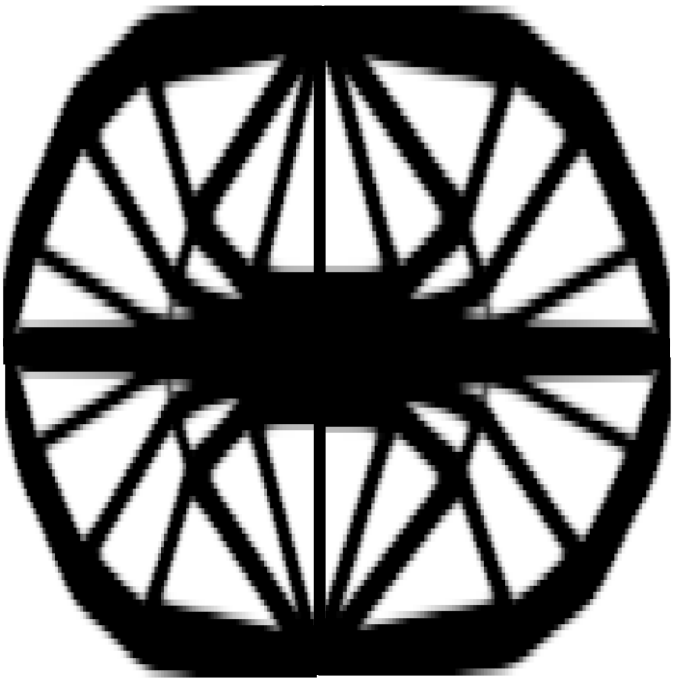


A Topology Optimization output using Finite Elements Implemented on a Structural problem

TOPOLOGY OPTIMIZATION USING FINITE ELEMENT METHODS BASED ON OPTIMALITY CRITERIA

-Manthan N. Dhisale (193109014)

Mentored by Mr. Akash Kapase

Thermal Conduction topology optimization results on 2D plate

Content

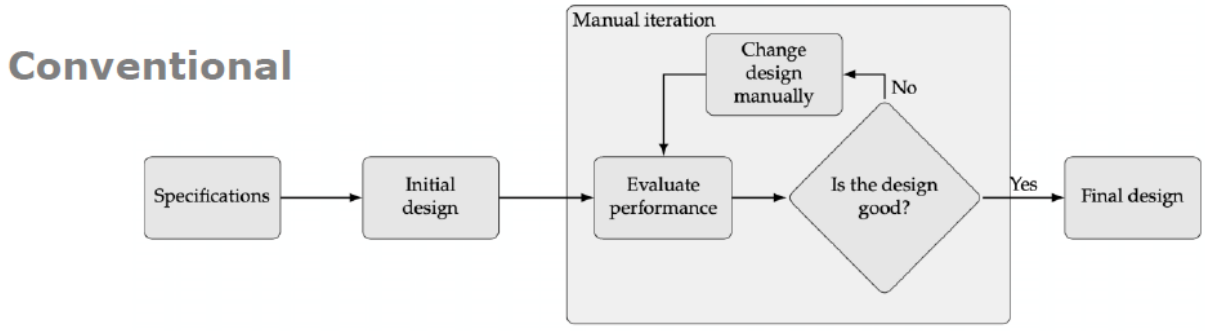
Introduction

Optimization can be referred in many ways. People optimize like investors optimize for getting maximum return on investments, manufacturers optimize to seek maximum efficiency in their production process while the engineers optimize their designs by adjusting various parameter to seek maximum performance from the system.

