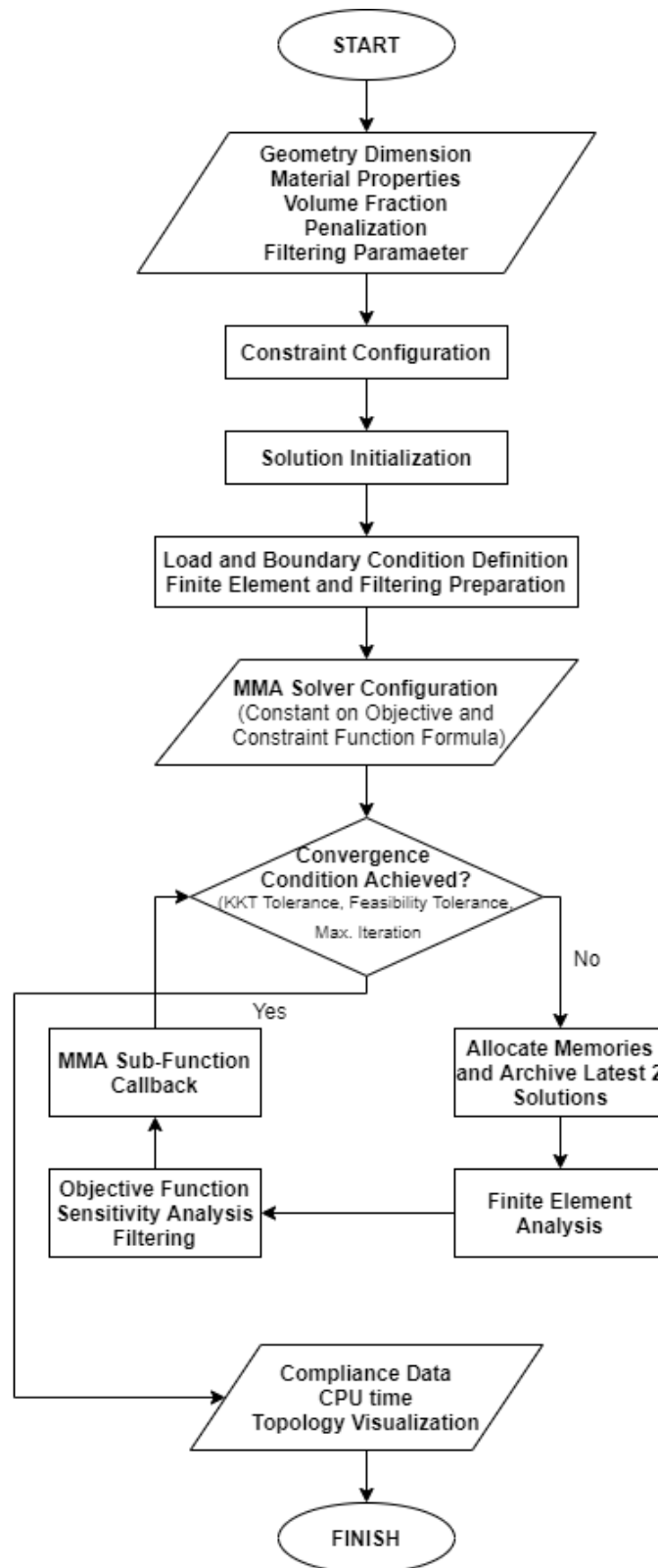


Figure 3.3 MMA Solver Workflow

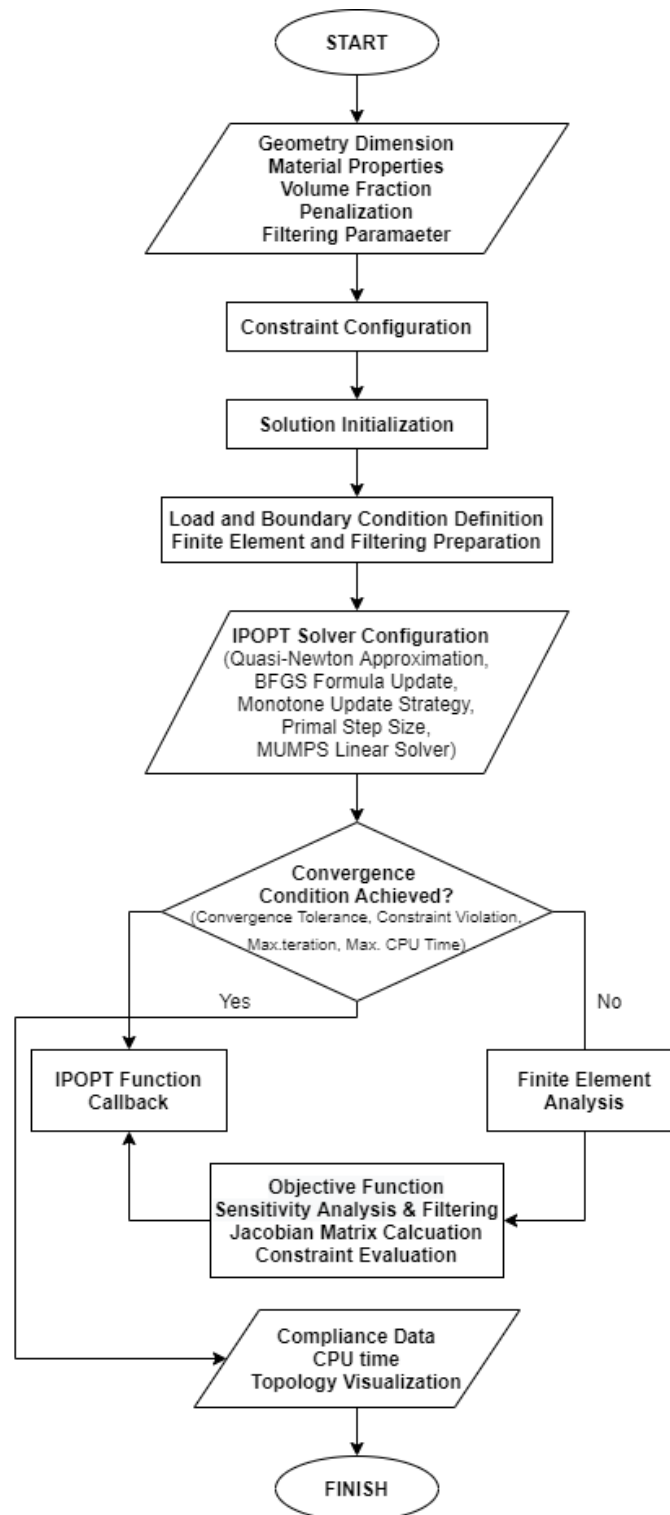


Figure 3.4 IPOPT Solver Workflow

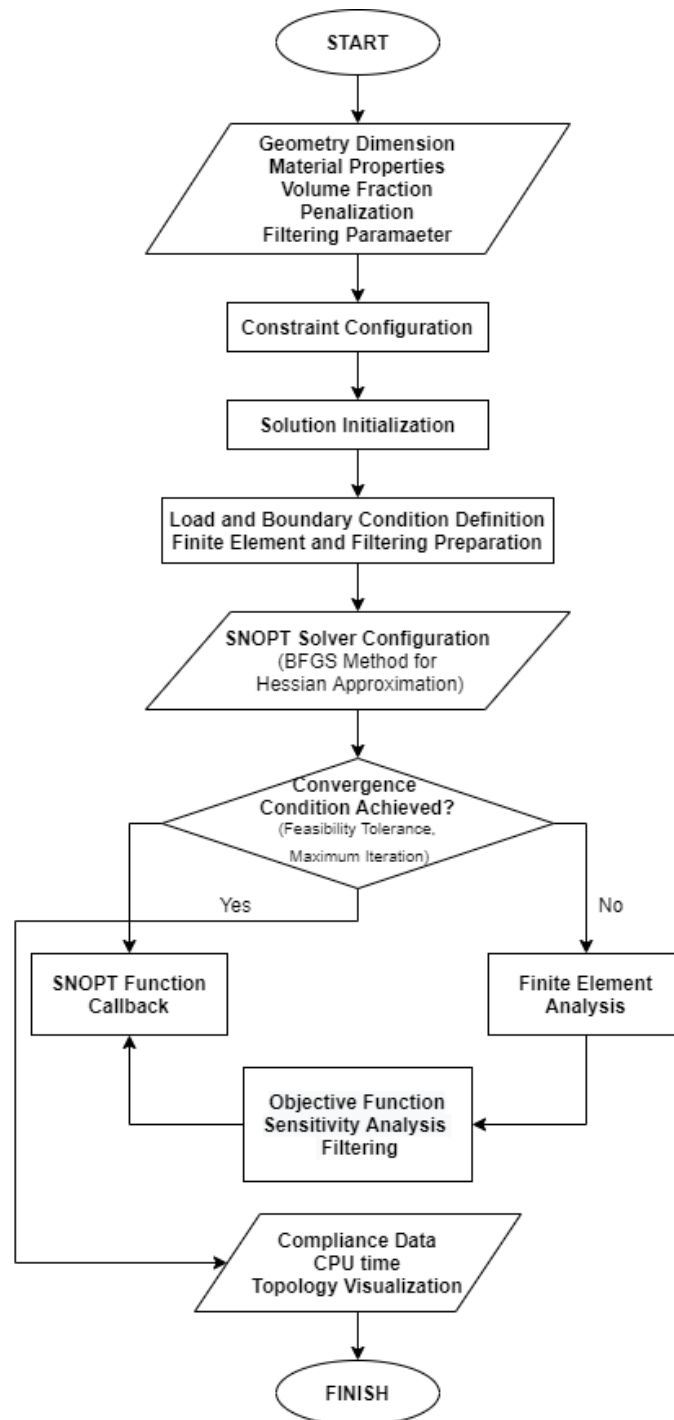


Figure 3.5 SNOPT Solver Workflow

3.2. Test Cases

The test set of topology optimization problems must be able to represent the characteristics of the methods' performances. Large-scale problems are more capable to provide more accurate designs so that the results are conclusive and correctly identify each method advantages and disadvantageous. However, small and medium-sized problems with much number of problem sets are also appropriate to be used in this benchmarking. This selection of test set problems is done by considering memory problem resulted by larger numbers of elements.

The test sets are classified into two domains: two dimensional and three dimensional design domains. The structure is discretized with 4-node rectangular element in 2D domain and 8-node hexahedral element in 3D domain. In each domain, the test set has four examples with different boundary conditions and location of the external loads with neglected structural weight (See Figure 3.7 and Figure 3.8). Large difference lengths ratio is considered in order to deepen the investigation of methods' capabilities in structural problems. Moreover, small volume bounds sometimes yields to difficulty in solving optimization problems. The material properties used in this study are Young's elasticity modulus $E_0 = 1$ for solid material, $E_{min} = 10^{-9}$ for void material, Poisson's ratio $\nu = 0.3$, limits of the material's thermal conductivity coefficient $k_0 = 1$ and $k_{min} = 10^{-3}$. All the static external load is equivalent to a vector with a value of one.

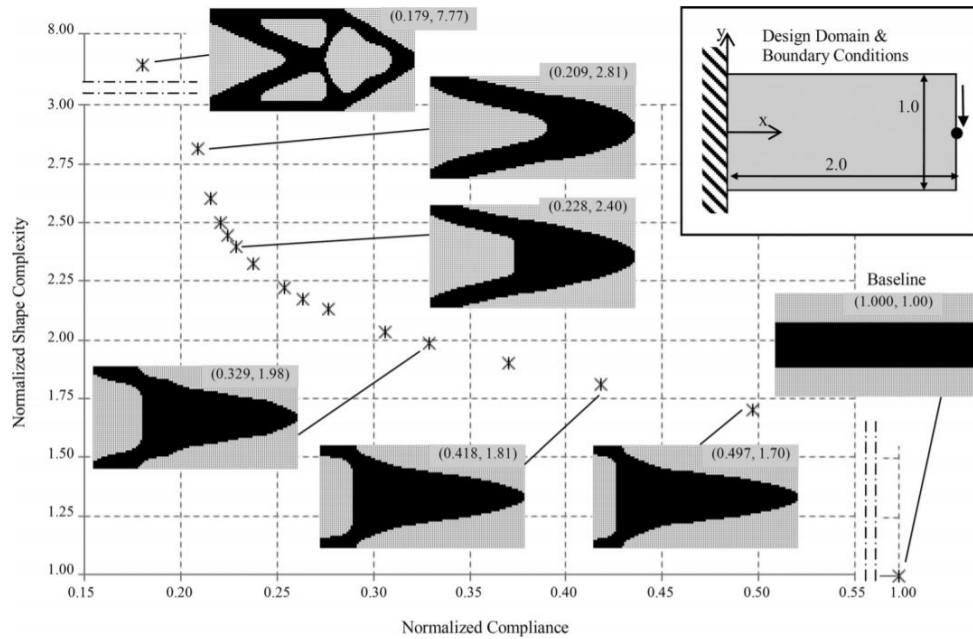


Figure 3.6 Plot of compliance verses shape complexity for cantilever problem^[25]

Four different types of design domains and loads are considered for minimum compliance. The aim of the optimization problem is to find the optimal material distribution, in terms of minimum compliance, whilst dealing with a constraint on the total amount of material. However, to deal with the manufacturing cost, an important attribute must be taken into account is the shape complexity.^[25]

Minimizing the compliance in structural problems is equivalent to maximizing the global stiffness of the structure under given external load. Meanwhile, the objective of the heat conduction cases is minimizing the maximum temperature under prescribed thermal loadings, and consequently, the temperature diffusion is maximized. The focus is restricted to minimum compliance problems with a constraint on the amount of material available. Therefore, the physical densities must be defined to adjust in such way that the volume constraint is satisfied.

Cantilever Problem

Cantilever problem is expressed as a horizontal deformable beam that is fixed supported at the one end and free at the other. The vertical loading is either a point load or distributed load with a value of 1 and it is located at the other end. Hence, the displacement will be computed by solving the finite element formulation. Generally, the important aspect evaluated in cantilever problem is the bending effect under loading condition.

Bridge Problem

Bridge problem is expressed as a horizontal block that is supported at each bottom corner by pin and roller. The block carries a vertical point load with a value of 1 at the bottom centre of the structure whilst the structure's weight is neglected.

MBB Problem

The MBB problem is expressed as a horizontal block that is supported at one end vertically by roller and the another bottom end by rolls. The beam sustains a point load with a value of 1 at the former top end.

Heat Transfer Problem

Heat transfer problem is expressed as a rectangular plate domain with a temperature of $T = 0^{\circ}C$ at heat sink on the middle of top face whilst insulated condition is applied at other boundaries. Here, thermal flux and heat convection are neglected in this problems. Heat generation is

distributed uniformly on the whole domain even on the void fluid element or fictitious element. So, the heat sink is proposed to absorb the heat whilst constant heat generation with the value of 0.01 is given until the system reaches a steady-state condition.

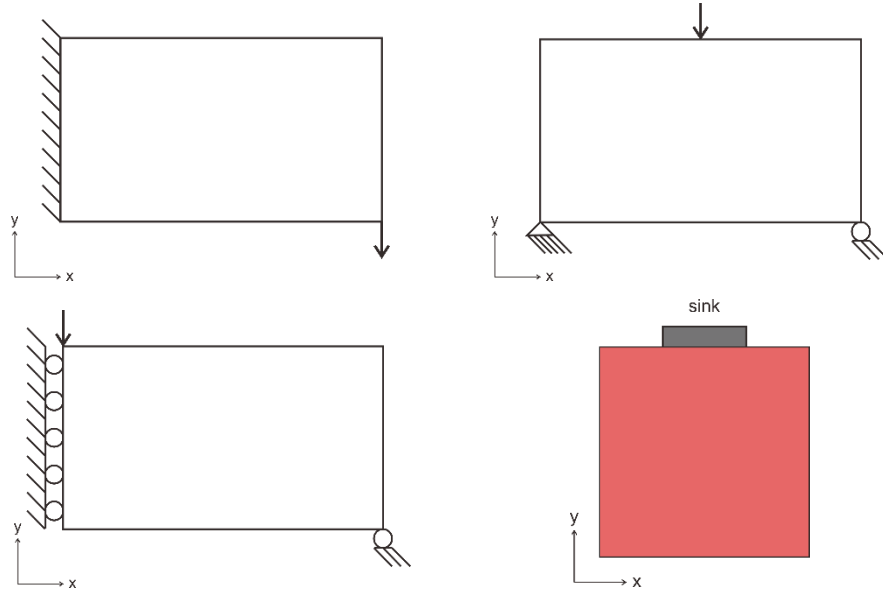


Figure 3.7 2D Cantilever problem (top left), 2D Bridge Problem (top right), 2D MBB Problem (bottom left), 2D Heat Problem (bottom right)

Table 3.1 Parameters used in 2D optimization problem

2D Problems	Element Area	Volume fraction	Filter Radius Size
Cantilever	90×30	0.5	2.5
Bridge	90×30	0.5	2.0
MBB	90×30	0.5	2.5
Heat Conduction	40×40	0.3	1.4

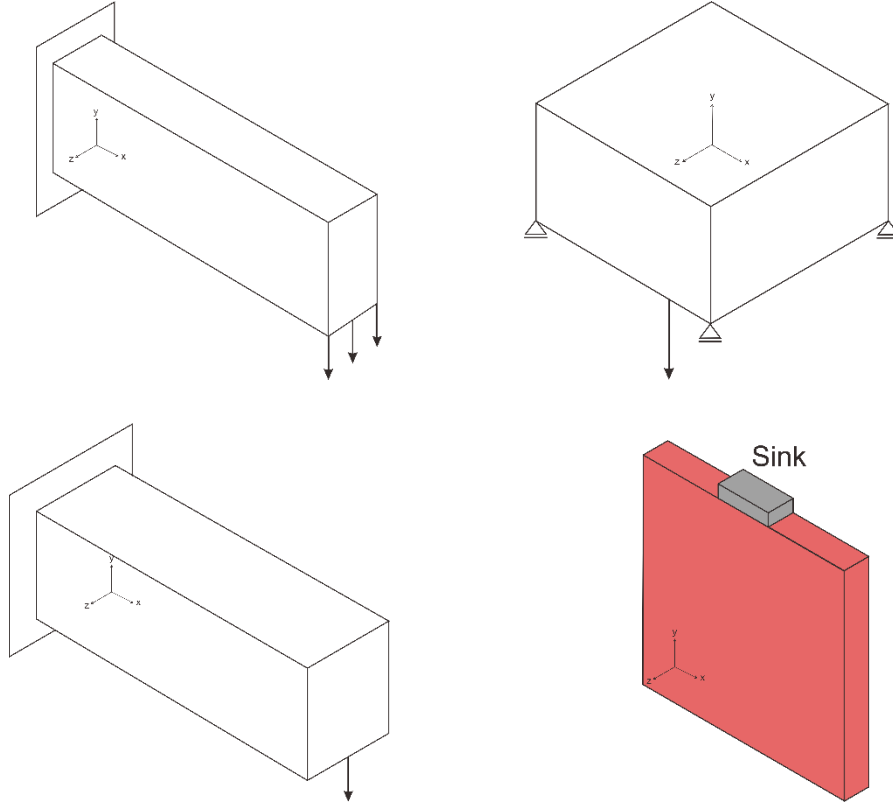


Figure 3.8 3D Cantilever problem (top left), 3D bridge problem (top right), 3D cantilever middle-scale problem (bottom left), 3D heat conduction problem (bottom right)

Table 3.2 Parameters used in 3D optimization problem

3D Problems	Element Area	Volume fraction	Filter Radius Size
Cantilever	$60 \times 20 \times 4$	0.3	1.5
Bridge	$40 \times 20 \times 40$	0.2	1.5
Cantilever M-S	$60 \times 20 \times 16$	0.15	1.5
Heat Conduction	$40 \times 40 \times 5$	0.3	1.4

3.3. Initialization

The performance of topology optimization methods is also affected by the starting point. All solvers are initialized as an homogeneous design $x_0 = Ve$ where V is the initial design variable and e is a vector of all ones. The algorithms in IPOPT and FMINCON are quite dependent on the starting points hence it should be initialized between lower and upper bounds. To make things easier, V is initialized to be the same as the prescribed volume fraction therefore it could be

determined that the value will not violate the design variables' bounds. Compared to SAND formulation, nested formulation does not require the displacement u_0 to be initialized.^[19]

3.4. Algorithm Parameter

Several optimization parameters are also set for each algorithm which are mainly affecting how each of the solver's iterative steps are taken. These parameters are detailed as follows.

