

# Project 3: Topology Optimization

## 1 Introduction

In this project, you will learn to implement an optimization algorithm for minimizing the compliance of a structure at its equilibrium state with respect to its topology. The tutorial is based on Dr. Sigmund's topology optimization [code](#) and [paper](#). The template code is attached at the end of the tutorial.

## 2 The compliance minimization problem

Topology optimization has been commonly used to design structures and materials with optimal mechanical, thermal, electromagnetic, acoustical, or other properties. The structure under design is segmented into  $n$  finite elements, and a density value  $x_i$  is assigned to each element  $i \in \{1, 2, \dots, n\}$ : A higher density corresponds to a less porous material element and higher Young's modulus. Reducing the density to zero is equivalent to creating a hole in the structure. Thus, the set of densities  $\mathbf{x} = \{x_i\}$  can be used to represent the topology of the structure and is considered as the variables to be optimized. A common topology optimization problem is compliance minimization, where we seek the "stiffest" structure within a certain volume limit to withstand a particular load:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{f} := \mathbf{d}^T \mathbf{K}(\mathbf{x}) \mathbf{d} \\ \text{subject to:} \quad & \mathbf{h} := \mathbf{K}(\mathbf{x}) \mathbf{d} = \mathbf{u}, \\ & \mathbf{g} := V(\mathbf{x}) \leq v, \\ & \mathbf{x} \in [0, 1]. \end{aligned} \tag{1}$$

Here  $V(\mathbf{x})$  is the total volume;  $v$  is an upper bound on volume;  $\mathbf{d} \in \mathbb{R}^{n_d \times 1}$  is the displacement of the structure under the load  $\mathbf{u}$ , where  $n_d$  is the degrees of freedom (DOF) of the system (i.e., the number of x- and y-coordinates of nodes from the finite element model of the structure);  $\mathbf{K}(\mathbf{x})$  is the global stiffness matrix for the structure.  $\mathbf{K}(\mathbf{x})$  is indirectly influenced by the topology  $\mathbf{x}$ , through the element-wise stiffness matrix

$$\mathbf{K}_i = \bar{\mathbf{K}}_e E(x_i), \tag{2}$$

and the local-to-global assembly:

$$\mathbf{K}(\mathbf{x}) = \mathbf{G}[\mathbf{K}_1, \mathbf{K}_2, \dots, \mathbf{K}_n], \tag{3}$$

where the matrix  $\bar{\mathbf{K}}_e$  is predefined according to the finite element type (we use first-order quadrilateral elements throughout this tutorial) and the nodal displacements of the element (we use square elements with unit lengths throughout this tutorial),  $\mathbf{G}$  is an assembly matrix,  $E(x_i)$  is the element-wise Young's modulus defined as a function of the density  $x_i$ :  $E(x_i) :=$

$\Delta E x_i^p + E_{\min}$ , where  $p$  (the penalty parameter) is usually set to 3. This cubic relationship between the topology and the modulus is determined by the material constitutive models, and numerically, it also helps binarize the topologies, i.e., to push the optimal  $\mathbf{x}_i$  to 1 or 0 (why?). The term  $E_{\min}$  is added to provide numerical stability.

### 3 Design sensitivity analysis

This problem has both inequality and equality constraints. However, the inequality ones are only related to the topology  $\mathbf{x}$ , and are either linear ( $V(\mathbf{x}) \leq v$ ) or simple bounds ( $\mathbf{x} \in [0, 1]$ ). We will show that these constraints can be easily handled. The problem thus involves two sets of variables: We can consider  $\mathbf{x}$  as the *decision variables* and  $\mathbf{u}$  as the state variables that are governed by  $\mathbf{x}$  through the equality constraint  $\mathbf{K}(\mathbf{x})\mathbf{d} = \mathbf{u}$ .

The reduced gradient (often called design sensitivity) can be calculated as

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \frac{\partial f}{\partial \mathbf{u}} \left( \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \right)^{-1} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}, \quad (4)$$

which leads to

$$\frac{df}{d\mathbf{x}} = -\mathbf{u}^T \frac{\partial \mathbf{K}}{\partial \mathbf{x}} \mathbf{u}. \quad (5)$$

Recall the relation between  $\mathbf{K}$  and  $\mathbf{x}$ , and notice that

$$\mathbf{u}^T \mathbf{K} \mathbf{u} = \sum_{i=1}^n \mathbf{u}_i^T \mathbf{K}_i \mathbf{u}_i, \quad (6)$$

i.e., the total compliance (strain energy) is the summation of element-wise compliance. We can further simplify the gradient as follows:

$$\begin{aligned} -\mathbf{u}^T \frac{\partial \mathbf{K}}{\partial \mathbf{x}} \mathbf{u} &= -\frac{\partial \mathbf{u}^T \mathbf{K} \mathbf{u}}{\partial \mathbf{x}} \\ &= -\frac{\partial \sum_{i=1}^n \mathbf{u}_i^T \mathbf{K}_i \mathbf{u}_i}{\partial \mathbf{x}} \\ &= [\dots, -\frac{\partial \mathbf{u}_i^T \mathbf{K}_i \mathbf{u}_i}{\partial x_i}, \dots] \\ &= [\dots, -\mathbf{u}_i^T \frac{\partial \mathbf{K}_i}{\partial x_i} \mathbf{u}_i, \dots] \\ &= [\dots, -\mathbf{u}_i^T \frac{\partial \bar{\mathbf{K}}_e \Delta E x_i^3}{\partial x_i} \mathbf{u}_i, \dots] \\ &= [\dots, -3\Delta E x_i^2 \mathbf{u}_i^T \bar{\mathbf{K}}_e \mathbf{u}_i, \dots] \end{aligned} \quad (7)$$