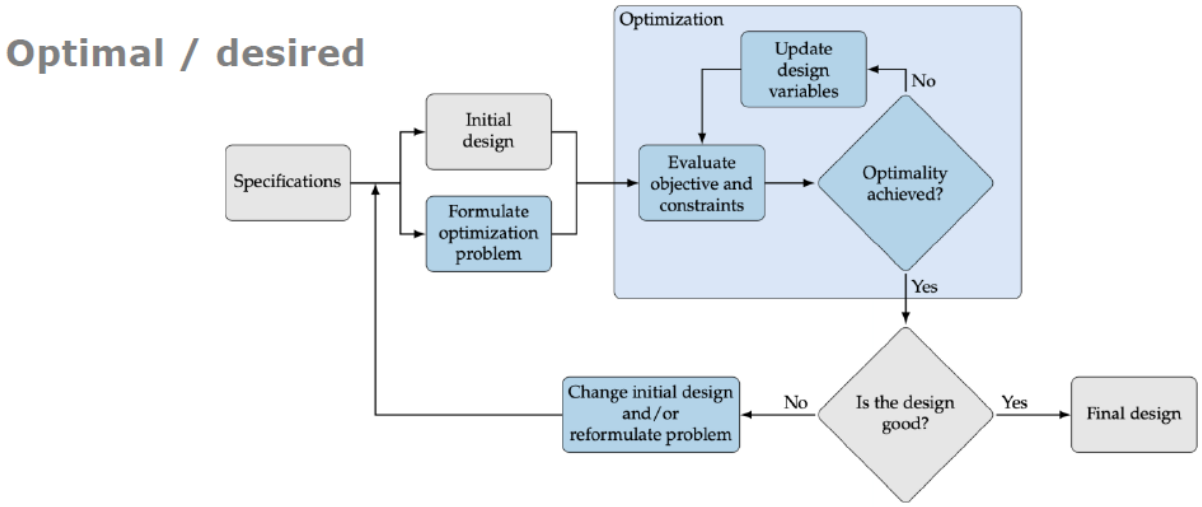
Even it's fascinating to note that Mother Nature optimizes. It optimizes various systems for minimum potential energy. Even during chemical reactions molecules react in such a way that the distance between the newly formed atomic interactions (bonds) has minimum total energy of reaction.

Optimization is an important tool for analysis of physical systems. Our goal is to find values of the variables that optimize (maximize or minimize) the objective of which the variables are either restricted or constrained by some ways.

A Conventional design process without involvement of optimization techniques can be understood as follows. The major pitfall of this process is that, the design process has to go over a number of iterations, until the desired performance or cost target is achieved.



On the contrary, with involvement of optimization, the conventional process can be simply modified as follows:



Mathematical Preliminaries

Mathematical Formulation:

Any optimization problem can be mathematically modelled in the following fashion:

$$\text{x}\epsilon R^n \; where \; x \; are \; design \; variables$$

$$\text{p}\epsilon R^m \; where \; p \; are \; parameters$$

$$\text{J}: (R^n, R^m) \rightarrow R^l \; where \; J \; are \; objective \; functions$$

$$\text{g}: (R^n, R^m, R^l) \rightarrow R^k \; where \; g \; are \; inequality \; constraints$$

$$\text{h}: (R^n, R^m, R^l) \rightarrow R^j \; where \; h \; are \; equality \; constraints$$

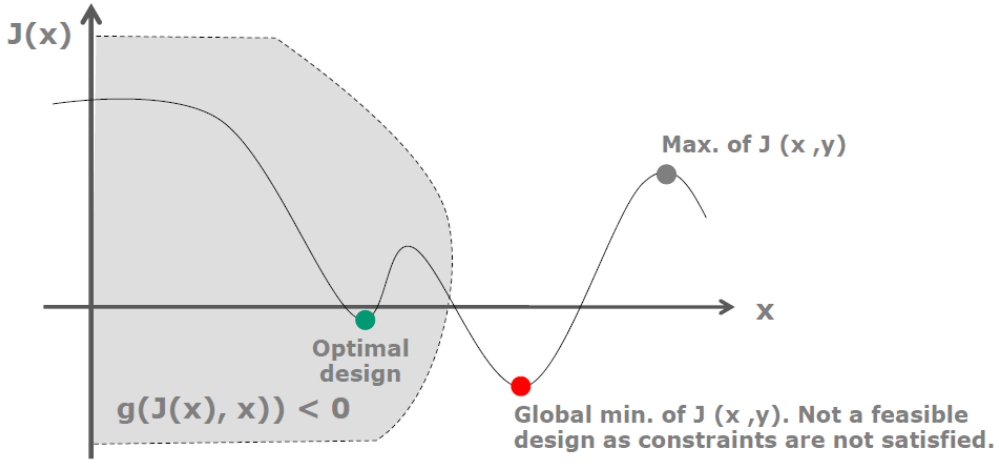
$$\text{find } x^* | \; \text{J}(x^*) \leq \text{J}(x^*)$$