Table 3.3 Parameter tuned in FMINCON^[19]

Parameter	Value	Description
Algorithm	interior point	Determine the optimization algorithm
Subproblem Algorithm	cg	Determine how the iteration step is calculated

Table 3.4 Parameter tuned in IPOPT^[19]

Parameter	Value	Description
mu strategy	monotone	Update strategy for barrier parameter
Limited memory max history	25	Maximum size of history in BFGS
Nlp scaling method	None	Technique for scaling the problem
Alpha for y	Full	Method to determine the step size of constraint multipliers
Recalc y	yes	Tells the algorithm to recalculate the multipliers at least square estimates

Table 3.5 Parameter tuned in SNOPT^[19]

Parameter	Value	Description
Subproblem algorithm	1 (Quasi Newton BFGS)	Determine how the iteration step is calculated

3.5. Stopping Criteria and Convergence Tolerance

The most commonly stopping criteria used in general nonlinear optimization problem is Karush-Kuhn-Tucker (KKT) optimality conditions that generalizes the method of Lagrange multipliers.

$$\begin{aligned}
 & \underset{x}{\text{minimize}} && f(x) \\
 & \text{subject to} && h_i(x) = 0 && i = 1, \dots, m \\
 & && g_i(x) \leq 0 && i = 1, \dots, t,
 \end{aligned} \tag{3.1}$$

where f is the objective function, h_i is the equality constraint function and g_i is the inequality constraint function. The number of equalities and inequalities are denoted by m and t respectively.

The Lagrange function is generated as follows:

$$L(x, \lambda, \mu) = f(x) + \sum_i^m \lambda_i h_i(x) + \sum_i^t \mu_i g_i(x), \tag{3.2}$$

where λ and μ are the Lagrange multipliers of the equality and inequality constraints, respectively.

The Karush-Kuhn-Tucker (KKT) theorem is then stated in below equation.

$$\begin{aligned}
 \nabla L(x^*, \lambda^*, \mu^*) &= 0 \\
 h_i(x^*) &= 0 && \forall i = 1, \dots, m \\
 g_i(x^*) &\leq 0 && \forall i = 1, \dots, t \\
 \mu_i^* &\geq 0 && \forall i = 1, \dots, t \\
 \mu_i^* g_i(x^*) &= 0 && \forall i = 1, \dots, t
 \end{aligned} \tag{3.3}$$

The solvers will try to compute the optimum solution iteratively until optimality conditions (KKT conditions) and feasibility conditions are lower than specified tolerances. Different compared to the convergence criteria of OC that depends on parameter change and feasibility error, not by Lagrangian multipliers. However, some maximum thresholds are given in order to prevent the solver of not being able to achieve the optimal design with a given tolerance.^[19]

Table 3.6 Parameter, description and values of the tolerance criteria^[19]

Solver	Parameter	Description	Value
MMA	kkt tol	Euclidean norm of the KKT error	1e-4
IPOPT	Tol	Tolerance of the NLP error	1e-6
SNOPT	Major optimality tolerance	Final accuracy of the dual variables	1e-6
FMINCON	Tolfun	Tolerance on the function value	1e-6
OC	change	Difference in the design variable	1e-4
OC, MMA	Feas tol	Euclidean norm of the feasibility error	1e-8
IPOPT	Constr viol tol	Tolerance of the constraint violation	1e-8
SNOPT	Major feasibility tolerance	Tolerance of the nonlinear constraint violation	1e-8
FMINCON	tolcon	Tolerance of the constraint violation	1e-8

Table 3.7 Parameter, description and values of termination criteria^[19]

Solver	Parameter	Description	Value
OC	loop	Maximum number of iterations	1000
MMA	max outer itn	Maximum number of iterations	1000
IPOPT	max iter	Maximum number of iterations	1000
SNOPT	Major iterations limit	Maximum number of iterations	1000
FMINCON	MaxIter	Maximum number of iterations	1000
IPOPT	max cpu time	Maximum CPU time	1200s (2D) / 12000s (3D)

CHAPTER IV

RESULT AND ANALYSIS

4.1. Result and Analysis on 2D Optimization Problems

4.1.1 Numerical and Topology Result on 2D Optimization Problems

All computations of 2D problems were done on an Intel Core i7-6700HQ CPUs, running at 2.60 GHz with 2 GB Memory for each core. Table 4.1 and Table 4.2 shows the final design topology of two-dimensional domain by all solvers for structural and heat conduction problem, respectively. Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4 shows the convergence plot of topology optimization process of two-dimensional domain by all solvers for cantilever, bridge, MBB and heat conduction problem respectively. Table 4.3, Table 4.4, Table 4.5, Table 4.6 collects the compliance values, the number of function evaluations and the CPU time required by all solvers for 2D cantilever, bridge, MBB and heat conduction problem, respectively.

Table 4.1 Topology layout of 2D structural optimization problem



















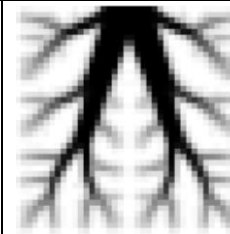
Solver	Cantilever Case	Bridge Case	MBB Case
OC			
FMINCON			
MMA			
IPOPT			
SNOPT			

Table 4.2 Topology layout of 2D heat transfer optimization problem

Solver	OC	FMINCON	MMA	IPOPT
2D Heat Conduction Case				

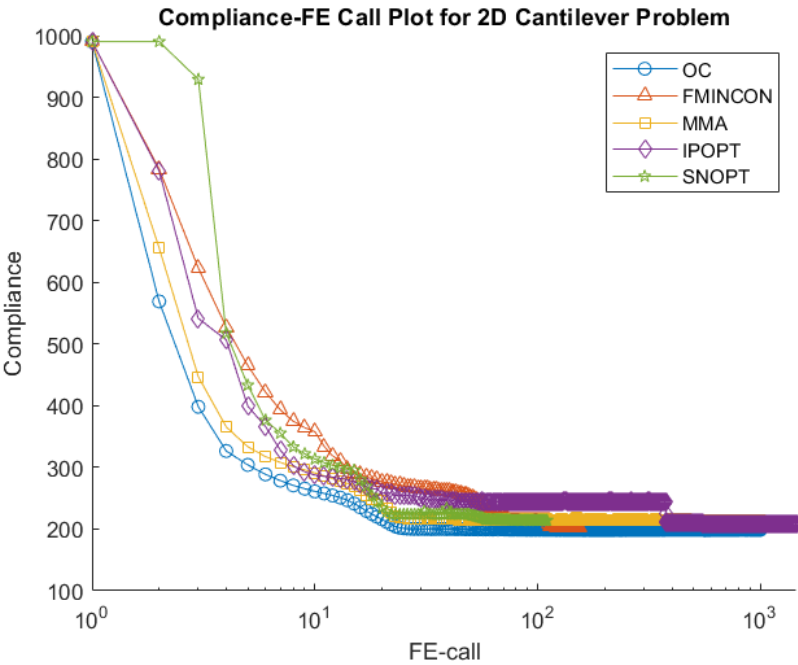


Figure 4.1 Convergence plot of 2D cantilever problem

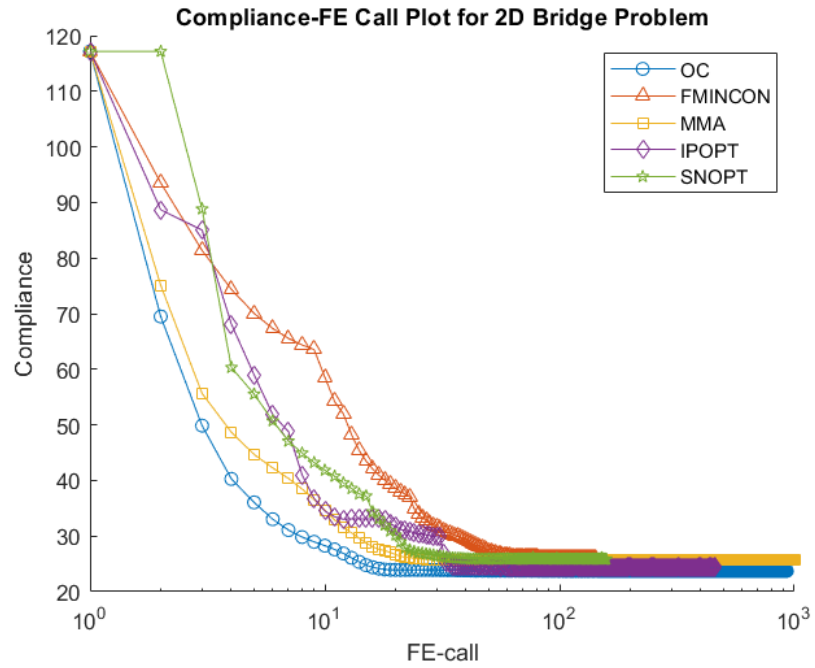


Figure 4.2 Convergence plot of 2D bridge problem

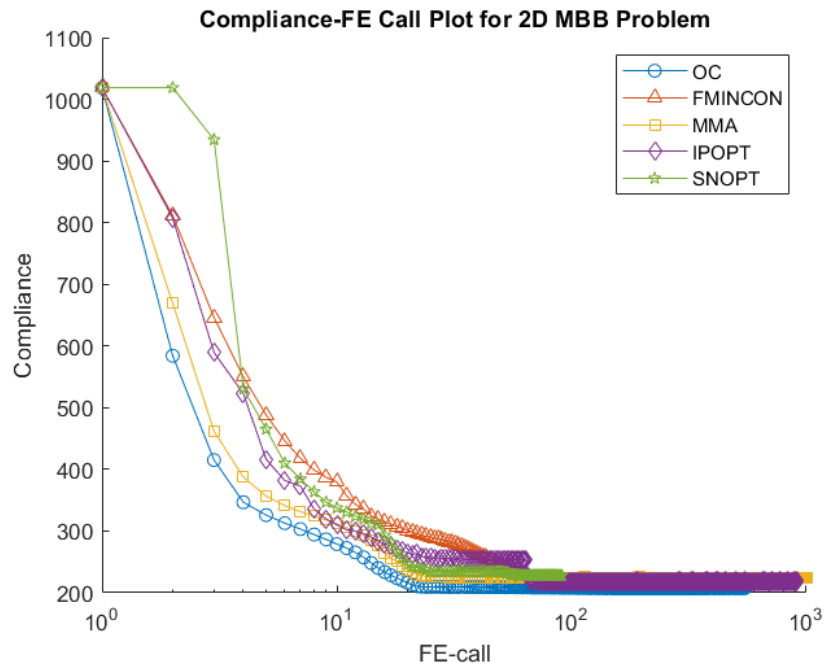


Figure 4.3 Convergence plot of 2D MBB problem

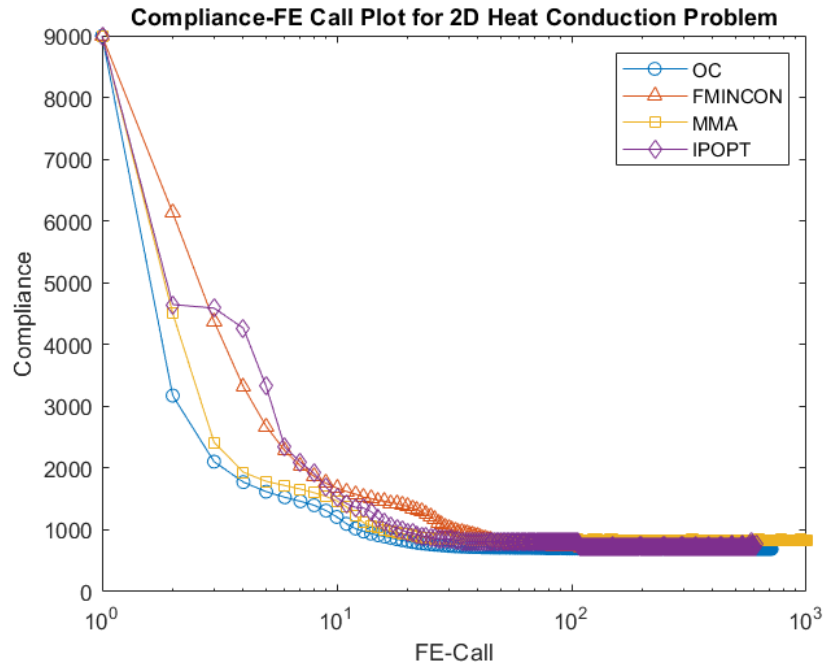


Figure 4.4 Convergence plot of 2D heat conduction problem

Table 4.3 Compliance, number of FE-call and CPU time of 2D cantilever problem

Solver	FE-Call	Compliance	CPU Time
OC	1000	198.865071	57.57
FMINCON	154	203.71222	31.4814
MMA	1000	214.498549	94.83
IPOPT	1447	208.848196	61.63471
SNOPT	110	214.149916	47.41929

Table 4.4 Compliance, number of FE-call and CPU time of 2D bridge problem

Solver	FE-Call	Compliance	CPU Time
OC	929	23.742007	54.24
FMINCON	178	24.972339	26.7727621
MMA	1000	25.647798	151.1
IPOPT	458	24.448718	20.8214338
SNOPT	158	25.845014	42.9657981

Table 4.5 Compliance, number of FE-call and CPU time of 2D MBB problem

Solver	FE-Call	Compliance	CPU Time
OC	547	207.438111	32.56
FMINCON	125	212.89	30.02711
MMA	1000	223.247279	100
IPOPT	912	219.142457	41.88771
SNOPT	91	228.039692	35.36788

Table 4.6 Compliance, number of FE-call and CPU time of 2D heat conduction problem

Solver	FE-Call	Compliance	CPU Time
OC	711	691.636991	5.466
FMINCON	224	708.266619	30.9533033
MMA	1000	828.672834	27.78
IPOPT	608	731.891967	8.4217077

4.1.2 Benchmarking Gradient-Based Methods for 2D Problems

In the process of benchmarking, few measures must be selected to be the most suitable to compare the topology optimization methods. The performance of a solver should be evaluated for the number of function evaluation, the CPU time, and the obtained objective function value. By using nested form to solve the topology optimization solver, it can be considered that the function evaluation is equivalent to the assembly of the stiffness matrix. Therefore, the performances of the methods should be evaluated towards the number of function evaluations. The y axis represent the objective function obtained whilst the x axis collects the number of function evaluations. This evaluation could point out the convergence trend of topology optimization processes. Due to several differences on how each solver works, attention should be given to the signification of function evaluation to ensure that it is the same as how iteration is done by each algorithm.

Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4 show the difference between solver's convergence performance whilst Table 4.3, Table 4.4, Table 4.5 and Table 4.6 collects the compliance values, number of function evaluation and CPU time of all solvers for 2D Topology optimization problem. OC has the lowest compliance values obtained but requires quite high FE-calls and CPU time. In contrast, CPU time required by OC in heat conduction problem is the least compared to other solvers. Moreover, OC's slope could be considered as the steepest curve among other solvers. Aside from OC, MMA is the second steepest curve while SNOPT shows constant compliance at the beginning of the iterations (FE-Calls) before reducing it. Compliance values obtained by MMA and SNOPT are adjacent to each other and considered to be high values among other solvers. However, MMA requires large FE-calls and CPU time in order to meet the convergence criteria. FMINCON's slope shows that the reduction of compliance values is much more gradual compared to the other solvers. FMINCON is more sensitive to the solver's parameter such as r_{min} and `volfrac` to avoid erratic result. FMINCON requires the least function evaluation and CPU time in all three structural cases unlike low FE-Call in SNOPT which resulted a mediocre topology that is suspected to be a premature termination. Similarly, the function evaluation needed by FMINCON to converge in heat conduction problem is the least among other solvers, nonetheless it requires the most CPU time. IPOPT's curve shows non continuous step trend on its curve with the final compliance values obtained are often lower than other three solvers. IPOPT also requires large number of FE-Calls to converge, yet the whole calculation does not require much computational effort due to the speed of each function evaluation. Both FMINCON and IPOPT yields the lowest compliance value outside of OC with several shifting between each other. Moreover, MMA and IPOPT shows slight sharp peaking on the compliance values from time to time after converge.

The density contour from the optimization process often has some difficulties when being integrated for manufacture. The desired material distribution is the least complicated design topology formed. This is due to the limitation of manufacturing and CAD model making. The resulting topology from each solver is included as one of the criteria to be benchmarked.

Table 4.1 shows the difference between solver's design topology for 2D structural optimization problem whilst Table 4.2 shows the solver's design topology layout for 2D

heat optimization problem. For the resulting topology, MMA has shown to be the best in terms of simplicity although the void zones still has some residual density value (not at the lower end of the density spectrum) and followed by OC which yields similar topology. The residual density value tends to be neglected or can be modelled as a thinned down material for weight saving due to less stiffness requirement compared to full density element. On the other hand, IPOPT yields a finer result but the resulting topology has a higher degree of complexity compared to MMA. This is due to the formation of branches like structure in the topology on Cantilever and MBB test case. But in the case of heat conduction, OC, FMINCON and IPOPT yield a more promising topology due to better root distribution unlike MMA. Naturally, this leads to lower temperature distribution. But due to the simplicity in MMA's heat conduction topology, the manufacturing process should be easier than other topologies. However, IPOPT has higher compliance compared to FMINCON due to substantial amount of grey present in the topology layout such shown in MMA's topology. In conclusion, FMINCON yields good topology results although in some cases, it could not yield a clean topology as shown by few density singularities in the topology. One thing to note is the heat conductivity topology result yielded from MMA and FMINCON is not symmetrical even though the prescribed boundary is symmetrical. Moreover, the root alike topology does not seem to be distributing well enough that yields to higher temperature distribution on MMA in comparison with other solvers. Finally, SNOPT who produces poor accurate designs has the most complex topology result which has a lot of branches like structure forming in the topology. This might be due to the premature termination of the solver. In the case of bridge problem, all of the solvers yielded good result and the resulting topology is very similar for each solvers. This might be caused by the simplicity of the bridge test case.

4.2. Result and Analysis on 3D Optimization Problems

4.2.1 Numerical and Topology Result on 3D Optimization Problems

All 3D problems are all computed on an Intel Xeon E3 1226 v3, running at 3.3 GHz with 4 GB Memory for each core. Table 4.7 and Table 4.8 shows the final design topology of three-dimensional domain by all solvers for structural problem and heat conduction problem, respectively. Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8 shows

the convergence plot of topology optimization process of three-dimensional domain by all solvers for cantilever, bridge, cantilever medium-scale and heat conduction problem, respectively. Table 4.9, Table 4.10, Table 4.11 and Table 4.12 collects the compliance values, the number of function evaluations and the CPU time required by all solvers for 3D cantilever, bridge, cantilever medium-scale problem and heat conduction problem, respectively.