### 4 The algorithm

The pseudo code for compliance minimization is the following:

- Problem setup (see details below)
- Algorithm setup:  $\epsilon = 0.001$  (or any small positive number),  $k = 0$  (counter for the iteration),  $\Delta x = 1000$  (or any number larger than  $\epsilon$ )
- While  $\|\Delta x\| \leq \epsilon$ , Do:
  - Update the stiffness matrix  $\mathbf{K}$  and the displacement (state)  $\mathbf{u}$  (finite element analysis)
  - Calculate element-wise compliance  $\mathbf{u}_i^T \bar{\mathbf{K}}_e \mathbf{u}_i$
  - Calculate partial derivatives

$$\frac{df}{dx_i} = -3\Delta E x_i^2 \mathbf{u}_i^T \bar{\mathbf{K}}_e \mathbf{u}_i$$

- The gradient with respect to  $g$  is a constant vector  $[1, \dots, 1]^T$
- Apply filter to  $\frac{df}{d\mathbf{x}}$  (See discussion later)
- Update  $\mathbf{x}$ :

$$\mathbf{x}'_{k+1} = \mathbf{x}_k - \alpha_k \left( \frac{df}{d\mathbf{x}} + \mu \mathbf{1} \right),$$

where  $\mu \geq 0$  is determined in the next step. To ensure that the gradient descent is successful, we will either set  $\alpha_k$  to a small value, or truncate  $\Delta x = -(\frac{df}{d\mathbf{x}} + \mu \mathbf{1})$  within a range (conceptually similar to the idea of trust region).

- Move  $\mathbf{x}'_{k+1}$  back to the feasible domain: If  $\mathbf{1}^T \mathbf{x}_k < v$  and  $-\mathbf{1}^T \frac{df}{d\mathbf{x}} < 0$ , then  $\mathbf{x}'_{k+1}$  satisfies  $g$  with  $\mu = 0$ . If  $\mathbf{x}'_{k+1}$  does not satisfy  $g$ , we will increase  $\mu$  using bisection, i.e., search in  $[0, \mu_{max}]$  where  $\mu_{max}$  is a large positive number. Also, we will truncate  $\mathbf{x}'_{k+1}$  between 0 and 1.
- Update  $\|\Delta x\|$ ,  $k = k + 1$

## 5 Implementation details

Some details of the template code are explained as follows:

- **The numbering rule for elements and nodes:** The template code assumes a rectangular design domain, where each element is modeled as a unit square. The numbering of elements and nodes are explained in Fig. 1.
- **Input parameters:** Inputs to the program are (1) the number of elements in the x and y directions (**nelx** and **nely**), (2) the volume fraction (**volfrac**, this is the number between 0 and 1 that specifies the ratio between the volume of the target topology and the maximum volume  $nelx \times nely$ ), (3) the penalty parameter of the Young's Modulus model (**penal**, usually = 3), and (4) the filter radius (**rmin**, usually = 3).

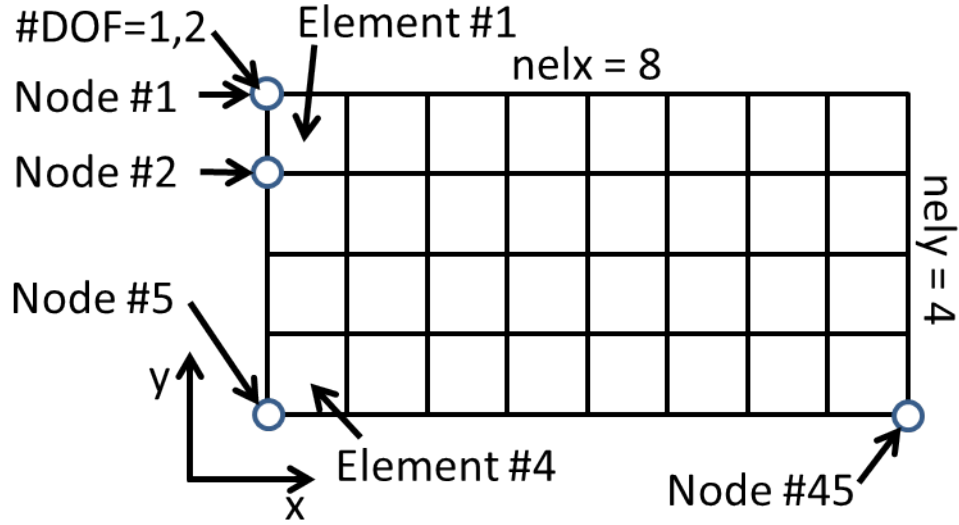


Figure 1: The numbering of elements and nodes

- **Material properties:**

```
%% MATERIAL PROPERTIES
E0 = 1;
Emin = 1e-9;
nu = 0.3;
```

Set the Young's Modulus (“E0”), and the Poisson's ratio (“nu”). Leave Emin as a small number.