$$\text{g}(x^*) \leq 0$$

$$\text{h}(x^*) = 0$$

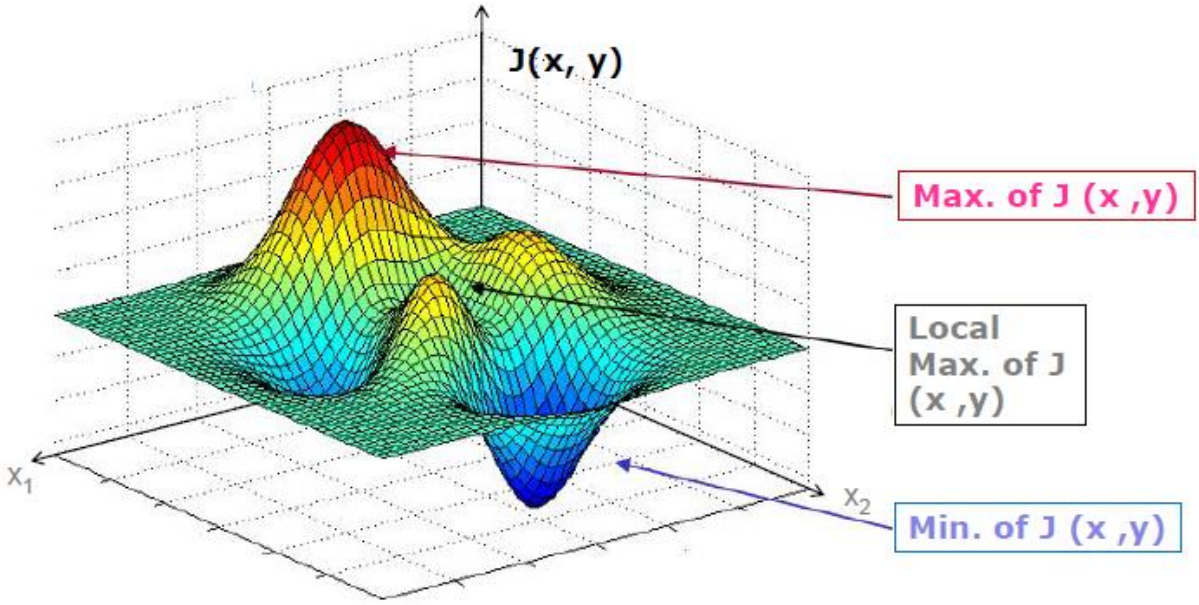
- Constraints arise because of finiteness of variables such as space, time, resources and limitations on available technology. They are the boundaries of design variables.
- Design variables cannot violate constraints while evaluating objective function
- In several situations, optimal designs lie at the intersection of several constraints
- Constraints can be of three types:
    1. Bounds $x_{iLB} \leq x_i \leq x_{iUB} \; where \; i = 1,2,3, \ldots \ldots n$
    2. Inequality $g_j(\text{x}) \leq 0 \; where \; j = 1,2,3, \ldots \ldots m1$
    3. Equality $h_k \leq 0 \; where \; k = 1,2,3, \ldots \ldots m2$

Design Optimization for 1D problems:

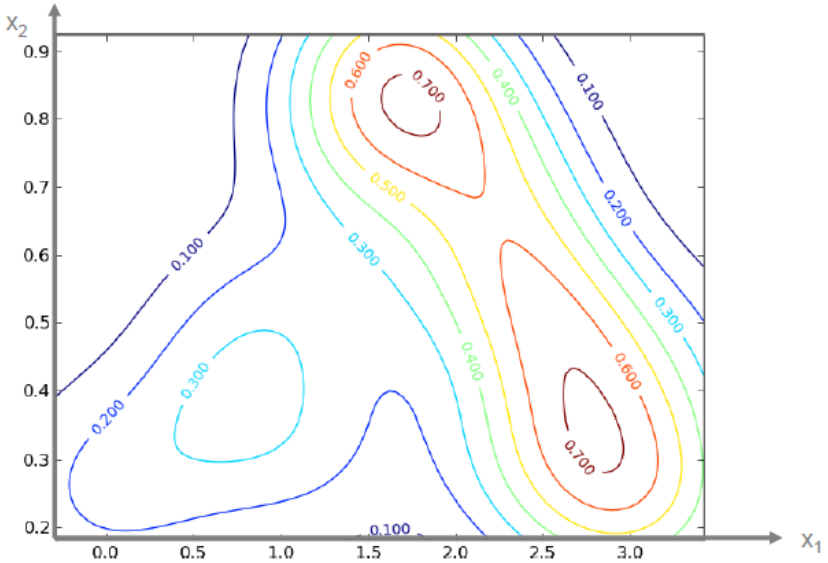


J(x) is the cost function that is to be minimized
X is the design variable
g(J(x), x) are inequality constraints
It is customary to minimize the cost function. If a quantity is to be maximized, the cost function is the negative of that quantity.

Design Optimization for 2D problems:



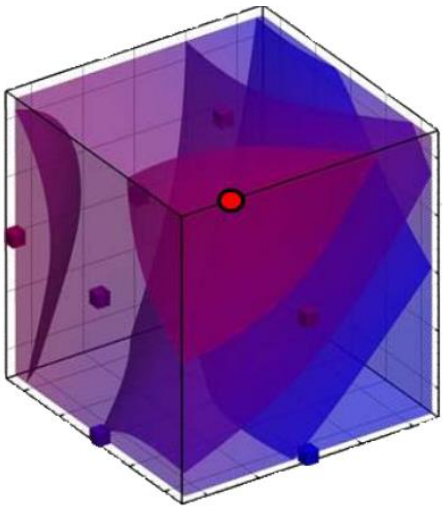
Here J(x) is the objective function to be optimised and x1, x2 are design variables



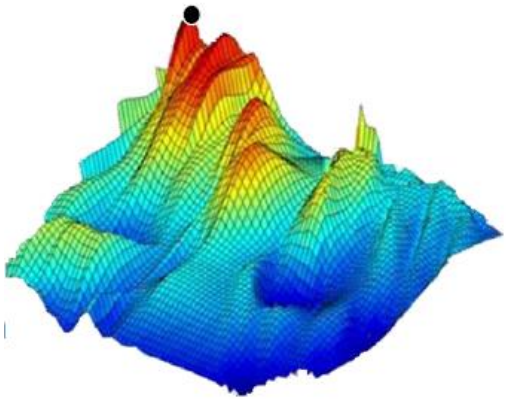
On each contour J is constant. The crowding of contours indicate rapid change in objective function value

Design Optimization for 3D and beyond problems:

Colours represent magnitude of objective function. Graphical methods can still be employed to identify and visualize maxima or minima. Graphical methods can still be used beyond objective functions with more than 3 variables, however it starts receiving a limitation beyond 4 or 5 design variables. Below plots show the objective function value in 3D and 4D space