Table 4.7 Topology layout of 3D structural optimization problem

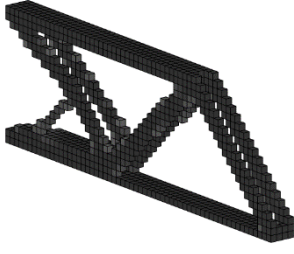
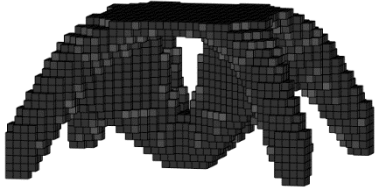
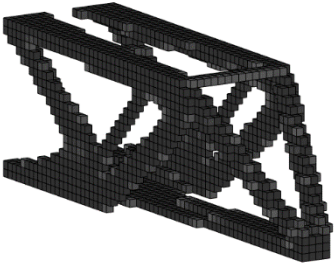
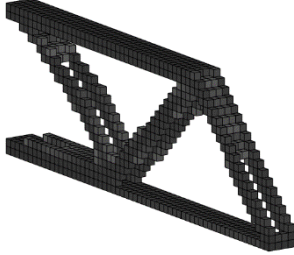
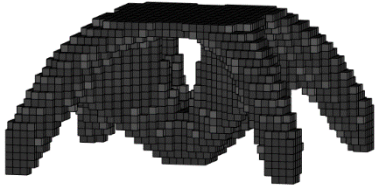
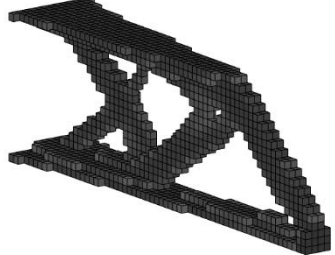
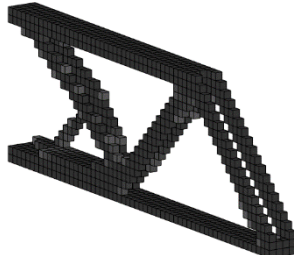
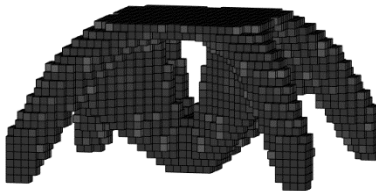
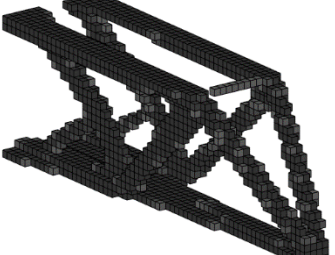
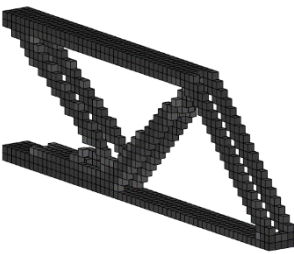
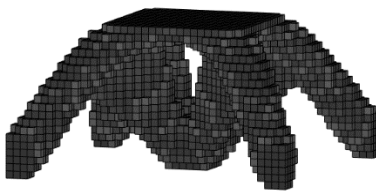
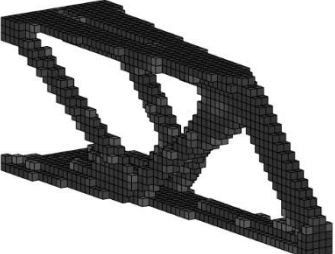
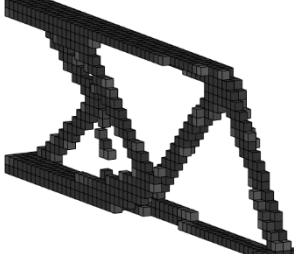
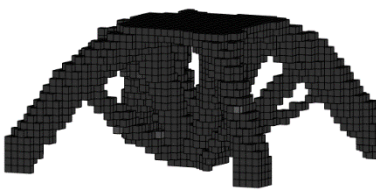
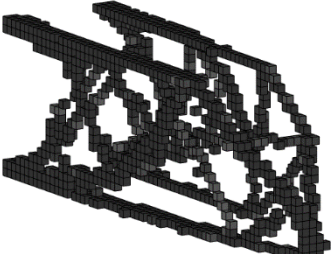
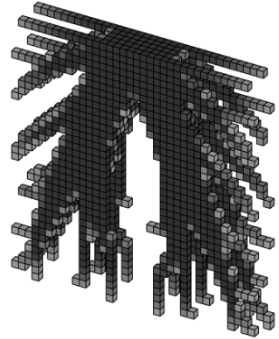
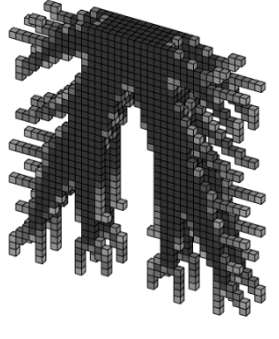
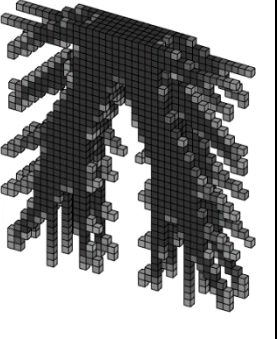
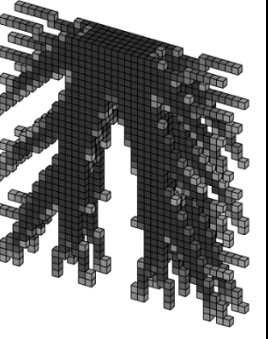
Solver	Cantilever Case	Bridge Case	Cantilever MS Case
OC			
FMINCON			
MMA			
IPOPT			
SNOPT			

Table 4.8 Topology layout of 3D heat transfer optimization problem

Solver	OC	FMINCON	MMA	IPOPT
3D Heat Conduction Case				

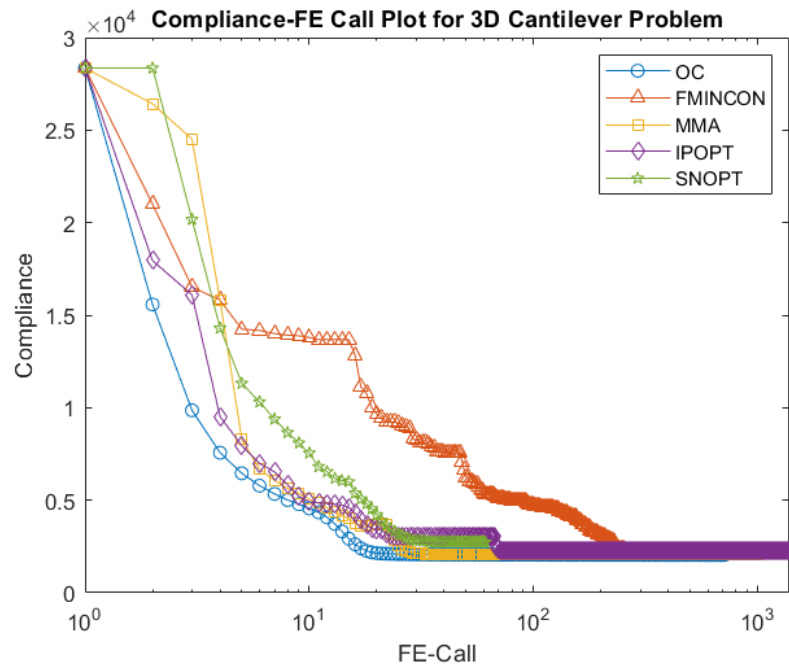


Figure 4.5 Convergence plot of 3D cantilever problem

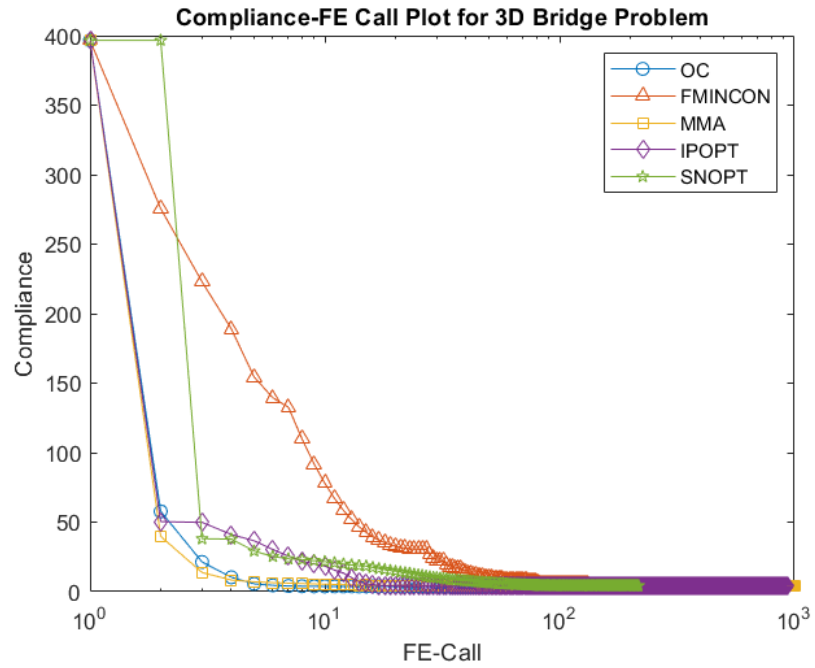


Figure 4.6 Convergence plot of 3D bridge problem

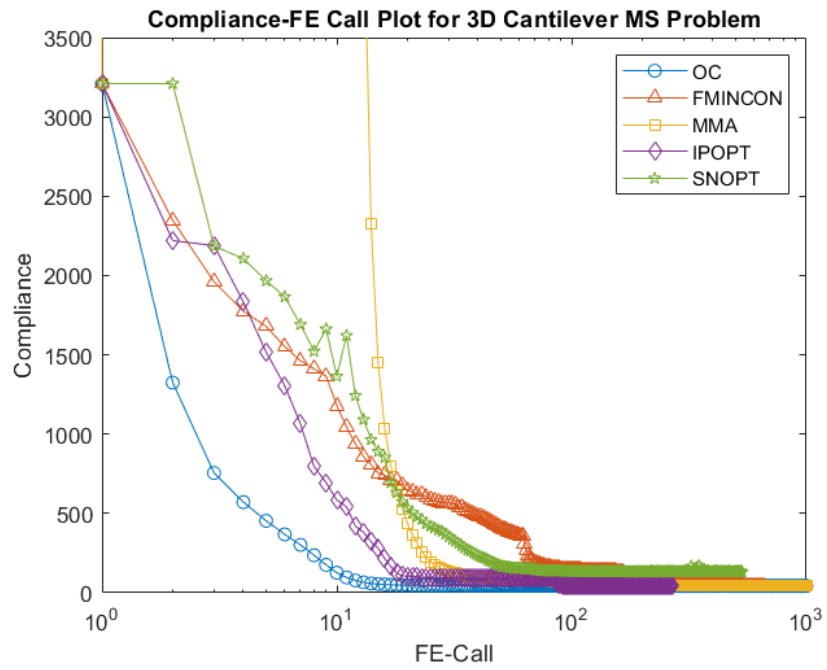


Figure 4.7 Convergence plot of 3D cantilever medium-scale problem

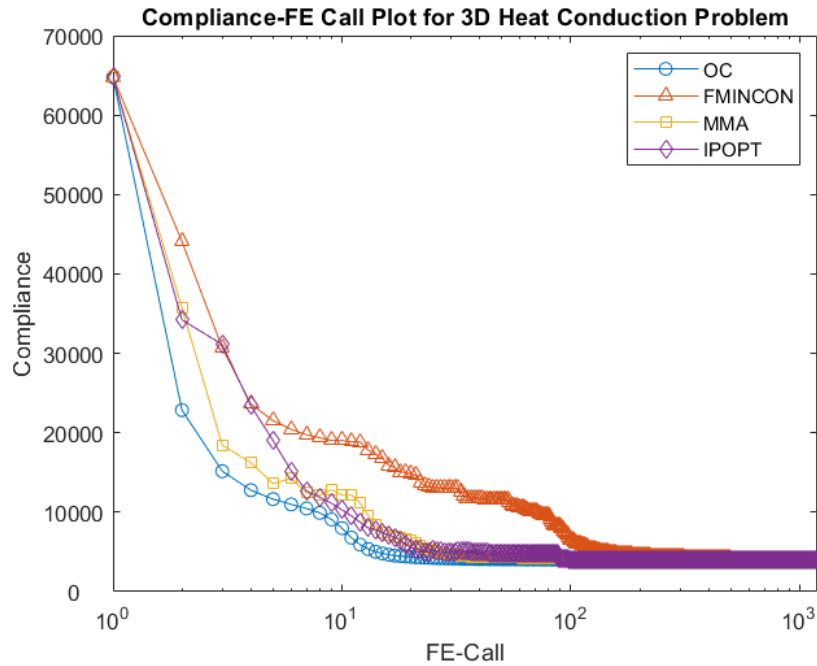


Figure 4.8 Convergence plot of 3D heat conduction problem

Table 4.9 Compliance, number of FE-call and CPU time of 3D cantilever problem

Solver	FE-Call	Compliance	CPU Time
OC	714	2052.507072	392.9
FMINCON	495	2178.014576	431.247523
MMA	1000	2054.141448	572.1
IPOPT	1383	2278.138928	798.2868768
SNOPT	61	2742.112676	86.771182

Table 4.10 Compliance, number of FE-call and CPU time of 3D bridge problem

Solver	FE-Call	Compliance	CPU Time
OC	209	3.720163	4030
FMINCON	881	3.756335	15285.1501
MMA	1000	3.75111	18240
IPOPT	929	3.862095	12487.78839
SNOPT	217	4.312656	10269.91511

Table 4.11 Compliance, number of FE-call and CPU time of 3D cantilever medium scale problem

Solver	FE-Call	Compliance	CPU Time
OC	1000	43.985828	5091
FMINCON	663	43.117825	3580.5159
MMA	1000	46.057962	5118
IPOPT	268	46.277265	1251.08455
SNOPT	532	131.871451	7219.258

Table 4.12 Compliance, number of FE-call and CPU time of 3D heat conduction problem

Solver	FE-Call	Compliance	CPU Time
OC	1000	3982.736759	211.3
FMINCON	681	4107.015917	254.4020315
MMA	1000	4057.41623	283.1
IPOPT	1193	4077.1675	277.4017956

4.2.2 Benchmarking on Gradient-Based Methods for 3D Problems

In this section, a benchmarking between Optimality Criteria (OC), Method of Moving Asymptotes (MMA), Interior Point Optimizer (IPOPT), Find Minimum of Constrained Nonlinear Multivariable Function (FMINCON) and Sparse Nonlinear Optimizer (SNOPT) of 3D problems is accomplished. The comparison is achieved based on the compliance values, function-evaluations, CPU time and the resulted topology design. As discussed earlier in the Initialization section, the starting points (initial designs and conditions) used in all solvers must be uniform in order to obtain fair benchmarking.

Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8 show the difference between solver's convergence performance whilst Table 4.9, Table 4.10, Table 4.11 and Table 4.12 collects the compliance values, number of function evaluation and CPU time of all solvers for 3D Topology optimization problem. Same as the characteristic shown in 2D problems, OC also has the lowest compliance value with steepest curve in comparison

with all other solvers in 3D problems except for the cantilever medium scale problem but still OC has favourable compliance value bested only by FMINCON, although this solver requires higher FE-Calls count and CPU time. However, CPU time needed by OC to complete the computation on 3D heat conduction problem is the least among other solvers. The compliance values obtained by MMA is in the second place compared to the other solvers, but its FE-Calls count and CPU time are among the highest. However, there are some erratic performance issues in the medium scale cantilever problem by MMA in which the compliance values experienced a huge peak at the beginning of the function evaluation. On the other hand, FMINCON is one of the solvers which has relatively low FE-Calls count and CPU time, although the decrement of compliance value is the most gradual among others. The utilization of conjugate gradient in FMINCON helps reduce the number of function evaluation required and the computational time. IPOPT and SNOPT shows several discontinuities in the compliance curves. However, in IPOPT, the compliance value consistently decreases whereas in SNOPT except in heat conduction problem, the compliance stays constant in the beginning of the iteration and sometimes it goes up at the middle of optimization process. The number of FE-Calls required in both solvers does not show any conclusive results as well as the CPU time which is not linearly affected by the function evaluation number. SNOPT attains the highest compliance values among other solvers. Overall, convergence plot of bridge problem and heat conduction problem show smooth continuous curves without any significant discontinuities compared to the other problems.

Table 4.7 shows the difference between solver's design topology for 3D structural optimization problem, whilst Table 4.8 shows the solver's design topology layout for 3D heat optimization problem. In structural problems, FMINCON and IPOPT yields a very similar topology result which might be attributed to the base algorithm used in both of these solvers which is the Interior Point Algorithm. Besides similar compliances, OC and MMA yields similar topology too. Design topology resulted from OC and MMA are considered as the most efficient structures due to the formation of the supporting beams. But if observed from a manufacturing standpoint, IPOPT and FMINCON has the most advantage due to its simplicity. SNOPT on the other hand yields the poorest design topology as the result of the complex webbing structure. Meanwhile, topology results

generated from heat conduction problem are similar for every optimization solver, differences usually occur on the topology's roots branching.

4.3. Topology Effect on Structural Stiffness

Compliance is directly related to structural strength and it can be expressed as strain energy. As the compliance value decreases, the displacement will decrease too. Considering the stress strain displacement relation, it is known that displacement is linearly proportional to stress and the smaller the stress, the higher the structural efficiency. The initial material distributions shifted to the optimum locations whereas the densities in trivial locations is made to voids. Therefore, the final structure would be more much more efficient.

Supporting beams formed in cantilever problem and MBB problem (See Table 4.1 and Table 4.7) resembles the Warren Truss whilst the topology layout in bridge problem resembles the Bowstring Truss. The supporting beams assemble to form a set of triangles (trusses) that serves as the load path which has an object to increase the structures' rigidity against the bending moment. In the bridge problem, the diagonal trusses sustain the prescribed external load in the centre and then the load is propagated through to the neighbouring supporting beam in order to increase structural efficiency.

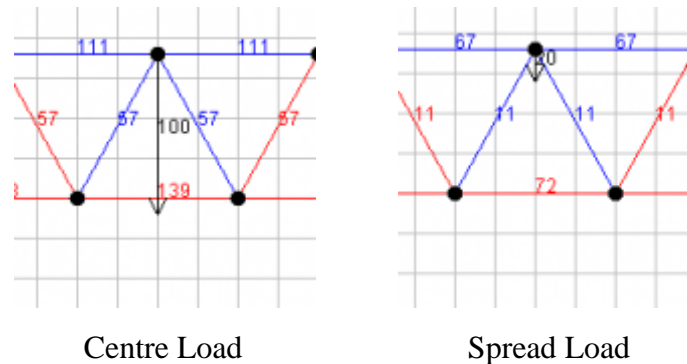


Figure 4.9 The load percentage distribution on the Warren Truss Bridge^[26]

The high compliance values obtained in 2D cantilever problem by SNOPT is estimated due to the middle supporting beam in the structure that is angled at a high degree from the normal surface of upper beam and therefore, almost tangent to the upper beam. On the other hand, design topology of 2D MBB problem solved by SNOPT also has an irrelevant supporting beams that does not match the formation of Warren Truss. Therefore, the distribution of the load through the trusses is not effective.



Figure 4.10 Irrelevant truss on 2D MBB problem topology result by SNOPT

Furthermore, there are also a premature formation of supporting beam in several topology result. This is due to the differences of the algorithms in each solver work on how to distribute the material densities in order to achieve the lowest compliance value. These phenomena are doubtlessly not desired to be formed in the topology result.

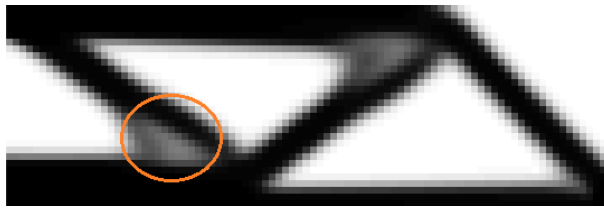


Figure 4.11 Premature formation of supporting beam on 2D MBB problem topology result by IPOPT

4.4. Topology Effect on Heat Conductivity

A number of topology optimization solvers have been proposed to solve steady-state heat conduction problems. The stationary heat conduction TO problems involving 2D and 3D domain are solved using density-based approach. Using density-based approach, the final topology sometimes shows substantial intermediate densities. However, one can interpret such areas as consisting of composite.

As Table 4.2 and Table 4.8 shown, the temperature on high density areas at the sink base are very low while the element furthest from the sink with void density have very high temperature. According to the heat conduction equation, areas with high heat transfer coefficient could help heat flows more easily and vice versa. Same as the tree root, the more dispersed the branching on topology, the better heat diffusivity.