- **Define loadings**

```
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
```

This line specifies the loading. Here F is a sparse column vector with  $2(nely+1)(nelx+1)$  elements.  $(2,1,-1,\dots)$  specifies that there is a force of  $-1$  at the second row and first column of the vector. According to the numbering convention of this code, this is to say that in the y direction of the first node, there is a downward force of magnitude 1.

- **Define boundary conditions** (Fixed DOFs):

```
fixeddofs = union([1:2:2*(nely+1)], [2*(nelx+1)*(nely+1)]);
```

This line specifies the nodes with fixed DOFs.  $[1:2:2*(nely+1)]$  are x directions of all nodes to the left side of the structure, and  $2*(nelx+1)*(nely+1)$  is the y direction of the last node (right bottom corner). See Fig. 2:

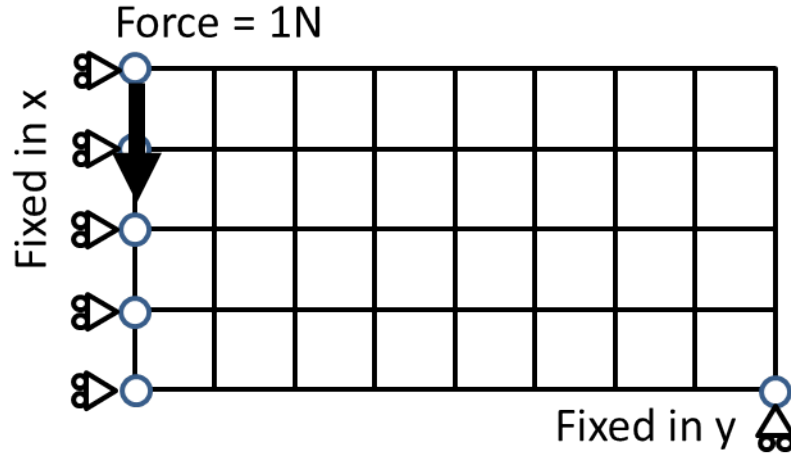


Figure 2: Define boundary conditions

- **Filtering of sensitivity (gradient)** Pure gradient descent may result in a topology with checkerboard patterns. While mathematically sound, such a solution can be infeasible from a manufacturing perspective or too expensive to realize (e.g., through additive manufacturing of porous structures). Therefore, a smoothed solution is often more preferred.

In the template we prepare a Gaussian filter, through the following code:

```
%% 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);
```

The design sensitivity can then be filtered by

```
dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
```

## 6 The template code

```
%%% Modified by Max Yi Ren (ASU) %%%%%%%%%%%%%%%

%%% AN 88 LINE TOPOLOGY OPTIMIZATION CODE Nov, 2010 %%%
function top88(nelx,nely,volfrac,penal,rmin,ft)

%% MATERIAL PROPERTIES
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 (HALF MBB-BEAM)
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);

%% 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;

%% START ITERATION
while change > 0.01
    loop = loop + 1;

    %% FE-ANALYSIS
    K = ?;
    U(freedofs) = ?;

    %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
    ce = ?; % element-wise strain energy
    c = ?; % total strain energy
    dc = ?; % design sensitivity
    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
end

```

```

%% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES
while ?
    ???
end
change = max(abs(xnew(:)-x(:)));
x = xnew;

%% PRINT RESULTS
fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, ...
    mean(xPhys(:)),change);

%% PLOT DENSITIES
colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off; drawnow;
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This Matlab code was written by E. Andreassen, A. Clausen, M. Schevenels,
% B. S. Lazarov and O. Sigmund, Department of Solid Mechanics,
% Technical University of Denmark,
% DK-2800 Lyngby, Denmark.
% Please sent your comments to: sigmund@fam.dtu.dk
%
% The code is intended for educational purposes and theoretical details
% are discussed in the paper
% "Efficient topology optimization in MATLAB using 88 lines of code,
% E. Andreassen, A. Clausen, M. Schevenels,
% B. S. Lazarov and O. Sigmund, Struct Multidisc Optim, 2010
% This version is based on earlier 99-line code
% by Ole Sigmund (2001), Structural and Multidisciplinary Optimization,
% Vol 21, pp. 120--127.
%
% The code as well as a postscript version of the paper can be
% downloaded from the web-site: http://www.topopt.dtu.dk
%
% Disclaimer:
% The authors reserves all rights but do not guaranty that the code is
% free from errors. Furthermore, we shall not be liable in any event
% caused by the use of the program.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



## 7 Local density constraint

Local density constraint has been discussed in [Wu. et al.](#) This formulation of topology optimization derives more porous-like structures that are robust against local defects.