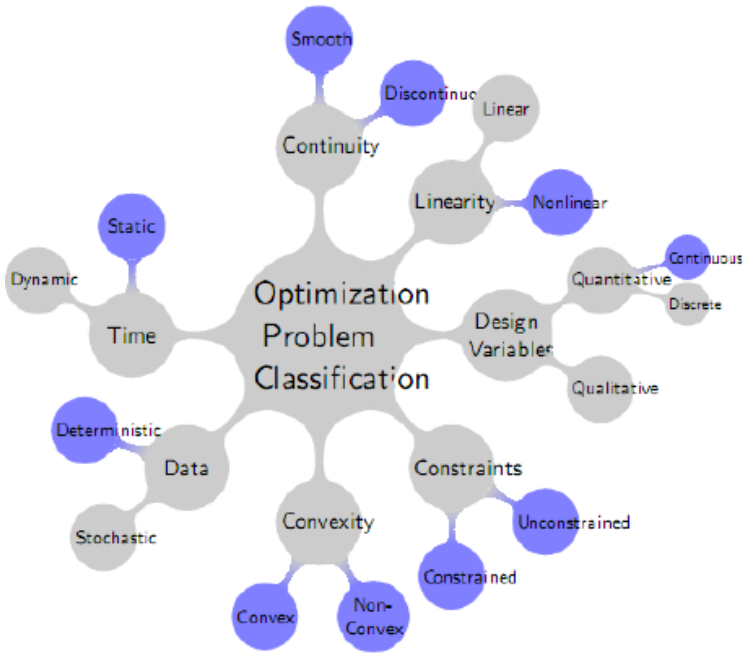


Plot for 3-D space



Plot for 4-D space

Classification of Optimization problems:



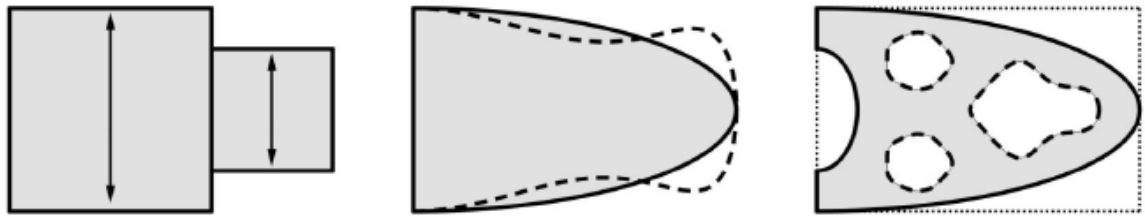
Basis for classifying the optimization problems:

- Continuity of J
- Linearity of J
- Types of design variables
- Types of constraints
- Nature of data
- Time variant / invariant

Structural Optimization

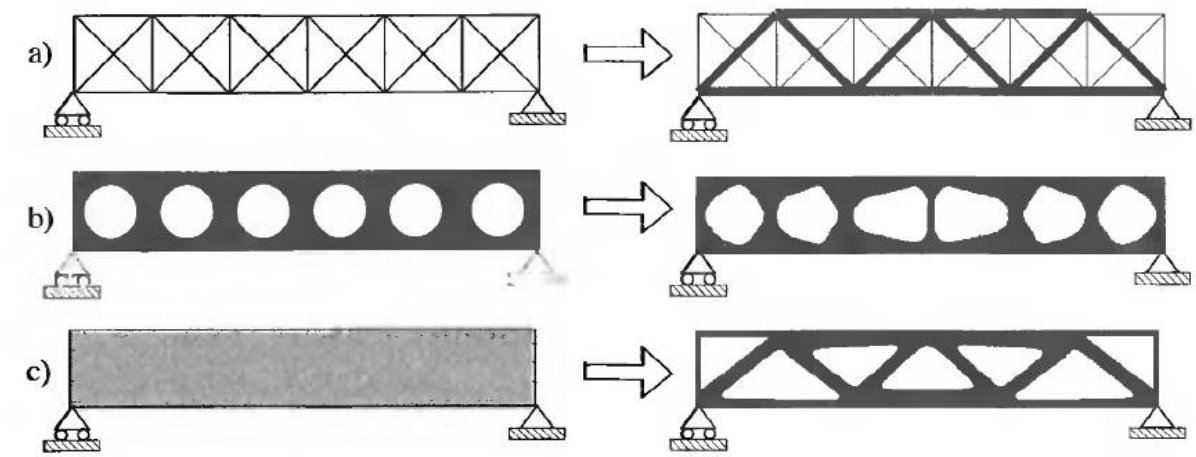
Sizing, shape and topology address different aspects of structural design problem. In a typical sizing (parametric) problem, the goal may be to find the optimal thickness distribution of a linearly elastic plate or optimal thickness distribution of a linearly elastic plate or truss structure. The optimal thickness distribution minimizes (or maximizes) a physical quantity such as mean compliance (external work), peak stress, deflection, etc. while equilibrium and other constraints on the state and design variables are satisfied. The design variable is the thickness of the plate and the state variable may be its deflection. The main feature of the sizing problem is that the domain of the design model and state variables is known apriority and is fixed throughout the optimization process. On the other hand, in shape optimization problem, the goal is to find the optimum shape of the domain, that is the shape problem is defined on a domain which is now a design variable. Topology optimization of the solid structures involves the determination of features such as the number and location and shape of the holes and connectivity of the domain.