CHAPTER V

CONCLUSION AND FUTURE WORKS

5.1. Conclusions

Structural and heat transfer topology optimization problems are differentiated by its domain i.e. two-dimensional and three-dimensional problems. All the problems are solved using gradient-based method i.e. Optimality Criteria (OC), Method of Moving Asymptotes (MMA), Interior Point Optimizer (IPOPT), Find Minimum of Constrained Nonlinear Multivariable Function (FMINCON) and Sparse Nonlinear Optimizer (SNOPT). This research is conducted in order to find the best gradient-based method and the optimum design for all 2D and 3D problems.

Nested approach is chosen to formulate the topology optimization problems. The formulation of the problem is minimizing the compliance of the structure and heat domain subject to the volume constraint. The Solid Isotropic Material Penalization (SIMP) material interpolation scheme is implemented in the density optimization to model the structural and thermal topology by penalizing the material density. The 2D optimization problems is divided into 4 examples, i.e. cantilever problem, bridge problem, MBB problem and heat conduction problem whilst 3D optimization problem is divided into two cantilever problem (small-scale and medium-scale problem), bridge problem and heat conduction problem.

1. The gradient-based optimization solvers have been successfully implemented in both structural and heat transfer topology optimization test cases

The implementation is supported by some publicly available optimization algorithm code. This are listed as follows:

- Optimality Criteria (OC) method is applied from educational paper entitled “*Efficient Topology Optimization in MATLAB using 88 Lines of Code*” by Sigmund O.
- Method of Moving Asymptotes (MMA) is applied from journal publications entitled “*MMA and GCMMA – Two Methods for Nonlinear Optimization*” by Kristen Svanberg.

- Find Minimum of Constrained Nonlinear Multivariable Function (FMINCON) is applied from MATLAB internal library consisting set of nonlinear optimization algorithms.
 - Interior Point Optimizer (IPOPT) is applied from software package for large-scale nonlinear optimization from COIN-OR.
 - Sparse Nonlinear Optimizer (SNOPT) is a commercial nonlinear solver developed by Stanford Systems Optimization Laboratory (SOL) with TOMLAB.
2. OC is considered as the best topology optimization method in terms of compliance minimization

After computations of all topology optimization problems are performed, it can be generalized that OC is the best optimization method in terms of the lowest compliance values because its algorithm is designed to solve optimization problem with monotonicity properties of negative objective function gradient only. However, MMA and IPOPT yields more realistic topology result to be manufactured in 2D and 3D problem, respectively. IPOPT can be considered as a versatile solver due to its simplicity to be integrated for any optimization problems. For FMINCON, it requires the usage of an exact Hessian formulation in order to solve 3D problems yet the resulting performances still do not outperform other solvers except for the 3D cantilever medium-scale problem. However, the use of exact Hessian in FMINCON lowers the computation time compared to standard FMINCON usage without any hessian input. MMA also yields to good result in compliance values and topology layout yet it is somehow mediocre due to the occurrence of slight divergence in the convergence values of 3D cantilever medium-scale problem. Moreover, MMA is highly affected by its termination criteria. SNOPT on the other hand, is the worst performer in this benchmarking which might be attributed to premature termination of the solver.

3. Optimum design for structural and heat transfer optimization problem

The optimum design for cantilever and MBB topology optimization problem are the formation of the Warren Truss whilst the optimum design for bridge topology optimization problem is the formation of Bowstring Truss. A series of triangular truss is formed in object to spread out the external load. The solver with the most fulfilling topology layout to the above mentioned criteria is MMA in 2D problems and IPOPT in 3D problems. The premature formation of supporting beam is undesirable by cause of uncertainty in manufacturing process.

Meanwhile, the topology results generated from the heat conduction problems show the distribution of root alike topology that decrease the maximum temperature.

5.2. Future Works

Future works should be done in order to improve the efficiency of the implementations of gradient-based optimization solvers. A benchmarking between both gradient-based and non-gradient based approach in the scope of 3D topology optimization problem can be a consideration for further research in order to investigate deepen towards the characteristics of non-gradient based. A performance profiles also can be added in the benchmarking profiles to inspect the percentage of problems solved by the solvers. It is also suggested to focus on robust optimization method that able to overcome the uncertainty issues.

REFERENCE

- [1] Turner, M. *The Current State, Outcome and Vision of Additive Manufacturing. Journal of Welding and Joining* 33, 1–5. 2015. doi:10.5781/jwj.2015.33.6.1
- [2] Krog, L., Tucker, A. & Rollema, G. *Application of topology, sizing and shape optimization methods to optimal design of aircraft components. 3rd Altair UK HyperWorks Users Conference* 1–12. 2002.
- [3] Lundgren, J., Klarbring, A., Lundgren, J. E. & Thore, C. J. *Topology optimization of periodic 3D heat transfer problems with 2D design. Structural and Multidisciplinary Optimization* 60, 2295–2303. 2019. doi:10.1007/s00158-019-02319-2
- [4] Rao, S. S. *Engineering Optimization: Theory and Practice: Fourth Edition. Engineering Optimization: Theory and Practice: Fourth Edition* 1–813. John Wiley and Sons. 2009. doi:10.1002/9780470549124
- [5] Rojas-Labanda, S. & Stolpe, M. *Benchmarking optimization solvers for structural topology optimization. Structural and Multidisciplinary Optimization* 52, 527–547. 2015. doi:10.1007/s00158-015-1250-z
- [6] Snyman, J. A. & Wilke, D. N. *Practical mathematical optimization: Basic optimization theory and gradient-based algorithms. in Springer Optimization and Its Applications* 133, 1–364. Springer International Publishing. 2018.
- [7] Nathan. *Development of Level Set approach for Gradient and Non-Gradient Based Structural Topology Optimization*. Undergraduate Thesis, Aerospace Engineering, Institut Teknologi Bandung, Bandung, Indonesia. 2019.
- [8] Iga, A., Nishiwaki, S., Izui, K. & Yoshimura, M. *Topology optimization for thermal conductors considering design-dependent effects, including heat conduction and convection. International Journal of Heat and Mass Transfer* 52, 2721–2732. 2009. doi:10.1016/j.ijheatmasstransfer.2008.12.013
- [9] Rozvany, G. *Topology Optimization of Structures and Composite Continua*. Dordrecht: Kluwer. 2000.

- [10] Johnsen, S. *Structural Topology Optimization: Basic Theory, Methods and Applications*. Master Thesis, Department of Engineering Design and Materials, *Norwegian University of Science and Technology*, Trondheim, Norway. 2013.
- [11] Guo, X. et al. *Self-supporting structure design in additive manufacturing through explicit topology optimization*. *Computer Methods in Applied Mechanics and Engineering* 323, 27–63. 2017. doi:10.1016/j.cma.2017.05.003
- [12] Guirguis, D. & Aly, M. F. *A derivative-free level-set method for topology optimization*. *Finite Elements in Analysis and Design* 120, 41–56. 2016. doi:10.1016/j.finel.2016.06.002.
- [13] Bendsøe, M.P., Sigmund, O. *Material interpolation schemes in topology optimization*. *Archive of Applied Mechanics* 69, 635–654. 1999. doi:10.1007/s004190050248.
- [14] Pedersen, C.G., Lund, J.J., Damkilde, L. and Kristensen, A.S. *Topology Optimization - Improved Checker-Board Filtering with Sharp Contours*, in: *19th Nordic Seminar on Computational Mechanics*. 2006.
- [15] De, S. *Introduction to Finite Elements* [Powerpoint Presentation]. MANE 4240 & CIVL 4240. Available at <https://homepages.rpi.edu/~des/4NodeQuad.pdf> (Accessed: 15 April 2020)
- [16] Liu, K., Tovar, A. *An efficient 3D topology optimization code written in Matlab*. *Structural and Multidisciplinary Optimization* 50, 1175–1196. 2014. doi:10.1007/s00158-014-1107-x
- [17] Gersborg-Hansen, A., Bendsøe, M.P., Sigmund, O. *Topology optimization of heat conduction problems using the finite volume method*. *Structural and Multidisciplinary Optimization* 31, 251–259. 2006. doi:10.1007/s00158-005-0584-3
- [18] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S., and Sigmund, O. *Efficient topology optimization in MATLAB using 88 lines of code*. *Structural and Multidisciplinary Optimization* 43, 1–16. 2011. doi:10.1007/s00158-010-0594-7
- [19] Rojas-Labanda, S., Stolpe, M. *Benchmarking optimization solvers for structural topology optimization*. *Structural and Multidisciplinary Optimization* 52, 527–547. 2015. doi:10.1007/s00158-015-1250-z
- [20] Hicken J.E., Alonso J.J. *Gradient-based Optimization* [Powerpoint Presentation]. AA222: *Introduction to Multidisciplinary Design Optimization*. Available at

http://adl.stanford.edu/aa222/Lecture_Notes_files/AA222-Lecture3.pdf (Accessed: 18 April 2020)

- [21] Ghaemi, M. *Topology Optimization Problems Using Optimality Criteria Methods*. PhD Thesis, Faculty of Civil Engineering, *Budapest University of Technology and Economics*, Hungary. 2009.
- [22] Svanberg, K. (2007): *MMA and GCMMA – two methods for nonlinear optimization*. *Kth I*, 1–15.
- [23] MathWorks, (2020). *fmincon* (R2020a). Available at <https://www.mathworks.com/help/optim/ug/fmincon.html> (Accessed: 20 April 2020)
- [24] Wächter, A., Biegler, L.T. On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming* 106 (1): 25–57. 2006. doi:10.1007/s10107-004-0559-y.
- [25] Gill, P.E., Murray, W., Saunders, M.A. *SNOPT: An SQP algorithm for large-scale constrained optimization*. *SIAM Journal on Optimization* 12, 979–1006. 2002. doi:10.1137/S1052623499350013
- [26] Boon, G. *Warren Truss*. [online] Garret’s Bridges. Available at <https://www.garrettsbridges.com/design/warren-truss/> (Accessed: 19 May 2020)
- [27] Hamza, K., Aly, M., Hegazi, H. *A Kriging-Interpolated Level-Set Approach for Structural Topology Optimization*. *ASME*. 2014. doi: 10.1115/1.4025706
- [28] Wang, X., Hu, P., Kang, Z. *Layout optimization of continuum structures embedded with movable components and holes simultaneously*. *Structural and Multidisciplinary Optimization* 61, 555–573. 2020. doi:10.1007/s00158-019-02378-5
- [29] Fanni, M., Shabara, N., Alkalla, M. *A Comparison between Different Topology Optimization Methods*. *Engineering Journal*. 2013.
- [30] Bourdin, B. *Filters in topology optimization*. *International Journal for Numerical Methods in Engineering* 50, 2143–2158. 2001. doi:10.1002/nme.116
- [31] Manuel, M.C.E., Lin, P.T., *Heat Exchanger Design with Topology Optimization*, in: *Heat Exchangers - Design, Experiment and Simulation*. *InTech*. 2017. doi:10.5772/66961

- [32] Yoshimura, M., Shimoyama, K., Misaka, T. and Obayashi, S. *Topology optimization of fluid problems using genetic algorithm assisted by the Kriging model. International Journal for Numerical Methods in Engineering* 109, 514–532. 2017. doi:10.1002/nme.5295
- [33] Yoon, M., Koo, B. *Topology design optimization of conductive thermal problems subject to design-dependent load using density gradients. Advances in Mechanical Engineering* 11. 2019. doi:10.1177/1687814019850735

Appendix A

2D Structural Topology Optimization

OC

```
function OC_top88_2D(nelx,nely,volfrac,penal,rmin,ft)
%% MATERIAL PROPERTIES
tic
nelx = 90;
nely = 30;
volfrac = 0.5;
penal = 3.0;
rmin = 2.5;
ft = 1;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% PREPARE FINITE ELEMENT ANALYSIS
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -
1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%% DEFINE LOADS AND SUPPORTS
% MBB PROBLEM
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)], [2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% CANTILEVER PROBLEM
F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [1:2*(nely+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% BRIDGE PROBLEM
F = sparse((nely+1)*(nelx)+2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [(2*nely)+1,2*(nely+1),2*(nely+1)*(nelx+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
```

```

for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
x = repmat(volfrac,nely,nelx);
xPhys = x;
loop = 0;
change = 1;
feas = 1;
maxloop = 1000;
tolx = 1e-4;
fileID = fopen('OC_2D_Data.txt','w');
videocut = false;
%% START ITERATION
while ((change > tolx) & (loop < maxloop))
    loop = loop + 1;
    %% FE-ANALYSIS
    sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
    K = sparse(iK,jK,sK); K = (K+K')/2;
    U(freedofs) = K(freedofs,freedofs)\F(freedofs);
    %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
    c = sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
    dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
    dv = ones(nely,nelx);
    %% FILTERING/MODIFICATION OF SENSITIVITIES
    if ft == 1
        dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
    elseif ft == 2
        dc(:) = H*(dc(:)./Hs);
        dv(:) = H*(dv(:)./Hs);
    end
    %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES
    l1 = 0; l2 = 1e9; move = 0.2;
    while (l2-l1) > 1e-8
        lmid = 0.5*(l2+l1);
        xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid))));
        if ft == 1
            xPhys = xnew;
        elseif ft == 2
            xPhys(:) = (H*xnew(:))./Hs;
        end
        if sum(xPhys(:)) > volfrac*nelx*nely, l1 = lmid; else l2 = lmid; end
    end
end

```

```

change = max(abs(xnew(:)-x(:)));
x = xnew;
OCtime(loop) = toc;
xmat = 1-xPhys;
%% PRINT RESULTS
fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, ...
    mean(xPhys(:)),change);
%% PLOT DENSITIES
    %% Plot Density & Temperature Distribution
if change < 0.01
    videocut = true;
end
if videocut == false
    figure(1); imagesc(x),colormap(flipud(gray)),caxis([0 1]);axis equal;
axis off; drawnow;
end
OC_2D_Data(loop) = c;
fprintf(fileID, '%f\n',OC_2D_Data(loop));
end
end

```

MMA

```

function f = MMA_top88_2D(nely,nelx,volfrac,penal,rmin)
%% MATERIAL PROPERTIES
tic
nelx = 90;
nely = 30;
volfrac = 0.5;
penal = 3;
rmin = 2.0;
%% MMA SOLVER PARAMETERS
m = 1;
n = nelx*nely;
xmin(1:n,1) = 0.1;
xmax(1:n,1) = 1;
a0 = 1;
a = 0;
c = 100;
d = 1;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0(1:nelx,1:nely) = volfrac;
x0 = x0(:);
x = [0.8*x0 0.5*x0 x0];
% UPPER AND LOWER BOUNDS
temp = 0.5*(max(x0)-min(x0));
initlow = x0 - temp;
initup = x0 + temp;
low = initlow;
upp = initup;
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;

```

```

for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
iter = 0;
feas = 1;
itermax = 1000;
feastol = 1e-8;
kkttol = 1e-4;
kktnorm = kkttol+10;
MMA_2D_Data = [];
while ((kktnorm > kkttol) & (iter < itermax) || (feas > feastol))
    iter = iter+1;
    %%% Some vectors are updated:
    xval = x(:,2+iter);
    xold1 = x(:,2+iter-1);
    xold2 = x(:,2+iter-2);
    %%% The user should now calculate function values and gradients
    %%% of the objective- and constraint functions at xval.
    %%% The results should be put in f0val, df0dx, fval and dfdx.
    [f0val,df0dx] = MMAGradient(xval,nely,nelx,penal,iH,jH,sH,H,Hs);
    fval = sum(xval) - volfrac*n;
    dfdx(1:n) = 1;
    %%% The MMA subproblem is solved at the point xval:
    [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
        mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
            f0val,df0dx,fval,dfdx,low,upp,a0,a,c,d);
    %%% The residual vector of the KKT conditions is calculated:
    [residu,kktnorm,residumax,feas] = ...
        kktcheck(m,n,xmma,ymma,zmma,lam,xsi,eta,mu,zet,s, ...
            xmin,xmax,df0dx,fval,dfdx,a0,a,c,d);
    x = [x xmma];
    %%% Saving data compliance
    MMAtime(iter) = toc;
    MMA_2D_Data = [MMA_2D_Data f0val];
    change = max(abs(xval - xold1));
    %% PRINT RESULTS
    fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',iter,f0val, ...
        mean(xmma(:)),change);
    %% PLOT DENSITIES
    x2 = vec2mat(xval',nelx);
    xmat = 1-x2;
    colormap(gray); imagesc(1-x2); caxis([0 1]); axis equal; axis off; drawnow;
end

```

```

save('MMA_2D_Time.mat','MMAtime');
save('MMA_2D_Topology.mat','xmat');
%% SAVE COMPLIANCE DATA MMA
fileID = fopen('MMA_2D_Data.txt','w');
for i = 1 : iter
    fprintf(fileID,'%f\n',MMA_2D_Data(i));
end
fclose(fileID);
end
% -----
function [c,dcvect] = MMAGradient(x,nely,nelx,penal,iH,jH,sH,H,Hs);
warning('off','all');
M = vec2mat(x,nelx);
%% MATERIAL PROPERTIES
E0 = 1;
Emin = 1e-9;
ft = 1;
nu = 0.3;
%% PREPARE FINITE ELEMENT ANALYSIS
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]); % Element
Stiffness
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx); % Node numbering ID
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -
1],nelx*nely,1); % Nodes ID at given element
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%% DEFINE LOADS AND SUPPORTS
% MBB PROBLEM
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% CANTILEVER PROBLEM
F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [1:2*(nely+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% BRIDGE PROBLEM
F = sparse((nely+1)*(nelx+2),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [(2*nely)+1,2*(nely+1),2*(nely+1)*(nelx+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+M(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((Emin+M.^penal*(E0-Emin)).*ce));

```



```

dc = -penal*(E0-Emin)*M.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
if ft == 1
dc(:) = H*(M(:).*dc(:))./Hs./max(1e-3,M(:));
elseif ft == 2
dc(:) = H*(dc(:)./Hs);
dv(:) = H*(dv(:)./Hs);
end
% Change the matrix x to vector
dcvect = [];
for i = 1:nely
    for j = 1:nelx
        dcvect = [dcvect;dc(i,j)];
    end
end
end
end

```

FMINCON

```

function history = fmincon_top88_2D
tic
%% SET UP SHARED VARIABLES WITH OUTFUN
history.x = [];
history.fval = [];
%% MATERIAL PROPERTIES
nelx = 90;
nely = 30;
volfrac = 0.5;
penal = 3.0;
rmin = 2.5;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0(1:nelx,1:nely) = volfrac;
xm = x0(:);
% UPPER AND LOWER BOUNDS
nvar=nelx*nely;
lb(1:nvar) = 0.0;
ub(1:nvar) = 1.0;
eq = ones(1,nelx*nely);
totalvolfrac = nely*nelx*volfrac;
%% PREPARE FINITE ELEMENT ANALYSIS
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -
1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%% DEFINE LOADS AND SUPPORTS

```

```

% MBB PROBLEM
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)], [2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% CANTILEVER PROBLEM
F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [1:2*(nely+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% BRIDGE PROBLEM
F = sparse((nely+1)*(nelx)+2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [(2*nely)+1,2*(nely+1),2*(nely+1)*(nelx+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% CALLBACK OBJECTIVE FUNCTION AND GRADIENT OBJECTIVE
objgrad =
@(x) funcfdf2D(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x);
hessfunc =
@(x,lamda)hessinterior(nely,nelx,penal,E0,Emin,edofMat,KE,iK,jK,freedofs,U,F,
nvar,x);
options =
optimoptions('fmincon','OutputFcn',@outfun,'SpecifyObjectiveGradient',true,...
.
    'SubproblemAlgorithm','cg','TolFun',1e-6,'TolCon',1e-
8,'MaxIteration',1000,...
    'MaxFunctionEvaluations', 10000,'HessianFcn',hessfunc)
% Run FMINCON.
delete FMINCON_2D_Data.txt
[x,fval] = fmincon(objgrad,xm,[],[],eq,totalvolfrac,lb,ub,[],options);
FMINCONtime = toc;
save('FMINCON_2D_Time.mat','FMINCONtime');
% HISTORICAL DATA COMPLIANCE

```

```

function stop = outfun(x,optimValues,state)
    stop = false;
    switch state
        case 'iter'
            history.fval = [history.fval; optimValues.fval];
            history.x = [history.x x];
        end
    end
%% PLOT DENSITIES
iter = size(history.x);
for i = 1:iter(2)
    x2 = vec2mat(history.x(:,i)',nelx);
    xmat = 1-x2;
    colormap(gray); imagesc(1-x2); caxis([0 1]); axis equal; axis off;
drawnow;
end
save('FMINCON_2D_Topology.mat','xmat');
end
% -----
function [c,dcvect] =
funcfdf2D(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x)
warning('off','all');
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
dctrans = dc';
dcvect = dctrans(:);
%% SAVING COMPLIANCE DATA AND CPU TIME
FEtime = toc;
diary FMINCON_2D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function Hess = hessinterior
(nely,nelx,penal,E0,Emin,edofMat,KE,iK,jK,freedofs,U,F,nvar,x)
xPhys = vec2mat(x,nelx);
% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
% SECOND DERIVATIVE
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
ddc = 2*(penal*(E0-Emin)*xPhys.^(penal-1)).^2.*(1./(Emin+xPhys.^penal*(E0-Emin))).*ce;

Hvect = zeros(nvar,1); a = zeros(nvar,1); m = 1;
for i = 1:nely

```

```

        for j = 1:nelx
            a(m) = m;
            Hvect(m) = ddc(i,j);
            m = m + 1;
        end
    end
Hess = sparse(a,a,Hvect,nvar,nvar);
end

```

IPOPT

```

function IPOPT_top88_2D
tic
%% MATERIAL PROPERTIES
nelx = 90;
nely = 30;
volfrac = 0.5;
penal = 3.0;
rmin = 2.0;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
xm(1:nelx,1:nely) = volfrac;
x0 = xm(:);
% UPPER AND LOWER BOUNDS
nvar = nely*nelx;
lbtemp(1:nvar) = 0.0;
ubtemp(1:nvar) = 1.0;
options.lb = lbtemp;           % Lower bounds on x
options.ub = ubtemp;          % Upper bounds on x
options.cl = nvar*volfrac-0.001; % Lower bounds on constraints.
options.cu = nvar*volfrac+0.001; % Upper bounds on constraints.
%% PREPARE FINITE ELEMENT ANALYSIS
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12; A12' A11]+nu*[B11 B12; B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -
1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
% MBB PROBLEM
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% CANTILEVER PROBLEM
F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);

```

```

fixeddofs = [1:2*(nely+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% BRIDGE PROBLEM
F = sparse((nely+1)*(nelx)+2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [(2*nely)+1,2*(nely+1),2*(nely+1)*(nelx+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
for j1 = 1:nely
e1 = (i1-1)*nely+j1;
for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
e2 = (i2-1)*nely+j2;
k = k+1;
iH(k) = e1;
jH(k) = e2;
sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
end
end
end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% IPOPT OPERATING SETTING
% Set the IPOPT options.
options.ipopt.jac_c_constant = 'yes';
options.ipopt.hessian_constant = 'no';
options.ipopt.hessian_approximation = 'limited-memory';
options.ipopt.limited_memory_max_history = 25;
options.ipopt.nlp_scaling_method = 'none';
options.ipopt.mu_strategy = 'monotone';
options.ipopt.tol = 1e-6;
options.ipopt.constr_viol_tol = 1e-8;
options.ipopt.max_iter = 1000;
options.ipopt.linear_solver = 'mumps';
options.ipopt.alpha_for_y = 'full';
options.ipopt.recalc_y = 'yes';
options.ipopt.print_level = 0;
options.ipopt.max_cpu_time = 100;
% The callback functions.
funcs.objective =
@(x) objective(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,x);
funcs.constraints = @constraints;
funcs.gradient =
@(x) gradient(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x);
funcs.jacobian = @(x) jacobian(nelx,nely) ;
funcs.jacobianstructure = @(x) jacobian(nelx,nely) ;
% Run IPOPT.
delete IPOPT_2D_Data.txt
[x info] = ipopt(x0,funcs,options);

```

```

IPOPTtime = toc;
save('IPOPT_2D_Time.mat','IPOPTtime');
%% PLOT DENSITIES
x2 = vec2mat(x',nelx);
xmat = 1-x2;
colormap(gray); imagesc(1-x2); caxis([0 1]); axis equal; axis off; drawnow;
save('IPOPT_2D_Topology.mat','xmat');
end
% -----
function f = objective
(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,x)
warning('off','all');
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
f = c;
Fetime = toc;
diary IPOPT_2D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function g = gradient
(nelx,nely,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x)
warning('off','all');
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
dctemp = dc';
dcvect = dctemp(:);
g = dcvect;
end
% -----
function lincons = constraints (x)
lincons = sum(x);
end
% -----
function J = jacobian (nelx,nely)
J = sparse(ones(1,nelx*nely));
end

```

SNOPT

```
function f = SNOPT_top88_2D(nely,nelx,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nelx = 90;
nely = 30;
volfrac = 0.5;
penal = 3;
rmin = 2.0;
E = 1;
Emin = 1e-9;
nu = 0.3;
n = nelx*nely;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
A = ones(1,n);
% UPPER AND LOWER BOUNDS
b_L = n*volfrac;
b_U = n*volfrac;
x_0 = volfrac*ones(n,1);
x_L = 0.1*ones(n,1);
x_U = 1.0*ones(n,1);
%% PREPARE FINITE ELEMENT ANALYSIS
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -
1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%% DEFINE LOADS AND SUPPORTS
% MBB PROBLEM
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% CANTILEVER PROBLEM
F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [1:2*(nely+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
% BRIDGE PROBLEM
F = sparse((nely+1)*(nelx+1)+2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = [(2*nely)+1,2*(nely+1),2*(nely+1)*(nelx+1)];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
%% PREPARE FILTER
```

```

iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% SNOPT OPERATING SETTING
funf =
@(x,Prob)con1_f(x,nelx,nely,penal,E,Emin,iK,jK,F,U,freedofs,edofMat,KE,Prob);
fung =
@(x,Prob)con1_g(x,nelx,nely,penal,E,Emin,iK,jK,F,U,freedofs,edofMat,KE,H,Hs,Prob);
% The callback functions.
Prob = conAssign(funf,fung, [], [], ...
                x_L, x_U, [], x_0, ...
                [], [], ...
                A, b_L, b_U, [], [], [], [], ...
                [], [], ...
                [], [], [], []);
Prob.Solver.Alg = 1;
Prob.optParam.bTol = 1e-8;
Prob.optParam.MaxIter = 1000;
% Run SNOPT.
delete SNOPT_2D_Data.txt
Result = tomRun('snopt',Prob,5);
res = Result.x_k;
SNOPTtime = toc;
save('SNOPT_2D_Time.mat','SNOPTtime');
%% PLOT DENSITIES
xPhys = vec2mat(res,nelx);
xmat = 1-xPhys;
colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off;
save('SNOPT_2D_Topology.mat','xmat');
end
% -----
function f =
con1_f(x,nelx,nely,penal,E,Emin,iK,jK,F,U,freedofs,edofMat,KE,Prob)
%% INITIALIZE ITERATION
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);

```



```

ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((Emin+xPhys.^penal*(E-Emin)).*ce));
Fetime = toc;
diary SNOPT_2D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
f = c;
end
% -----
function g =
con1_g(x,nelx,nely,penal,E,Emin,iK,jK,F,U,freedofs,edofMat,KE,H,Hs,Prob)
%% INITIALIZE ITERATION
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E-Emin)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc = -penal*(E-Emin)*xPhys.^(penal-1).*ce;
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
dctemp = dc';
g = dctemp(:);
end

```

Appendix B

3D Structural Topology Optimization

OC

```
function OC_top88_3D(nelx,nely,nelz,volfrac,penal,rmin)
tic
nelx = 60;
nely = 20;
nelz = 4;
volfrac = 0.3;
penal = 3.0;
rmin = 1.5;
ft = 1;
% USER-DEFINED LOOP PARAMETERS
maxloop = 1000;      % Maximum number of iterations
tolx = 1e-4;         % Termination criterion
displayflag = 0;     % Display structure flag
% USER-DEFINED MATERIAL PROPERTIES
E0 = 1;              % Young's modulus of solid material
Emin = 1e-9;         % Young's modulus of void-like material
nu = 0.3;            % Poisson's ratio
%% CANTILEVER SMALL SCALE PROBLEM
% top(60,20,4,0.3,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, 0:nelz);           % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1;                        % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz);          % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% BRIDGE PROBLEM
% top(40,20,40,0.2,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx/2, 0, nelz/2);          % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1;                        % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid([0 0 nelx nelx],[0 0 0 0],[0 nelz 0 nelz]); %
Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% CANTILEVER MEDIUM SCALE PROBLEM
% top(60,20,16,0.15,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, nelz/2);           % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1;                        % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz);          % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
```

```

%% PREPARE FINITE ELEMENT ANALYSIS
nele = nelx*nely*nelz;
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(loadof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nely,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
iK = reshape(kron(edofMat,ones(24,1))',24*24*nele,1);
jK = reshape(kron(edofMat,ones(1,24))',24*24*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for i1 = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (i1-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-
k2)^2));
                    end
                end
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
x = repmat(volfrac,[nely,nelx,nelz]);
xPhys = x;
loop = 0;
change = 1;
fileID = fopen('OC_3D_Data.txt','w');
videocut = false;
% START ITERATION
while change > tolx && loop < maxloop
    loop = loop+1;
    % FE-ANALYSIS
    sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
    K = sparse(iK,jK,sK); K = (K+K')/2;

```

```

U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
dv = ones(nely,nelx,nelz);
% FILTERING AND MODIFICATION OF SENSITIVITIES
if ft == 1
    dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
elseif ft == 2
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
end
% OPTIMALITY CRITERIA UPDATE
l1 = 0; l2 = 1e9; move = 0.2;
while (l2-l1) > 1e-8
    lmid = 0.5*(l2+l1);
    xnew = max(0,max(x-move,min(1,min(x+move,(x.*sqrt(-dc./dv/lmid))))));
    if ft == 1
        xPhys = xnew;
    elseif ft == 2
        xPhys(:) = (H*xnew(:))./Hs;
    end
    if sum(xPhys(:)) > volfrac*nele, l1 = lmid; else l2 = lmid; end
end
change = max(abs(xnew(:)-x(:)));
x = xnew;
OCtime(loop) = toc;
% PRINT RESULTS
fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f
ch.:%7.3f\n',loop,c,mean(xPhys(:)),change);
% PLOT DENSITIES
figure(1); clf; display_3D(xPhys); drawnow;
OC_3D_Data(loop) = c;
fprintf(fileID,'%f\n',OC_3D_Data(loop));
end
save('OC_3D_Time.mat','OCtime');
fclose(fileID);
clf; display_3D(xPhys);
save('OC_3D_Topology.mat','xPhys');
end
% == GENERATE ELEMENT STIFFNESS MATRIX ==
function [KE] = lk_H8(nu)
A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;
     -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/144*A'*[1; nu];
K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
      k(2) k(1) k(2) k(4) k(6) k(7);
      k(2) k(2) k(1) k(4) k(7) k(6);
      k(3) k(4) k(4) k(1) k(8) k(8);
      k(5) k(6) k(7) k(8) k(1) k(2);
      k(5) k(7) k(6) k(8) k(2) k(1)];
K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
      k(8) k(9) k(12) k(5) k(3) k(5);
      k(10) k(10) k(13) k(7) k(4) k(6);
      k(6) k(5) k(11) k(9) k(2) k(10);
      k(4) k(3) k(5) k(2) k(9) k(12)];

```

```

    k(11) k(4) k(6) k(12) k(10) k(13)];
K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
    k(7) k(6) k(4) k(10) k(13) k(10);
    k(5) k(5) k(3) k(8) k(12) k(9);
    k(9) k(10) k(2) k(6) k(11) k(5);
    k(12) k(13) k(10) k(11) k(6) k(4);
    k(2) k(12) k(9) k(4) k(5) k(3)];
K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
    k(11) k(14) k(11) k(12) k(9) k(8);
    k(11) k(11) k(14) k(12) k(8) k(9);
    k(13) k(12) k(12) k(14) k(7) k(7);
    k(10) k(9) k(8) k(7) k(14) k(11);
    k(10) k(8) k(9) k(7) k(11) k(14)];
K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
    k(2) k(1) k(8) k(4) k(6) k(11);
    k(8) k(8) k(1) k(5) k(11) k(6);
    k(3) k(4) k(5) k(1) k(8) k(2);
    k(5) k(6) k(11) k(8) k(1) k(8);
    k(4) k(11) k(6) k(2) k(8) k(1)];
K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
    k(11) k(14) k(7) k(12) k(9) k(2);
    k(7) k(7) k(14) k(10) k(2) k(9);
    k(13) k(12) k(10) k(14) k(7) k(11);
    k(10) k(9) k(2) k(7) k(14) k(7);
    k(12) k(2) k(9) k(11) k(7) k(14)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1 K2 K3 K4;
    K2' K5 K6 K3';
    K3' K6 K5' K2';
    K4 K3 K2 K1'];
end
% === DISPLAY 3D TOPOLOGY (ISO-VIEW) ===
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.8) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
hx z+hx; x+hx y-hx z+hx; x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
end

```

MMA

```
function MMA_top88_3D (nelx,nely,nelz,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nelx = 60;
nely = 20;
nelz = 4;
volfrac = 0.3;
penal = 3;
rmin = 1.5;
E = 1;
Emin = 1e-9;
nu = 0.3;
%% MMA SOLVER PARAMETERS
a0 = 1;
a = 0;
c = 10000000;
d = 1;
m = 1;
% Display flag
displayflag = 0;
n = nelx*nely*nelz;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0 = repmat(volfrac,[nely,nelx,nelz]);
x0 = x0(:);
x = [0.8*x0 0.5*x0 x0];
% UPPER AND LOWER BOUNDS
temp = 0.5*(max(x0)-min(x0));
initlow = x0 - temp;
initup = x0 + temp;
low = initlow;
upp = initup;
% Defining the upper and lower solution limit
xmin(1:n,1) = 0.0001;
xmax(1:n,1) = 1;
%% CANTILEVER SMALL SCALE PROBLEM
% top(60,20,4,0.3,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, 0:nelz); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% BRIDGE PROBLEM
% top(40,20,40,0.2,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx/2, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
```

```

[iif,jf,kf] = meshgrid([0 0 nelx nelx],[0 0 0 0],[0 nelz 0 nelz]); %
Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% CANTILEVER MEDIUM SCALE PROBLEM
% top(60,20,16,0.15,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(loaddof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],n,1);
iK = reshape(kron(edofMat,ones(24,1))',24*24*n,1);
jK = reshape(kron(edofMat,ones(1,24))',24*24*n,1);
%% PREPARE FILTER
iH = ones(n*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for il = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (il-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((il-i2)^2+(j1-j2)^2+(k1-
k2)^2));
                    end
                end
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);

```

```

%% INITIALIZE ITERATION
kkttol = 1e-4;
kktnorm = kkttol+10;
iter = 0;
itermax = 1000;
feastol = 1e-8;
MMA_3D_Data = [];
while ((kktnorm > kkttol) & (iter < itermax) || (feas > feastol))
    iter = iter+1;
    %%% Some vectors are updated:
    xval = x(:,2+iter);
    xold1 = x(:,2+iter-1);
    xold2 = x(:,2+iter-2);
    %%% The user should now calculate function values and gradients
    %%% of the objective- and constraint functions at xval.
    %%% The results should be put in f0val, df0dx, fval and dfdx.
    [f0val,df0dx] =
MMAGradient(nely,nelx,nelz,penal,E,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,xv
al);
    fval = sum(xval) - volfrac*n;
    dfdx(1:n) = 1;
    %%% The MMA subproblem is solved at the point xval:
    [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
    mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,fval,dfdx,low,upp,a0,a,c,d);
    %%% The residual vector of the KKT conditions is calculated:
    [residu,kktnorm,residumax,feas] = ...
    kktcheck(m,n,xmma,ymma,zmma,lam,xsi,eta,mu,zet,s, ...
    xmin,xmax,df0dx,fval,dfdx,a0,a,c,d);
    x = [x xmma];
%% SAVE COMPLIANCE DATA
MMAtime(iter) = toc;
MMA_3D_Data = [MMA_3D_Data f0val];
change = max(abs(xval - xold1));
%% PRINT RESULTS
disp([' It.: ' sprintf('%4i',iter) ' Obj.: ' sprintf('%10.4f',f0val) ...
' Vol.: ' sprintf('%6.3f',sum(xval)/(n)) ...
' ch.: ' sprintf('%6.3f',change)])
%% PLOT DENSITIES FOR EACH ITERATION
if displayflag == 1
    counter = 1;
    xPhys = zeros(nely,nelx,nelz);
    for k = 1:nelz
        for i = 1:nelx
            for j = 1:nely
                xPhys(j,i,k) = xval(counter);
                counter = counter + 1;
            end
        end
    end
    clf; display_3D(xPhys)
end
end
save('MMA_3D_Time.mat','MMAtime');
%% PLOT FINAL DENSITIES
if displayflag == 0
    counter = 1;

```



```

        xPhys = zeros(nely,nelx,nelz);
    for k = 1:nelz
        for i = 1:nelx
            for j = 1:nely
                xPhys(j,i,k) = xval(counter);
                counter = counter + 1;
            end
        end
    end
end
display_3D(xPhys)
end
save('MMA_3D_Topology.mat','xPhys');
%% SAVE COMPLIANCE DATA
fileID = fopen('MMA_3D_Data.txt','w');
for i = 1 : iter
    fprintf(fileID,'%f\n',MMA_3D_Data(i));
end
fclose(fileID);
end
% -----
function [c,dcvect] =
MMAGradient(nely,nelx,nelz,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x
)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
ft = 1;
nele = nelx*nely*nelz;
for k = 1:nelz
    for i = 1:nelx
        for j = 1:nely
            xPhys(j,i,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
%% FILTERING AND MODIFICATION OF SENSITIVITIES
if ft == 1
    dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
elseif ft == 2
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
end
dcvect = zeros(nely*nelx*nelz,1);
dcvect = dc(:);
end
% === GENERATE ELEMENT STIFFNESS MATRIX ===
function [KE] = lk_H8(nu)
A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;

```

```

-48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/144*A'*[1; nu];

K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
      k(2) k(1) k(2) k(4) k(6) k(7);
      k(2) k(2) k(1) k(4) k(7) k(6);
      k(3) k(4) k(4) k(1) k(8) k(8);
      k(5) k(6) k(7) k(8) k(1) k(2);
      k(5) k(7) k(6) k(8) k(2) k(1)];
K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
      k(8) k(9) k(12) k(5) k(3) k(5);
      k(10) k(10) k(13) k(7) k(4) k(6);
      k(6) k(5) k(11) k(9) k(2) k(10);
      k(4) k(3) k(5) k(2) k(9) k(12);
      k(11) k(4) k(6) k(12) k(10) k(13)];
K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
      k(7) k(6) k(4) k(10) k(13) k(10);
      k(5) k(5) k(3) k(8) k(12) k(9);
      k(9) k(10) k(2) k(6) k(11) k(5);
      k(12) k(13) k(10) k(11) k(6) k(4);
      k(2) k(12) k(9) k(4) k(5) k(3)];
K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
      k(11) k(14) k(11) k(12) k(9) k(8);
      k(11) k(11) k(14) k(12) k(8) k(9);
      k(13) k(12) k(12) k(14) k(7) k(7);
      k(10) k(9) k(8) k(7) k(14) k(11);
      k(10) k(8) k(9) k(7) k(11) k(14)];
K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
      k(2) k(1) k(8) k(4) k(6) k(11);
      k(8) k(8) k(1) k(5) k(11) k(6);
      k(3) k(4) k(5) k(1) k(8) k(2);
      k(5) k(6) k(11) k(8) k(1) k(8);
      k(4) k(11) k(6) k(2) k(8) k(1)];
K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
      k(11) k(14) k(7) k(12) k(9) k(2);
      k(7) k(7) k(14) k(10) k(2) k(9);
      k(13) k(12) k(10) k(14) k(7) k(11);
      k(10) k(9) k(2) k(7) k(14) k(7);
      k(12) k(2) k(9) k(11) k(7) k(14)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1 K2 K3 K4;
      K2' K5 K6 K3';
      K3' K6 K5' K2';
      K4 K3 K2 K1'];
end
% === DISPLAY 3D TOPOLOGY (ISO-VIEW) ===
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;