In the aspects of a solid/object body:



1) Parametric optimization 2) Shape optimization 3) Topology optimization

In the aspects of a truss/2D frame:



a) Sizing/Parametric Optimization of 2D truss structure b) Shape optimization of 2D plate with circular holes c) Complete solid/filled body with topology optimization

Initial problem formulations are shown on the left hand side and optimized structures on right hand side.

Methodology

Parametric Shape Optimization

Consider the membrane problem. Suppose that we wish to find an optimal thickness distribution $h \in H$ at minimizes the objective function

$$J(h) = \iint_\Omega j(u(x))\, dx,$$

Where J : R → R is a specified continuously differentiable function
Note that the objective function does not contain h explicitly. Rather, u depends on h via the solution of

$$-\nabla \cdot (h(x)\nabla u(x)) = f(x),\ x \in \Omega, \quad u(x) = 0,\ x \in \partial\Omega.$$

In addition, we want to minimize J over an admissible set of thickness functions H. For instance,

$$\mathcal{H} = \left\{ h : \Omega \to \mathbb{R}_+ \;\middle|\; h_{\min} \le h(x) \le h_{\max}, \iint_\Omega h(x)\, dx = h_0|\Omega| \right\}$$

Solution with Projected Gradient Descent

If we are able to compute the functional derivative δJ(h) / δh of J with respect to h, we can use the projected gradient algorithm to find the optimal h:

$$h_{n+1} = P_{\mathcal{H}} \left(h_n - \alpha_n \frac{\delta J(h_n)}{\delta h} \right)$$

We compute the projection $P_H(h)$ of h : Ω → R as follows:

$$P_{\mathcal{H}}(h) = \max(h_{\min}, \min(h + \ell, h_{\max})),$$

where the constant $l \in R$ is chosen such that

$$\iint_\Omega P_{\mathcal{H}}(h)(x)\, dx = h_0|\Omega|.$$

In practice, simple algorithms like the bisection method are used to estimate l

Adjoint Method to compute δJ(h) / δh

Direct calculation of δJ(h) / δh is cumbersome since J depends indirectly on h via the dependence of u on h.
The adjoint method provides an efficient and elegant alternative to compute this functional derivative.
The Lagrangian for the constrained minimization of J is written using the adjoint field p : Ω → R as

$$L(h,u,p) = \iint_\Omega j(u(x))\, dx - \iint_\Omega p\left(\nabla \cdot (h(x)\nabla u(x)) + f(x)\right)\, dx$$

Note that setting δL(h; u; p)/ δp = 0 yields the governing equation for u. We will call this the primal equation.
δL(h; u; p)/ δp = 0 is called the adjoint equation. For the current problem, the adjoint equation takes the form

$$-\nabla \cdot (h(x)\nabla p(x)) = -j'(u(x)), \quad x \in \Omega,$$
$$p(x) = 0, \quad x \in \partial\Omega.$$

Denote the solutions of the primal and adjoint equations as uh and ph, respectively.
The required gradient δJ/δh : $\Omega \rightarrow$ R is now computed as

$$\frac{\delta J(h)}{\delta h}(x) = \nabla u_h(x) \cdot \nabla p_h(x).$$

This technically corresponds to an L2 projection of the functional derivative δJ/δh