```

```

        if (rho(j,i,k) > 0.8) % User-defined display density threshold
            vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx;x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
            vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
            patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
            hold on;
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
end

```

FMINCON

```

function history = fmincon_top88_3D
tic
%% SET UP SHARED VARIABLES WITH OUTFUN
history.x = [];
history.fval = [];
%% MATERIAL PROPERTIES
nelx = 60;
nely = 20;
nelz = 4;
volfrac = 0.3;
penal = 3.0;
rmin = 1.5;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0(1:nely,1:nelx,1:nelz) = volfrac;
xm = x0(:);
eq = ones(1,nele);
totalvolfrac = nele*volfrac;
% UPPER AND LOWER BOUNDS
nele=nelx*nely*nelz;
lb(1:nele) = 0.0;
ub(1:nele) = 1;
%% CANTILEVER SMALL SCALE PROBLEM
% top(60,20,4,0.3,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, 0:nelz); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% BRIDGE PROBLEM
% top(40,20,40,0.2,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx/2, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs

```

```

load dof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid([0 0 nelx nelx],[0 0 0 0],[0 nelz 0 nelz]); %
Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% CANTILEVER MEDIUM SCALE PROBLEM
% top(60,20,16,0.15,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
load dof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(load dof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
iK = reshape(kron(edofMat,ones(24,1))',24*24*nele,1);
jK = reshape(kron(edofMat,ones(1,24))',24*24*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for il = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (il-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((il-i2)^2+(j1-j2)^2+(k1-
k2)^2));
                    end
                end
            end
        end
    end
end
end
end
end
end

```

```

H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% CALLBACK OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
objgrad =
@(x) funcfdf3D(nely,nelx,nelz,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs
,nele,x);
hessfunc =
@(x,lamda)hessinterior(nely,nelx,nelz,penal,E0,Emin,edofMat,KE,iK,jK,freedofs
,U,F,nele,x);
% FMINCON OPERATOR SETTING
options =
optimoptions('fmincon','OutputFcn',@outfun,'SpecifyObjectiveGradient',true,..
.
'TolFun',1e-6,'TolCon',1e-8,'MaxIteration',1000,'MaxFunctionEvaluations',
10000,...
'HessianFcn',hessfunc);
% RUNNING FMINCON
delete FMINCON_3D_Data.txt
[x,fval] = fmincon(objgrad,xm,[],[],eq,totalvolfrac,lb,ub,[],options);
FMINCONtime = toc;
save('FMINCON_3D_Time.mat','FMINCONtime');
% HISTORICAL DATA COMPILATION
function stop = outfun(x,optimValues,state)
    stop = false;
    switch state
        case 'iter'
            history.fval = [history.fval; optimValues.fval];
            history.x = [history.x x];
    end
end
%% CHANGE VECTOR X TO MATRIX
iter = size(history.x);
Rho = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            Rho(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
%% PLOT DENSITIES
figure()
display_3D(Rho)
save('FMINCON_3D_Topology.mat','Rho')
end
% -----
function [c,dcvect] =
funcfdf3D(nely,nelx,nelz,penal,E0,Emin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,nel
e,x)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz

```

```

        xPhys(i,j,k) = x(counter);
        counter = counter + 1;
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).^U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
dcvect = zeros(1,nely*nelx*nelz);
%% FILTERING AND MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            dcvect(counter) = dc(i,j,k);
            counter = counter + 1;
        end
    end
end
end
FEtime = toc;
diary FMINCON_3D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function Hess = hessinterior
(nely,nelx,nelz,penal,E0,Emin,edofMat,KE,iK,jK,freedofs,U,F,nele,x)
xPhys = zeros(nely,nelx,nelz);
m = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(m);
            m = m + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% SECOND DERIVATIVE
ce = reshape(sum((U(edofMat)*KE).^U(edofMat),2),[nely,nelx,nelz]);
ddc = 2*(penal*(E0-Emin)*xPhys.^(penal-1)).^2.*(1/(Emin+xPhys.^penal*(E0-Emin))).*ce;
Hvect = zeros(nele,1); a = zeros(nele,1); m = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            a(m) = m;

```

```

        Hvect(m) = ddc(i,j,k);
        m = m + 1;
    end
end
end
Hess = sparse(a,a,Hvect,nele,nele);
end
% === GENERATE ELEMENT STIFFNESS MATRIX ===
function [KE] = lk_H8(nu)
A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;
     -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/144*A'*[1; nu];
K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
      k(2) k(1) k(2) k(4) k(6) k(7);
      k(2) k(2) k(1) k(4) k(7) k(6);
      k(3) k(4) k(4) k(1) k(8) k(8);
      k(5) k(6) k(7) k(8) k(1) k(2);
      k(5) k(7) k(6) k(8) k(2) k(1)];
K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
      k(8) k(9) k(12) k(5) k(3) k(5);
      k(10) k(10) k(13) k(7) k(4) k(6);
      k(6) k(5) k(11) k(9) k(2) k(10);
      k(4) k(3) k(5) k(2) k(9) k(12);
      k(11) k(4) k(6) k(12) k(10) k(13)];
K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
      k(7) k(6) k(4) k(10) k(13) k(10);
      k(5) k(5) k(3) k(8) k(12) k(9);
      k(9) k(10) k(2) k(6) k(11) k(5);
      k(12) k(13) k(10) k(11) k(6) k(4);
      k(2) k(12) k(9) k(4) k(5) k(3)];
K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
      k(11) k(14) k(11) k(12) k(9) k(8);
      k(11) k(11) k(14) k(12) k(8) k(9);
      k(13) k(12) k(12) k(14) k(7) k(7);
      k(10) k(9) k(8) k(7) k(14) k(11);
      k(10) k(8) k(9) k(7) k(11) k(14)];
K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
      k(2) k(1) k(8) k(4) k(6) k(11);
      k(8) k(8) k(1) k(5) k(11) k(6);
      k(3) k(4) k(5) k(1) k(8) k(2);
      k(5) k(6) k(11) k(8) k(1) k(8);
      k(4) k(11) k(6) k(2) k(8) k(1)];
K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
      k(11) k(14) k(7) k(12) k(9) k(2);
      k(7) k(7) k(14) k(10) k(2) k(9);
      k(13) k(12) k(10) k(14) k(7) k(11);
      k(10) k(9) k(2) k(7) k(14) k(7);
      k(12) k(2) k(9) k(11) k(7) k(14)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1 K2 K3 K4;
      K2' K5 K6 K3';
      K3' K6 K5' K2';
      K4 K3 K2 K1'];
end
% -----
function display_3D(rho)
[nely,nelx,nelz] = size(rho);

```

```

hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.8) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]);
end

```

IPOPT

```

function IPOPT_top88_3D
%% MATERIAL PROPERTIES
tic
nelx = 60;
nely = 20;
nelz = 4;
volfrac = 0.3;
penal = 3;
rmin = 1.5;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
xm(1:nely,1:nelx,1:nelz) = volfrac;
x0 = xm(:);
% UPPER AND LOWER BOUNDS
nele = nely*nelx*nelz;
lbtemp(1:nele) = 0.0;
ubtemp(1:nele) = 1.0;
options.lb = lbtemp; % Lower bounds on x
options.ub = ubtemp; % Upper bounds on x
options.cl = nele*volfrac-0.001; % Lower bounds on constraints.
options.cu = nele*volfrac+0.001; % Upper bounds on constraints.
%% CANTILEVER SMALL SCALE PROBLEM
% top(60,20,4,0.3,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, 0:nelz); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs

```



```

[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% BRIDGE PROBLEM
% top(40,20,40,0.2,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx/2, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid([0 0 nelx nelx],[0 0 0 0],[0 nelz 0 nelz]); %
Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% CANTILEVER MEDIUM SCALE PROBLEM
% top(60,20,16,0.15,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(loaddof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
iK = reshape(kron(edofMat,ones(24,1))',24*24*nele,1);
jK = reshape(kron(edofMat,ones(1,24))',24*24*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for il = 1:nelx
        for jl = 1:nely
            e1 = (k1-1)*nelx*nely + (il-1)*nely+jl;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
                    for j2 = max(jl-(ceil(rmin)-1),1):min(jl+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;

```

```

        sH(k) = max(0, rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-
k2)^2));
    end
end
end
end
end
H = sparse(iH, jH, sH);
Hs = sum(H, 2);
%% IPOPT OPERATING SETTING
% Set the IPOPT options.
options.ipopt.jac_c_constant      = 'yes';
options.ipopt.hessian_constant   = 'no';
options.ipopt.hessian_approximation = 'limited-memory';
options.ipopt.limited_memory_update_type = 'bfgs';
options.ipopt.mu_strategy        = 'monotone';
options.ipopt.tol                 = 1e-6;
options.ipopt.constr_viol_tol     = 1e-8;
options.ipopt.max_iter            = 1000;
options.ipopt.linear_solver       = 'mumps';
options.ipopt.alpha_for_y         = 'primal';
options.ipopt.print_level         = 0;
options.ipopt.max_cpu_time        = 1200;
% The callback functions.
funcs.objective                  =
@(x) objective(nely, nelx, nelz, penal, E0, Emin, edofMat, KE, iK, jK, freedofs, U, F, nele
, x);
funcs.constraints                 = @constraints;
funcs.gradient                   =
@(x) gradient(nely, nelx, nelz, penal, E0, Emin, edofMat, KE, iK, jK, freedofs, U, F, H, Hs,
nele, x);
funcs.jacobian                   = @(x) jacobian(nely, nelx, nelz);
funcs.jacobianstructure          = @(x) jacobian(nely, nelx, nelz);
% Run IPOPT.
delete IPOPT_3D_Data.txt
[x info] = ipopt(x0, funcs, options);
IPOPTtime = toc;
save('IPOPT_3D_Time.mat', 'IPOPTtime');
%% PLOT DENSITIES
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xmat(i, j, k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
clf; display_3D(xmat);
save('IPOPT_3D_Topology.mat', 'xmat');
end
% -----
function f = objective
(nely, nelx, nelz, penal, E0, Emin, edofMat, KE, iK, jK, freedofs, U, F, nele, x)
warning('off', 'all');
xPhys = zeros(nely, nelx, nelz);

```

```

counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
f = c;
FEtime = toc;
diary IPOPT_3D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function g = gradient
(nely,nelx,nelz,penal,E0,Emin,edofMat,KE,iK,jK,freedofs,U,F,H,Hs,nele,x)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
dcvect = zeros(1,nely*nelx*nelz);
%% FILTERING AND MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            dcvect(counter) = dc(i,j,k);
            counter = counter + 1;
        end
    end
end
end
g = dcvect;

```

```

end
% -----
function lincons = constraints (x)
    lincons = sum(x);
end
% -----
function J = jacobian (nely,nelx,nelz)
    J = sparse(ones(1,nelx*nely*nelz));
end
% === GENERATE ELEMENT STIFFNESS MATRIX ===
function [KE] = lk_H8(nu)
A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;
     -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/144*A'*[1; nu];

K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
      k(2) k(1) k(2) k(4) k(6) k(7);
      k(2) k(2) k(1) k(4) k(7) k(6);
      k(3) k(4) k(4) k(1) k(8) k(8);
      k(5) k(6) k(7) k(8) k(1) k(2);
      k(5) k(7) k(6) k(8) k(2) k(1)];
K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
      k(8) k(9) k(12) k(5) k(3) k(5);
      k(10) k(10) k(13) k(7) k(4) k(6);
      k(6) k(5) k(11) k(9) k(2) k(10);
      k(4) k(3) k(5) k(2) k(9) k(12);
      k(11) k(4) k(6) k(12) k(10) k(13)];
K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
      k(7) k(6) k(4) k(10) k(13) k(10);
      k(5) k(5) k(3) k(8) k(12) k(9);
      k(9) k(10) k(2) k(6) k(11) k(5);
      k(12) k(13) k(10) k(11) k(6) k(4);
      k(2) k(12) k(9) k(4) k(5) k(3)];
K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
      k(11) k(14) k(11) k(12) k(9) k(8);
      k(11) k(11) k(14) k(12) k(8) k(9);
      k(13) k(12) k(12) k(14) k(7) k(7);
      k(10) k(9) k(8) k(7) k(14) k(11);
      k(10) k(8) k(9) k(7) k(11) k(14)];
K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
      k(2) k(1) k(8) k(4) k(6) k(11);
      k(8) k(8) k(1) k(5) k(11) k(6);
      k(3) k(4) k(5) k(1) k(8) k(2);
      k(5) k(6) k(11) k(8) k(1) k(8);
      k(4) k(11) k(6) k(2) k(8) k(1)];
K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
      k(11) k(14) k(7) k(12) k(9) k(2);
      k(7) k(7) k(14) k(10) k(2) k(9);
      k(13) k(12) k(10) k(14) k(7) k(11);
      k(10) k(9) k(2) k(7) k(14) k(7);
      k(12) k(2) k(9) k(11) k(7) k(14)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1 K2 K3 K4;
      K2' K5 K6 K3';
      K3' K6 K5' K2';
      K4 K3 K2 K1'];
end

```

```

% -----
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.8) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
hx z+hx; x+hx y-hx z+hx; x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]);
end

```

SNOPT

```

function f = SNOPT_top88_3D(nely,nelx,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nelx = 60;
nely = 20;
nelz = 4;
volfrac = 0.3;
rmin = 1.5;
penal = 3;
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
nele = nelx*nely*nelz;
A = ones(1,nele);
% UPPER AND LOWER BOUNDS
b_L = nele*volfrac;
b_U = nele*volfrac;
x_0 = volfrac*ones(nele,1);
x_L = 0.1*ones(nele,1);
x_U = 1.0*ones(nele,1);
%% CANTILEVER SMALL SCALE PROBLEM
% top(60,20,4,0.3,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, 0:nelz); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs

```

```

% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% BRIDGE PROBLEM
% top(40,20,40,0.2,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx/2, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid([0 0 nelx nelx],[0 0 0 0],[0 nelz 0 nelz]); %
Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% CANTILEVER MEDIUM SCALE PROBLEM
% top(60,20,16,0.15,3.0,1.5)
% USER-DEFINED LOAD DOFs
[il,jl,kl] = meshgrid(nelx, 0, nelz/2); % Coordinates
loadnid = kl*(nelx+1)*(nely+1)+il*(nely+1)+(nely+1-jl); % Node IDs
loaddof = 3*loadnid(:) - 1; % DOFs
% USER-DEFINED SUPPORT FIXED DOFs
[iif,jf,kf] = meshgrid(0,0:nely,0:nelz); % Coordinates
fixednid = kf*(nelx+1)*(nely+1)+iif*(nely+1)+(nely+1-jf); % Node IDs
fixeddof = [3*fixednid(:); 3*fixednid(:)-1; 3*fixednid(:)-2]; % DOFs
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = 3*(nelx+1)*(nely+1)*(nelz+1);
F = sparse(loaddof,1,-1,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(nu);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = 3*nodeids(:)+1;
edofMat = repmat(edofVec,1,24)+ ...
    repmat([0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1 ...
    3*(nely+1)*(nelx+1)+[0 1 2 3*nely + [3 4 5 0 1 2] -3 -2 -1]],nele,1);
iK = reshape(kron(edofMat,ones(24,1))',24*24*nele,1);
jK = reshape(kron(edofMat,ones(1,24))',24*24*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for il = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (il-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;

```

```
jH(k) = e2;
sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-
k2)^2));

end
end
end
end
end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% SNOPT OPERATING SETTING
funf =
@(x,Prob)conl_f(x,nelx,nely,nelz,penal,E0,Emin,iK,jK,F,U,freedofs,edofMat,KE,
Prob,nele);
fung =
@(x,Prob)conl_g(x,nelx,nely,nelz,penal,E0,Emin,iK,jK,F,U,freedofs,edofMat,KE,
H,Hs,Prob,nele);
% The callback functions.
Prob = conAssign(funf,fung,[],[],...
    x_L,x_U,[],x_0,...
    [],[],...
    A,b_L,b_U,[],[],[],[],...
    [],[],...
    [],[],[],[]);
Prob.Solver.Alg = 1;
Prob.optParam.bTol = 1e-8;
Prob.optParam.MaxIter = 1000;
% Run SNOPT.
delete SNOPT_3D_Data.txt
Result = tomRun('snopt',Prob,0);
res = Result.x_k;
SNOPTtime = toc;
save('SNOPT_3D_Time.mat','SNOPTtime');
%% CHANGING VECTOR X TO MATRIX
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xmat(i,j,k) = res(counter);
            counter = counter + 1;
        end
    end
end
end
%% PLOT DENSITIES
clf; display_3D(xmat);
save('SNOPT_3D_Topology.mat','xmat')
end
% -----
function f =
conl_f(x,nelx,nely,nelz,penal,E0,Emin,iK,jK,F,U,freedofs,edofMat,KE,Prob,nele
)
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
```

```

        xPhys(i,j,k) = x(counter);
        counter = counter + 1;
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce)));
f = c;
FEtime = toc;
diary SNOPT_3D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function g =
con1_g(x,nelx,nely,nelz,penal,E0,Emin,iK,jK,F,U,freedofs,edofMat,KE,H,Hs,Prob
,nele)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),24*24*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
dcvect = zeros(1,nely*nelx*nelz);
%% FILTERING AND MODIFICATION OF SENSITIVITIES
dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            dcvect(counter) = dc(i,j,k);
            counter = counter + 1;
        end
    end
end
end
g = dcvect;
end
% == GENERATE ELEMENT STIFFNESS MATRIX ==
function [KE] = lk_H8(nu)

```



```

A = [32 6 -8 6 -6 4 3 -6 -10 3 -3 -3 -4 -8;
     -48 0 0 -24 24 0 0 0 12 -12 0 12 12 12];
k = 1/144*A'*[1; nu];

K1 = [k(1) k(2) k(2) k(3) k(5) k(5);
      k(2) k(1) k(2) k(4) k(6) k(7);
      k(2) k(2) k(1) k(4) k(7) k(6);
      k(3) k(4) k(4) k(1) k(8) k(8);
      k(5) k(6) k(7) k(8) k(1) k(2);
      k(5) k(7) k(6) k(8) k(2) k(1)];
K2 = [k(9) k(8) k(12) k(6) k(4) k(7);
      k(8) k(9) k(12) k(5) k(3) k(5);
      k(10) k(10) k(13) k(7) k(4) k(6);
      k(6) k(5) k(11) k(9) k(2) k(10);
      k(4) k(3) k(5) k(2) k(9) k(12);
      k(11) k(4) k(6) k(12) k(10) k(13)];
K3 = [k(6) k(7) k(4) k(9) k(12) k(8);
      k(7) k(6) k(4) k(10) k(13) k(10);
      k(5) k(5) k(3) k(8) k(12) k(9);
      k(9) k(10) k(2) k(6) k(11) k(5);
      k(12) k(13) k(10) k(11) k(6) k(4);
      k(2) k(12) k(9) k(4) k(5) k(3)];
K4 = [k(14) k(11) k(11) k(13) k(10) k(10);
      k(11) k(14) k(11) k(12) k(9) k(8);
      k(11) k(11) k(14) k(12) k(8) k(9);
      k(13) k(12) k(12) k(14) k(7) k(7);
      k(10) k(9) k(8) k(7) k(14) k(11);
      k(10) k(8) k(9) k(7) k(11) k(14)];
K5 = [k(1) k(2) k(8) k(3) k(5) k(4);
      k(2) k(1) k(8) k(4) k(6) k(11);
      k(8) k(8) k(1) k(5) k(11) k(6);
      k(3) k(4) k(5) k(1) k(8) k(2);
      k(5) k(6) k(11) k(8) k(1) k(8);
      k(4) k(11) k(6) k(2) k(8) k(1)];
K6 = [k(14) k(11) k(7) k(13) k(10) k(12);
      k(11) k(14) k(7) k(12) k(9) k(2);
      k(7) k(7) k(14) k(10) k(2) k(9);
      k(13) k(12) k(10) k(14) k(7) k(11);
      k(10) k(9) k(2) k(7) k(14) k(7);
      k(12) k(2) k(9) k(11) k(7) k(14)];
KE = 1/((nu+1)*(1-2*nu))*...
    [ K1 K2 K3 K4;
      K2' K5 K6 K3';
      K3' K6 K5' K2';
      K4 K3 K2 K1'];
end
% -----
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely

```

```

        y = nely*hy - (j-1)*hy;
        if (rho(j,i,k) > 0.8) % User-defined display density threshold
            vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx;x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
            vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
            patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
            hold on;
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]);
end

```

Appendix C

2D Heat Conduction Topology Optimization

OC

```
function f=HeatOC2D(nelx,nely,volfrac,penal,rmin);
tic
%% MATERIAL PROPERTIES
nely = 40;
nelx = 40;
volfrac = 0.3;
penal = 3;
rmin = 1.4;
ft = 1;
k0 = 1;           % Young's modulus of solid material
kmin = 1e-3;       % Young's modulus of void-like material
%% USER-DEFINED LOOP PARAMETERS
maxloop = 1000;    % Maximum number of iterations
tolx = 1e-4;       % Termination criterion
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = int32(nelx/2-nelx/20:nelx/2+nelx/20); j1 = nely; % Coordinates
fixedid = il*(nely+1)+(nely+1-j1); % Coordinates
fixeddof = reshape(fixedid,[],1);
nele = nelx*nely;
ndof = (nelx+1)*(nely+1);
F = sparse(1:ndof,1,0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk;
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,4)+ repmat([0 nely+1 nely -1],nele,1);
iK = reshape(kron(edofMat,ones(4,1))',4*4*nele,1);
jK = reshape(kron(edofMat,ones(1,4))',4*4*nele,1);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for il = 1:nelx
    for j1 = 1:nely
        e1 = (il-1)*nely+j1;
        for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((il-i2)^2+(j1-j2)^2));
            end
        end
    end
end
```

```

end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
x = repmat(volfrac,[nely,nelx]);
xPhys = x;
loop = 0;
change = 1.;
fileID = fopen('OC_2D_Data.txt','w');
%% START ITERATION
while ((change > tolX) & (loop < maxloop))
    loop = loop + 1;
    %% FE-ANALYSIS
    sK = reshape(KE(:)*(kmin+xPhys(:)'.^penal*(k0-kmin)),16*nelx*nely,1);
    K = sparse(iK,jK,sK); K = (K+K')/2;
    U(freedofs) = K(freedofs,freedofs)\F(freedofs);
    %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
    c = sum(sum((kmin+xPhys.^penal*(k0-kmin)).*ce));
    dc = -penal*(k0-kmin)*xPhys.^(penal-1).*ce;
    dv = ones(nely,nelx);
    %% FILTERING/MODIFICATION OF SENSITIVITIES
    if ft == 1
        dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
    elseif ft == 2
        dc(:) = H*(dc(:))./Hs;
        dv(:) = H*(dv(:))./Hs;
    end
    %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES
    l1 = 0; l2 = 1e9; move = 0.2;
    while (l2-l1) > 1e-8
        lmid = 0.5*(l2+l1);
        xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid))));
        if ft == 1
            xPhys = xnew;
        elseif ft == 2
            xPhys(:) = (H*xnew(:))./Hs;
        end
        if sum(xPhys(:)) > volfrac*nelx*nely, l1 = lmid; else l2 = lmid; end
    end
    change = max(abs(xnew(:)-x(:)));
    x = xnew;
    OCtime(loop) = toc;
    disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
        ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change)])
    %% PLOT DENSITY
    imagesc(x),colormap(flipud(gray)),caxis([0 1]);axis equal; axis off;
    drawnow;
    OC_2D_Data(loop) = c;
    fprintf(fileID,'%f\n',OC_2D_Data(loop));
end
fclose(fileID);
save('OC_2D_Time.mat','OCtime');
save('OC_2D_Topology.mat','x')
end
%% THERMAL CONDUCTIVITY MATRIX

```

```

function [KE]=lk
KE = [ 2/3 -1/6 -1/3 -1/6
       -1/6 2/3 -1/6 -1/3
       -1/3 -1/6 2/3 -1/6
       -1/6 -1/3 -1/6 2/3];
end

```

MMA

```

function f = MMA_HE_2D(nely,nelx,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nely = 40;
nelx = 40;
volfrac = 0.3;
penal = 3;
rmin = 1.4;
k0 = 1;           % Young's modulus of solid material
kmin = 1e-3;      % Young's modulus of void-like material
%% MMA SETTING
m = 1;
nele = nelx*nely;
xmin(1:nele,1) = 0.1;
xmax(1:nele,1) = 1;
a0 = 1;
a = 0;
c = 100;
d = 1;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0(1:nelx,1:nely) = volfrac;
x0 = x0(:);
x = [0.8*x0 0.5*x0 x0];
% UPPER AND LOWER BOUNDS
temp = 0.5*(max(x0)-min(x0));
initlow = x0 - temp;
initup = x0 + temp;
low = initlow;
upp = initup;
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = int32(nelx/2-nelx/20:nelx/2+nelx/20); j1 = nely; % Coordinates
fixedid = il*(nely+1)+(nely+1-j1); % Coordinates
fixeddof = reshape(fixedid,[],1);
nele = nelx*nely;
ndof = (nelx+1)*(nely+1);
F = sparse(1:ndof,1,0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk;
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,4)+ repmat([0 nely+1 nely -1],nele,1);
iK = reshape(kron(edofMat,ones(4,1))',4*4*nele,1);
jK = reshape(kron(edofMat,ones(1,4))',4*4*nele,1);
%% PREPARE FILTER

```

```

iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
kkttol = 1e-4;
kktnorm = kkttol+10;
iter = 0;
itermax = 1000;
feas = 1;
change = 1.;
MMA_2D_Data = [];
while ((kktnorm > kkttol) & (iter < itermax) || (feas > 1e-8))
    iter = iter+1;
    %%% Some vectors are updated:
    xval = x(:,2+iter);
    xold1 = x(:,2+iter-1);
    xold2 = x(:,2+iter-2);
    %%% The user should now calculate function values and gradients
    %%% of the objective- and constraint functions at xval.
    %%% The results should be put in f0val, df0dx, fval and dfdx.
    [f0val,df0dx] =
MMAGradient(xval,nely,nelx,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs);
    fval = sum(xval) - volfrac*nele;
    dfdx(1:nele) = 1;
    %%% The MMA subproblem is solved at the point xval:
    [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
    mmasub(m,nele,iter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,fval,dfdx,low,upp,a0,a,c,d);
    %%% The residual vector of the KKT conditions is calculated:
    [residu,kktnorm,residumax,feas] = ...
    kktcheck(m,nele,xmma,ymma,zmma,lam,xsi,eta,mu,zet,s, ...
    xmin,xmax,df0dx,fval,dfdx,a0,a,c,d);
    x = [x xmma];
    %% Saving data compliance
    MMAtime(iter) = toc;
    MMA_2D_Data = [MMA_2D_Data f0val];
    change = max(abs(xval - xold1));
    %% PRINT RESULTS
    fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',iter,f0val, ...
    mean(xmma(:)),change);

```

```

%% PLOT DENSITIES
x2 = vec2mat(xval',nelx);
xmat = 1-x2;
imagesc(x2),colormap(flipud(gray)),caxis([0 1]);axis equal; axis off;
drawnow;
end
%% SAVE TIME AND TOPOLOGY DATA
save('MMA_2D_Time.mat','MMAtime');
save('MMA_2D_Topology.mat','xmat');
%% SAVE COMPLIANCE DATA MMA
fileID = fopen('MMA_2D_Data.txt','w');
for i = 1 : iter
    fprintf(fileID,'%f\n',MMA_2D_Data(i));
end
fclose(fileID);
end
% -----
function [c,dcvect] =
MMAGradient(x,nely,nelx,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs);
warning('off','all');
% Optimization Parameters
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+xPhys(:)'.^penal*(k0-kmin)),16*nely*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((kmin+xPhys.^penal*(k0-kmin)).*ce));
dc = -penal*(k0-kmin)*xPhys.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
% Change the matrix x to vector
dcvect = [];
for i = 1:nely
    for j = 1:nelx
        dcvect = [dcvect;dc(i,j)];
    end
end
end
%% THERMAL CONDUCTIVITY MATRIX
function [KE]=lk
KE = [ 2/3 -1/6 -1/3 -1/6
       -1/6 2/3 -1/6 -1/3
       -1/3 -1/6 2/3 -1/6
       -1/6 -1/3 -1/6 2/3];
end

```

FMINCON

```

function history = fmincon_HE_2D
tic
%% SET UP SHARED VARIABLES WITH OUTFUN
history.x = [];
history.fval = [];
%% MATERIAL PROPERTIES

```

```

nely = 40;
nelx = 40;
volfrac = 0.3;
penal = 3;
rmin = 1.4;
k0 = 1; % Young's modulus of solid material
kmin = 1e-3; % Young's modulus of void-like material
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
x0(1:nelx,1:nely) = volfrac;
xm = x0(:);
eq = ones(1,nelx*nely);
totalvolfrac = nely*nelx*volfrac;
% UPPER AND LOWER BOUNDS
nele = nelx*nely;
lb(1:nele) = 0.0;
ub(1:nele) = 1.0;
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = int32(nelx/2-nelx/20:nelx/2+nelx/20); j1 = nely; % Coordinates
fixedid = il*(nely+1)+(nely+1-j1); % Coordinates
fixeddof = reshape(fixedid,[],1);
nele = nelx*nely;
ndof = (nelx+1)*(nely+1);
F = sparse(1:ndof,1,0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk;
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,4)+ repmat([0 nely+1 nely -1],nele,1);
iK = reshape(kron(edofMat,ones(4,1))',4*4*nele,1);
jK = reshape(kron(edofMat,ones(1,4))',4*4*nele,1);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for il = 1:nelx
    for j1 = 1:nely
        e1 = (il-1)*nely+j1;
        for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((il-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% CALLBACK OBJECTIVE FUNCTION AND GRADIENT OBJECTIVE
objgrad =
@(x) funcfdf2D(nelx,nely,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x);

```



```

options =
optimoptions('fmincon','OutputFcn',@outfun,'SpecifyObjectiveGradient',true,...
.
    'SubproblemAlgorithm','cg','TolFun',1e-6,'TolCon',1e-
8,'MaxIteration',1000,...
    'MaxFunctionEvaluations', 10000,'HessianFcn',hessfunc)
% Run FMINCON.
delete FMINCON_2D_Data.txt
[x,fval] = fmincon(objgrad,xm,[],[],eq,totalvolfrac,lb,ub,[],options);
FMINCONtime = toc;
save('FMINCON_2D_Time.mat','FMINCONtime');
% HISTORICAL COMPLIANCE DATA
function stop = outfun(x,optimValues,state)
    stop = false;
    switch state
        case 'iter'
            history.fval = [history.fval; optimValues.fval];
            history.x = [history.x x];
    end
end
%% PLOT DENSITIES
iter = size(history.x);
for i = 1:iter(2)-1
    x2 = vec2mat(history.x(:,i)',nelx);
    xmat = 1-x2;
    imagesc(x2),colormap(flipud(gray)),caxis([0 1]);axis equal; axis off;
drawnow;
end
save('FMINCON_2D_Topology.mat','xmat');
end
% -----
function [c,dcvect] =
funcfdf2D(nelx,nely,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x)
warning('off','all');
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+xPhys(:)'.^penal*(k0-kmin)),16*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
c = sum(sum((kmin+xPhys.^penal*(k0-kmin)).*ce));
dc = -penal*(k0-kmin)*xPhys.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
dctrans = dc';
dcvect = dctrans(:);
%% SAVING COMPLIANCE DATA AND CPU TIME
FETime = toc;
diary FMINCON_2D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
%% THERMAL CONDUCTIVITY MATRIX
function [KE]=lk
KE = [ 2/3 -1/6 -1/3 -1/6

```

```

-1/6 2/3 -1/6 -1/3
-1/3 -1/6 2/3 -1/6
-1/6 -1/3 -1/6 2/3];
end

```

IPOPT

```

function IPOPT_HE_2D
tic
%% MATERIAL PROPERTIES
nely = 40;
nelx = 40;
volfrac = 0.3;
penal = 3;
rmin = 1.4;
k0 = 1;           % Young's modulus of solid material
kmin = 1e-3;      % Young's modulus of void-like material
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
xm(1:nelx,1:nely) = volfrac;
x0 = xm(:);
% UPPER AND LOWER BOUNDS
nele = nely*nelx;
lbtemp(1:nele) = 0.0;
ubtemp(1:nele) = 1.0;
options.lb = lbtemp;           % Lower bounds on x
options.ub = ubtemp;           % Upper bounds on x
options.cl = nele*volfrac-0.001; % Lower bounds on constraints.
options.cu = nele*volfrac+0.001; % Upper bounds on constraints.
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = int32(nelx/2-nelx/20:nelx/2+nelx/20); j1 = nely; % Coordinates
fixedid = il*(nely+1)+(nely+1-j1); % Coordinates
fixeddof = reshape(fixedid,[],1);
nele = nelx*nely;
ndof = (nelx+1)*(nely+1);
F = sparse(1:ndof,1,0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk;
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,4)+ repmat([0 nely+1 nely -1],nele,1);
iK = reshape(kron(edofMat,ones(4,1))',4*4*nele,1);
jK = reshape(kron(edofMat,ones(1,4))',4*4*nele,1);
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;

```

```

        k = k+1;
        iH(k) = e1;
        jH(k) = e2;
        sH(k) = max(0, rmin-sqrt((i1-i2)^2+(j1-j2)^2));
    end
end
end
end
H = sparse(iH, jH, sH);
Hs = sum(H, 2);
%% IPOPT OPERATING SETTING
% Set the IPOPT options.
options.ipopt.jac_c_constant = 'yes';
options.ipopt.hessian_constant = 'no';
options.ipopt.hessian_approximation = 'limited-memory';
options.ipopt.limited_memory_max_history = 25;
options.ipopt.nlp_scaling_method = 'none';
options.ipopt.mu_strategy = 'monotone';
options.ipopt.tol = 1e-6;
options.ipopt.constr_viol_tol = 1e-8;
options.ipopt.max_iter = 1000;
options.ipopt.linear_solver = 'mumps';
options.ipopt.alpha_for_y = 'full';
options.ipopt.recalc_y = 'yes';
options.ipopt.print_level = 0;
options.ipopt.max_cpu_time = 100;
% The callback functions.
funcs.objective =
@ (x) objective(nelx, nely, penal, k0, kmin, U, edofMat, KE, iK, jK, freedofs, F, x);
funcs.constraints = @constraints;
funcs.gradient =
@ (x) gradient(nelx, nely, penal, k0, kmin, U, edofMat, KE, iK, jK, freedofs, F, H, Hs, x);
funcs.jacobian = @ (x) jacobian(nelx, nely);
funcs.jacobianstructure = @ (x) jacobian(nelx, nely);
% Run IPOPT.
delete IPOPT_2D_Data.txt
[x info] = ipopt(x0, funcs, options);
IPOPTtime = toc;
save('IPOPT_2D_Time.mat', 'IPOPTtime');
%% PLOT DENSITIES
x2 = vec2mat(x', nelx);
xmat = 1-x2;
imagesc(x2), colormap(flipud(gray)), caxis([0 1]); axis equal; axis off;
drawnow;
save('IPOPT_2D_Topology.mat', 'xmat');
end
% -----
function f = objective
(nelx, nely, penal, k0, kmin, U, edofMat, KE, iK, jK, freedofs, F, x)
warning('off', 'all');
xPhys = vec2mat(x, nelx);
%% FE-ANALYSIS
sK = reshape(KE(:) * (kmin+xPhys(:)'.^penal*(k0-kmin)), 16*nelx*nely, 1);
K = sparse(iK, jK, sK); K = (K+K')/2;
U(freedofs) = K(freedofs, freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat), 2), nely, nelx);

```

```

c = sum(sum((kmin+xPhys.^penal*(k0-kmin)).*ce));
f = c;
diary IPOPT_2D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function g = gradient
(nelx,nely,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x)
warning('off','all');
xPhys = vec2mat(x,nelx);
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+xPhys(:)'.^penal*(k0-kmin)),16*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
dc = -penal*(k0-kmin)*xPhys.^(penal-1).*ce;
%% FILTERING/MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
dctemp = dc';
dcvect = dctemp(:);
g = dcvect;
end
% -----
function lincons = constraints (x)
lincons = sum(x);
end
% -----
function J = jacobian (nelx,nely)
J = sparse(ones(1,nelx*nely));
end
%% THERMAL CONDUCTIVITY MATRIX
function [KE]=lk
KE = [ 2/3 -1/6 -1/3 -1/6
       -1/6 2/3 -1/6 -1/3
       -1/3 -1/6 2/3 -1/6
       -1/6 -1/3 -1/6 2/3];
end

```

Appendix D

3D Heat Conduction Topology Optimization

OC

```
function OC_HE_3D(nelx,nely,nelz,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nelx = 40;
nely = 40;
nelz = 5;
volfrac = 0.3;
penal = 3.0;
rmin = 1.4;
ft = 1;
k0 = 1; % Young's modulus of solid material
kmin = 1e-3; % Young's modulus of void-like material
%% USER-DEFINED LOOP PARAMETERS
maxloop = 1000; % Maximum number of iterations
tolx = 1e-4; % Termination criterion
displayflag = 1; % Display structure flag
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = nelx/2-nelx/20:nelx/2+nelx/20; jl = nely; kl = 0:nelz; % Coordinates
fixedxy = il*(nely+1)+(nely+1-jl); % Coordinates
fixednid =
repmat(fixedxy',size(kl))+repmat(kl*(nelx+1)*(nely+1),size(fixedxy,2),1);
fixeddof = reshape(fixednid,[],1);
%% PREPARE FINITE ELEMENT ANALYSIS
nele = nelx*nely*nelz;
ndof = (nelx+1)*(nely+1)*(nelz+1);
F = sparse(1:ndof,1,-0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(k0);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,8)+ ...
repmat([0 nely + [1 0] -1 ...
(nely+1)*(nelx+1)+[0 nely + [1 0] -1]],nele,1);
iK = reshape(kron(edofMat,ones(8,1))',8*8*nele,1);
jK = reshape(kron(edofMat,ones(1,8))',8*8*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for kl = 1:nelz
    for il = 1:nelx
        for jl = 1:nely
            el = (kl-1)*nelx*nely + (il-1)*nely+jl;
```

```

        for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
            for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
                for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)

                    e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                    k = k+1;
                    iH(k) = e1;
                    jH(k) = e2;
                    sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-k2)^2));

                end
            end
        end
    end
end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
x = repmat(volfrac,[nely,nelx,nelz]);
xPhys = x;
loop = 0;
change = 1;
fileID = fopen('OC_3D_Data.txt','w');
% START ITERATION
while change > tolx && loop < maxloop
    loop = loop+1;
    %% FE-ANALYSIS
    sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
    K = sparse(iK,jK,sK); K = (K+K')/2;
    U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
    %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
    c = sum(sum(sum((kmin+(1-kmin)*xPhys.^penal).*ce)));
    dc = -penal*(1-kmin)*xPhys.^(penal-1).*ce;
    dv = ones(nely,nelx,nelz);
    %% FILTERING AND MODIFICATION OF SENSITIVITIES
    if ft == 1
        dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
    elseif ft == 2
        dc(:) = H*(dc(:))./Hs;
        dv(:) = H*(dv(:))./Hs;
    end
    % OPTIMALITY CRITERIA UPDATE
    l1 = 0; l2 = 1e9; move = 0.2;
    while (l2-l1) > 1e-8
        lmid = 0.5*(l2+l1);
        xnew = max(0,max(x-move,min(1,min(x+move,(x.*sqrt(-dc./dv/lmid))))));
        if ft == 1
            xPhys = xnew;
        elseif ft == 2
            xPhys(:) = (H*xnew(:))./Hs;
        end
        if sum(xPhys(:)) > volfrac*nele, l1 = lmid; else l2 = lmid; end
    end
    change = max(abs(xnew(:)-x(:)));
    x = xnew;
    OCtime(loop) = toc;
end