Regularization of Iterates

The iterates h_n(x) often do not have the right regularity, and hence need to be regularized.
The simplest option is to apply a filter to h. For instance, the Heaviside filter acts on h to produce a more regular H_βh, for large positive constant β, as

$$H_\beta h(x) = 1 - \exp(-\beta h(x)) + h(x)\exp(-\beta).$$

One could also instead do a smoothing of h directly by solving, for small β > 0,

$$-\epsilon^2\nabla^2\tilde{h}(x) + \tilde{h}(x) = h(x), \quad x \in \Omega,$$
$$\nabla\tilde{h}(x) \cdot n(x) = 0, \quad x \in \partial\Omega.$$
$$-\epsilon^2\nabla^2 q(x) + q(x) = \nabla u_h(x) \cdot \nabla p_h(x), \quad x \in \Omega,$$
$$\nabla q(x) \cdot n(x) = 0, \quad x \in \partial\Omega.$$

Denoting by qh the solution of this equation,

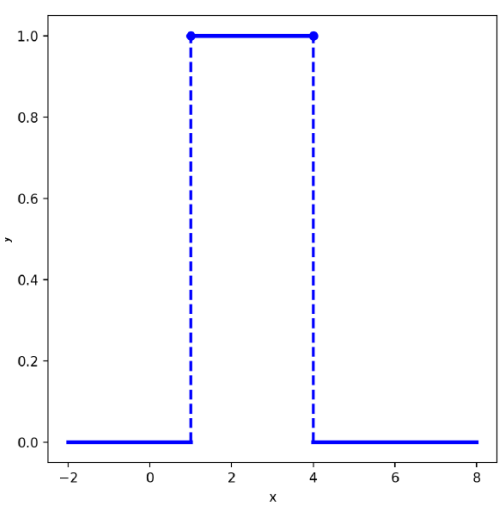
$$(\delta J(h)/\delta h)(x) = q_h(x)$$

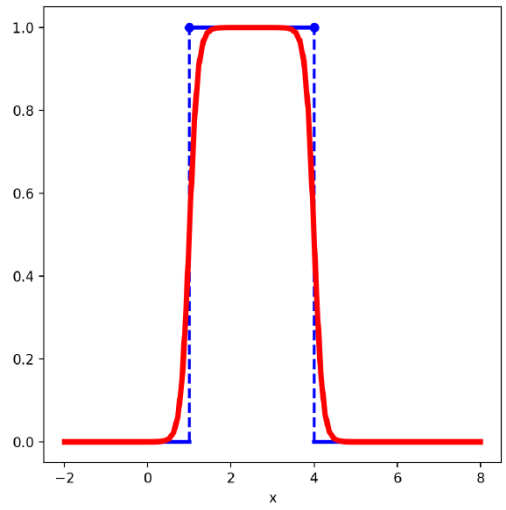
Pseudo Parametric Shape Optimization Algorithm

1. Initialize thickness h.
2. Iterate until convergence:
   - Solve the primal equation to find uh.
   - Solve the adjoint equation to find ph.
   - Compute gradient of objective function, δJ/δh, using uh and ph.
   - Update h: h = h - $\alpha * (\delta$J/δh)
   - Enforce constraints on h, if any.
   - Regularize h.

In order to extend the Shape Optimization Algorithm for Topology Optimization we follow the below approach

Density based Homogenization





Given a subdomain A ⊆ Ω of the domain Ω ⊆ R^d, d > 1, the characteristic function x_A : Ω → R defined as

$$\chi_A(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in A, \\ 0, & \boldsymbol{x} \notin A, \end{cases}$$

We define a density ρ_A : Ω → [0; 1] as a smooth approximation of the characteristic function x_A

Given a density function ρ, the subdomain of Ω that it approximates can be extracted from it numerically using a tolerance $\epsilon > 0$ as

$$A(\rho) = \{\boldsymbol{x} \in \Omega \,|\, \rho(\boldsymbol{x}) > \epsilon\}.$$

SIMP: Smooth Interpolation of Material Properties

Consider an initial design where the material, with material property C, occupies a region Ω ⊆ R^d.