```

```

        % PRINT RESULTS
        fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f
ch.:%7.3f\n',loop,c,mean(xPhys(:)),change);
        OC_3D_Data(loop) = c;
        fprintf(fileID,'%f\n',OC_3D_Data(loop));
    end
    fclose(fileID);
    clf; display_3D(xPhys);
    save('OC_3D_Time.mat','Octime');
    save('OC_3D_Topology.mat','xPhys');
end
% === GENERATE THERMAL CONDUCTIVITY MATRIX ===
function [KE] = lk_H8(k)
A1 = 4*eye(2); A2 = -eye(2);
A3 = fliplr(A2); A4 = -ones(2);
KE1 = [A1 A2; A2 A1];
KE2 = [A3 A4; A4 A3];
KE = 1/12*k*[KE1 KE2; KE2 KE1];
end
% === DISPLAY 3D TOPOLOGY (ISO-VIEW) ===
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.5) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx;x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
end

```

MMA

```

function MMA_top88_3D (nelx,nely,nelz,volfrac,penal,rmin)
tic
%% MATERIAL PROPERTIES
nelx = 40;
nely = 40;
nelz = 5;
volfrac = 0.3;
penal = 3.0;
rmin = 1.4;

```

```

k0 = 1;
kmin = 1e-3;
%% MMA SETTING
a0 = 1;
a = 0;
c = 10000000;
d = 1;
m = 1;
displayflag = 1;
%% SOLUTION INITIALIZATION
% VOLUME FRACTION
nele = nelx*nely*nelz;
x0 = repmat(volfrac, [nely, nelx, nelz]);
x0 = x0(:);
x = [0.8*x0 0.5*x0 x0];
% UPPER AND LOWER BOUNDS
temp = 0.5*(max(x0)-min(x0));
initlow = x0 - temp;
initup = x0 + temp;
low = initlow;
upp = initup;
% Defining the upper and lower solution limit
xmin(1:nele,1) = 0.0001;
xmax(1:nele,1) = 1;
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = nelx/2-nelx/20:nelx/2+nelx/20; j1 = nely; k1 = 0:nelz; % Coordinates
fixedxy = il*(nely+1)+(nely+1-j1); % Coordinates
fixednid =
repmat(fixedxy', size(k1))+repmat(k1*(nelx+1)*(nely+1), size(fixedxy,2),1);
fixeddof = reshape(fixednid, [],1);
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = (nelx+1)*(nely+1)*(nelz+1);
F = sparse(1:ndof,1,-0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(k0);
nodegrd = reshape(1:(nely+1)*(nelx+1), nely+1, nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1), nely*nely, 1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids, size(nodeidz))+repmat(nodeidz, size(nodeids));
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,8)+ ...
repmat([0 nely + [1 0] -1 ...
(nely+1)*(nelx+1)+[0 nely + [1 0] -1]],nele,1);
iK = reshape(kron(edofMat,ones(8,1))',8*8*nele,1);
jK = reshape(kron(edofMat,ones(1,8))',8*8*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for i1 = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (i1-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)

```



```

        for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
            e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
            k = k+1;
            iH(k) = e1;
            jH(k) = e2;
            sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-
k2)^2));
        end
    end
end
end
end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% INITIALIZE ITERATION
kkttol = 1e-4;
kktnorm = kkttol+10;
iter = 0;
itermax = 1000;
MMA_3D_Data = [];
while ((kktnorm > kkttol) & (iter < itermax) || (feas > 1e-8))
    iter = iter+1;
    %%% Some vectors are updated:
    xval = x(:,2+iter);
    xold1 = x(:,2+iter-1);
    xold2 = x(:,2+iter-2);
    %%% The user should now calculate function values and gradients
    %%% of the objective- and constraint functions at xval.
    %%% The results should be put in f0val, df0dx, fval and dfdx.
    [f0val,df0dx] =
MMAGradient(nely,nelx,nelz,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x
val);
    fval = sum(xval) - volfrac*nele;
    dfdx(1:nele) = 1;
    %%% The MMA subproblem is solved at the point xval:
    [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
mmasub(m,nele,iter,xval,xmin,xmax,xold1,xold2, ...
f0val,df0dx,fval,dfdx,low,upp,a0,a,c,d);
    %%% The residual vector of the KKT conditions is calculated:
    [residu,kktnorm,residumax,feas] = ...
kktcheck(m,nele,xmma,ymma,zmma,lam,xsi,eta,mu,zet,s, ...
xmin,xmax,df0dx,fval,dfdx,a0,a,c,d);
    x = [x xmma];
    %% SAVE DATA COMPLIANCE
    MMAtime(iter) = toc;
    MMA_3D_Data = [MMA_3D_Data f0val];
    change = max(abs(xval - xold1));
    %% PRINT RESULTS
    disp([' It.: ' sprintf('%4i',iter) ' Obj.: ' sprintf('%10.4f',f0val) ...
' Vol.: ' sprintf('%6.3f',sum(xval)/(nele)) ...
' ch.: ' sprintf('%6.3f',change)])
    %% PLOT DENSITIES
    counter = 1;
    xPhys = zeros(nely,nelx,nelz);
    for k = 1:nelz

```

```

    for i = 1:nelx
        for j = 1:nely
            xPhys(j,i,k) = xval(counter);
            counter = counter + 1;
        end
    end
end
display_3D(xPhys)
%% SAVE TIME AND TOPOLOGY DATA
save('MMA_3D_Time.mat','MMAtime');
save('MMA_3D_Topology.mat','xPhys');
%% SAVE COMPLIANCE DATA
fileID = fopen('MMA_3D_Data.txt','w');
for i = 1 : iter
    fprintf(fileID, '%f\n', MMA_3D_Data(i));
end
fclose(fileID);
HE3DVid = VideoWriter('MMA_HE_3D.avi');
HE3DVid.FrameRate = 20;
open(HE3DVid);
for i=1:length(P)
    frame = P(i) ;
    writeVideo(HE3DVid, frame);
end
close(HE3DVid)
end
% === GENERATE THERMAL CONDUCTIVITY MATRIX ===
function [KE] = lk_H8(k)
A1 = 4*eye(2); A2 = -eye(2);
A3 = fliplr(A2); A4 = -ones(2);
KE1 = [A1 A2; A2 A1];
KE2 = [A3 A4; A4 A3];
KE = 1/12*k*[KE1 KE2; KE2 KE1];
end
% -----
function [c,dcvect] =
MMAGradient(nely,nelx,nelz,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,x
)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
ft = 1;
nele = nelx*nely*nelz;

for k = 1:nelz
    for i = 1:nelx
        for j = 1:nely
            xPhys(j,i,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS

```

```

ce = reshape(sum((U(edofMat)*KE).^U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((kmin+(1-kmin)*xPhys.^penal).*ce)));
dc = -penal*(1-kmin)*xPhys.^(penal-1).*ce;
%% FILTERING AND MODIFICATION OF SENSITIVITIES
if ft == 1
    dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
elseif ft == 2
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
end
dcvect = zeros(nely*nelx*nelz,1);
dcvect = dc(:);
end
% == DISPLAY 3D TOPOLOGY (ISO-VIEW) ==
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.5) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
                hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
                rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]); pause(1e-6);
end

```

FMINCON

```

function history = fmincon_top88_3D
tic
%% SET UP SHARED VARIABLES WITH OUTFUN
history.x = [];
history.fval = [];
%% MATERIAL PROPERTIES
nelx = 40;
nely = 40;
nelz = 5;
volfrac = 0.3;
rmin = 1.4;
penal = 3.0;
k0 = 1;
kmin = 1e-3;
%% SOLUTION INITIALIZATION

```

```

% VOLUME FRACTION
x0(1:nely,1:nelx,1:nelz) = volfrac;
xm = x0(:);
eq = ones(1,nele);
totalvolfrac = nele*volfrac;
% UPPER AND LOWER BOUNDS
nele=nelx*nely*nelz;
    lb(1:nele) = 0.0;
    ub(1:nele) = 1;
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = nelx/2-nelx/20:nelx/2+nelx/20; j1 = nely; k1 = 0:nelz;      % Coordinates
fixedxy = il*(nely+1)+(nely+1-j1);                                % Coordinates
fixednid =
    repmat(fixedxy',size(k1))+repmat(k1*(nelx+1)*(nely+1),size(fixedxy,2),1);
fixeddof = reshape(fixednid,[],1);
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = (nelx+1)*(nely+1)*(nelz+1);
F = sparse(1:ndof,1,-0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(k0);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,8)+ ...
    repmat([0 nely + [1 0] -1 ...
        (nely+1)*(nelx+1)+[0 nely + [1 0] -1]],nele,1);
iK = reshape(kron(edofMat,ones(8,1))',8*8*nele,1);
jK = reshape(kron(edofMat,ones(1,8))',8*8*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for il = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (il-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(il-(ceil(rmin)-1),1):min(il+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((il-i2)^2+(j1-j2)^2+(k1-
k2)^2));
                    end
                end
            end
        end
    end
end
H = sparse(iH,jH,sH);

```

```

Hs = sum(H,2);
%% CALLBACK OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
objgrad =
@(x) funcfdf3D(nely,nelx,nelz,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs
,nele,x);
hessfunc =
@(x,lamda)hessinterior(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs
,U,F,nele,x);
options =
optimoptions('fmincon','OutputFcn',@outfun,'SpecifyObjectiveGradient',true,..
.
    'TolFun',1e-6,'TolCon',1e-8,'MaxIteration',1000,'MaxFunctionEvaluations',
10000,...
    'HessianFcn',hessfunc);
% RUNNING FMINCON
delete FMINCON_3D_Data.txt
[x,fval] = fmincon(objgrad,xm,[],[],eq,totalvolfrac,lb,ub,[],options);
FMINCONtime = toc;
save('FMINCON_3D_Time.mat','FMINCONtime');
% HISTORICAL DATA COMPILATION
function stop = outfun(x,optimValues,state)
    stop = false;
    switch state
        case 'iter'
            history.fval = [history.fval; optimValues.fval];
            history.x = [history.x x];
    end
end
%% CHANGE VECTOR X TO MATRIX
iter = size(history.x);
for m = 1:iter(2)-1
    Rho = zeros(nely,nelx,nelz);
    counter = 1;
    for i = 1:nely
        for j = 1:nelx
            for k = 1:nelz
                Rho(i,j,k) = x(counter)
                counter = counter + 1;
            end
        end
    end
end
%% PLOT DENSITIES
clf; display_3D(Rho)
end
save('FMINCON_3D_Topology.mat','Rho')
end
% === GENERATE THERMAL CONDUCTIVITY MATRIX ===
function [KE] = lk_H8(k)
A1 = 4*eye(2); A2 = -eye(2);
A3 = fliplr(A2); A4 = -ones(2);
KE1 = [A1 A2; A2 A1];
KE2 = [A3 A4; A4 A3];
KE = 1/12*k*[KE1 KE2; KE2 KE1];
end
% -----

```

```

function [c,dcvect] =
funcfdf3D(nely,nelx,nelz,penal,k0,kmin,U,edofMat,KE,iK,jK,freedofs,F,H,Hs,nel
e,x)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((kmin+(1-kmin)*xPhys.^penal).*ce)));
dc = -penal*(1-kmin)*xPhys.^(penal-1).*ce;
dcvect = zeros(1,nely*nelx*nelz);
%% FILTERING AND MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            dcvect(counter) = dc(i,j,k);
            counter = counter + 1;
        end
    end
end
end
FEtime = toc;
diary FMINCON_3D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function Hess = hessinterior
(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs,U,F,nele,x)
xPhys = zeros(nely,nelx,nelz);
m = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(m);
            m = m + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);

```

```

%% SECOND DERIVATIVE
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
ddc = 2*(penal*(1-kmin)*xPhys.^(penal-1)).^2.*(1/(kmin+(1-
kmin)*xPhys.^penal)).*ce;
%% HESSIAN MATRIX
Hvect = zeros(nele,1); a = zeros(nele,1); m = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            a(m) = m;
            Hvect(m) = ddc(i,j,k);
            m = m + 1;
        end
    end
end
Hess = sparse(a,a,Hvect,nele,nele);
end
% -----
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.5) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]);
end

```

IPOPT

```

function IPOPT_top88_3D
tic
%% MATERIAL PROPERTIES
nelx = 40;
nely = 40;
nelz = 5;
volfrac = 0.3;
rmin = 1.4;
penal = 3.0;
k0 = 1;
kmin = 1e-3;

```

```

%% SOLUTION INITIALIZATION
% VOLUME FRACTION
xm(1:nely,1:nelx,1:nelz) = volfrac;
x0 = xm(:);
nele = nely*nelx*nelz;
% UPPER AND LOWER BOUNDS
lbtemp(1:nele) = 0.0;
ubtemp(1:nele) = 1.0;
options.lb = lbtemp; % Lower bounds on x
options.ub = ubtemp; % Upper bounds on x
options.cl = nele*volfrac-0.001; % Lower bounds on constraints.
options.cu = nele*volfrac+0.001; % Upper bounds on constraints.
%% DEFINE HEAT GENERATION AND BOUNDARY CONDITION
il = nelx/2-nelx/20:nelx/2+nelx/20; j1 = nely; k1 = 0:nelz; % Coordinates
fixedxy = il*(nely+1)+(nely+1-j1); % Coordinates
fixednid =
repmat(fixedxy',size(k1))+repmat(k1*(nelx+1)*(nely+1),size(fixedxy,2),1);
fixeddof = reshape(fixednid,[],1);
%% PREPARE FINITE ELEMENT ANALYSIS
ndof = (nelx+1)*(nely+1)*(nelz+1);
F = sparse(1:ndof,1,-0.01,ndof,1);
U = zeros(ndof,1);
freedofs = setdiff(1:ndof,fixeddof);
KE = lk_H8(k0);
nodegrd = reshape(1:(nely+1)*(nelx+1),nely+1,nelx+1);
nodeids = reshape(nodegrd(1:end-1,1:end-1),nely*nelx,1);
nodeidz = 0:(nely+1)*(nelx+1):(nelz-1)*(nely+1)*(nelx+1);
nodeids = repmat(nodeids,size(nodeidz))+repmat(nodeidz,size(nodeids));
edofVec = nodeids(:)+1;
edofMat = repmat(edofVec,1,8)+ ...
repmat([0 nely + [1 0] -1 ...
(nely+1)*(nelx+1)+[0 nely + [1 0] -1]],nele,1);
iK = reshape(kron(edofMat,ones(8,1))',8*8*nele,1);
jK = reshape(kron(edofMat,ones(1,8))',8*8*nele,1);
%% PREPARE FILTER
iH = ones(nele*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for k1 = 1:nelz
    for i1 = 1:nelx
        for j1 = 1:nely
            e1 = (k1-1)*nelx*nely + (i1-1)*nely+j1;
            for k2 = max(k1-(ceil(rmin)-1),1):min(k1+(ceil(rmin)-1),nelz)
                for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
                    for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-
1),nely)
                        e2 = (k2-1)*nelx*nely + (i2-1)*nely+j2;
                        k = k+1;
                        iH(k) = e1;
                        jH(k) = e2;
                        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2+(k1-
k2)^2));
                    end
                end
            end
        end
    end
end

```



```

end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
%% IPOPT OPERATING SETTING
% Set the IPOPT options.
options.ipopt.jac_c_constant      = 'yes';
options.ipopt.hessian_constant   = 'no';
options.ipopt.hessian_approximation = 'limited-memory';
options.ipopt.limited_memory_update_type = 'bfgs';
options.ipopt.mu_strategy        = 'monotone';
options.ipopt.tol                 = 1e-6;
options.ipopt.constr_viol_tol     = 1e-8;
options.ipopt.max_iter            = 1000;
options.ipopt.linear_solver       = 'mumps';
options.ipopt.alpha_for_y         = 'primal';
options.ipopt.print_level         = 0;
options.ipopt.max_cpu_time        = 600;
% The callback functions.
funcs.objective                  =
@(x) objective(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs,U,F,nele,x);
funcs.constraints                 = @constraints;
funcs.gradient                   =
@(x) gradient(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs,U,F,H,Hs,nele,x);
funcs.jacobian                   = @(x) jacobian(nely,nelx,nelz);
funcs.jacobianstructure          = @(x) jacobian(nely,nelx,nelz);
% Run IPOPT.
delete IPOPT_3D_Data.txt
[x info] = ipopt(x0,funcs,options);
IPOPTtime = toc;
save('IPOPT_3D_Time.mat','IPOPTtime');
%% PLOT DENSITIES
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xmat(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
clf; display_3D(xmat);
save('IPOPT_3D_Topology.mat','xmat');
end
% -----
function f = objective
(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs,U,F,nele,x)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end

```

```

        end
    end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
c = sum(sum(sum((kmin+(1-kmin)*xPhys.^penal).*ce)));
f = c;
Fetime = toc;
diary IPOPT_3D_Data.txt
fprintf("%f",c)
fprintf("\n")
diary off
end
% -----
function g = gradient
(nely,nelx,nelz,penal,k0,kmin,edofMat,KE,iK,jK,freedofs,U,F,H,Hs,nele,x)
warning('off','all');
xPhys = zeros(nely,nelx,nelz);
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            xPhys(i,j,k) = x(counter);
            counter = counter + 1;
        end
    end
end
end
%% FE-ANALYSIS
sK = reshape(KE(:)*(kmin+(1-kmin)*xPhys(:)'.^penal),8*8*nele,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs,:) = K(freedofs,freedofs)\F(freedofs,:);
%% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),[nely,nelx,nelz]);
dc = -penal*(1-kmin)*xPhys.^(penal-1).*ce;
dcvect = zeros(1,nely*nelx*nelz);
%% FILTERING AND MODIFICATION OF SENSITIVITIES
dc(:) = H*(xPhys(:).*dc(:))./Hs./max(1e-3,xPhys(:));
counter = 1;
for i = 1:nely
    for j = 1:nelx
        for k = 1:nelz
            dcvect(counter) = dc(i,j,k);
            counter = counter + 1;
        end
    end
end
end
g = dcvect;
end
% -----
function lincons = constraints (x)
    lincons = sum(x);
end
% -----

```

```

function J = jacobian (nely,nelx,nelz)
    J = sparse(ones(1,nelx*nely*nelz));
end
% === GENERATE THERMAL CONDUCTIVITY MATRIX ===
function [KE] = lk_H8(k)
A1 = 4*eye(2); A2 = -eye(2);
A3 = fliplr(A2); A4 = -ones(2);
KE1 = [A1 A2; A2 A1];
KE2 = [A3 A4; A4 A3];
KE = 1/12*k*[KE1 KE2; KE2 KE1];
end
% -----
function display_3D(rho)
[nely,nelx,nelz] = size(rho);
hx = 1; hy = 1; hz = 1; % User-defined unit element size
face = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
set(gcf,'Name','ISO display','NumberTitle','off');
for k = 1:nelz
    z = (k-1)*hz;
    for i = 1:nelx
        x = (i-1)*hx;
        for j = 1:nely
            y = nely*hy - (j-1)*hy;
            if (rho(j,i,k) > 0.5) % User-defined display density threshold
                vert = [x y z; x y-hx z; x+hx y-hx z; x+hx y z; x y z+hx; x y-
hx z+hx; x+hx y-hx z+hx;x+hx y z+hx];
                vert(:,[2 3]) = vert(:,[3 2]); vert(:,2,:) = -vert(:,2,:);
                patch('Faces',face,'Vertices',vert,'FaceColor',[0.2+0.8*(1-
rho(j,i,k)),0.2+0.8*(1-rho(j,i,k)),0.2+0.8*(1-rho(j,i,k))]);
                hold on;
            end
        end
    end
end
axis equal; axis tight; axis off; box on; view([30,30]);
end

```