If the optimal material distribution that satisfies the external constraints only occupies a subdomain A ⊆ Ω, then this can be approximated using the ersatz material approximation as C_ϵ : Ω → R, for small $\epsilon > 0$:

$$\boldsymbol{C}_\epsilon(\boldsymbol{x}) = \chi_A(\boldsymbol{x})\boldsymbol{C} + (1 - \chi_A(\boldsymbol{x}))\epsilon\boldsymbol{C}$$

Smoothing x_A with a density ρ, we can reframe the topology optimization problem in terms of density by replacing the material property C with C_ϵ. This is called the Smooth Interpolation of Material Properties (SIMP) method.

The rest of the algorithm is the same as discussed for parametric optimization.

Finally, the optimal domain can be extracted from ρ. In practice, interpolating functions ζ: R →R are used to sharpen the density. A common choice is ζ (t) = t^3.

Sigmund-Bendsoe method for Structural Optimization

The 4 major parts required for topology optimization are as follows:

1. Problem definition and compliance minimization
2. Optimality Criteria based optimizer
3. Mesh Independency filter
4. Finite element part

The disadvantage of homogenization based approach mentioned above is that determining the optimal microstructures (material presence) is cumbersome if resolved, especially for non-compliance minimization problems. So we follow the second approach, power-law based method (SIMP), where material properties are assumed constant (isotropic) within each element. The material properties are defined as some power times material properties.

Problem Definition:

Considering the optimization problem for minimizing the compliance (i.e Maximizing stiffness):

$$\begin{aligned} \min_{\mathbf{x}}: \quad & c(\mathbf{x}) = \mathbf{U}^T\mathbf{K}\mathbf{U} = \sum_{e=1}^{N} (x_e)^p \, \mathbf{u}_e^T \, \mathbf{k}_0 \, \mathbf{u}_e \\ \text{subject to}: \quad & \frac{V(\mathbf{x})}{V_0} = f \\ : \quad & \mathbf{K}\mathbf{U} = \mathbf{F} \\ : \quad & \mathbf{0} < \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{1} \end{aligned}$$

Where U and F are global displacements and force vectors, K is global stiffness matrix and ue is element displacement matrix and ke is element stiffness matrix
x is design variable vector, which in this case will be density
xmin is the limit established on relative densities to prevent singularities
N=nelx X nely is the number of discrete elements in the design domain
P is the penalization index used in SIMP method (p=3 is taken based on experiments)
V(x)is the Material volume and Vo is the design volume
F(volfrac) is the volume fraction

The above optimization problem can be solved by various approaches:

1. Optimality Criteria method
2. Method of Moving Asymptotes
3. Sequential Linear Programming

The simplest approach that can be followed is the Optimality Criteria Method.
The solution updating scheme is taken from Bendsoe, where xnew is given as:

$$x_e^{\text{new}} =$$

$$\begin{cases} \max(x_{\min}, x_e - m) \\ \quad \text{if} \quad x_e B_e^{\eta} \leq \max(x_{\min}, x_e - m)\,, \\ x_e B_e^{\eta} \\ \quad \text{if} \quad \max(x_{\min}, x_e - m) < x_e B_e^{\eta} < \min(1, x_e + m)\,, \\ \min(1, x_e + m) \\ \quad \text{if} \quad \min(1, x_e + m) \leq x_e B_e^{\eta}\,, \end{cases}$$

Where m is a move-limit, η= ½ is damping coefficient and Be is found from optimality KKT conditions:

$$B_e = \frac{-\dfrac{\partial c}{\partial x_e}}{\lambda \dfrac{\partial V}{\partial x_e}}$$

Here λ which is Langrange multiplier found by using bisection algorithm
The sensitivity of the objective function is given by:

$$\frac{\partial c}{\partial x_e} = -p(x_e)^{p-1}\, \mathbf{u}_e^T\, \mathbf{k}_0\, \mathbf{u}_e$$

To ensure that the solutions to the above topology problem exist, we use a filter method called the Mesh Independence. It updates the sensitivities of the problem as:

$$\widehat{\frac{\partial c}{\partial x_e}} = \frac{1}{x_e \sum_{f=1}^{N} \hat{H}_f} \sum_{f=1}^{N} \hat{H}_f\, x_f\, \frac{\partial c}{\partial x_f}$$

The convolution operator Hf is given by:

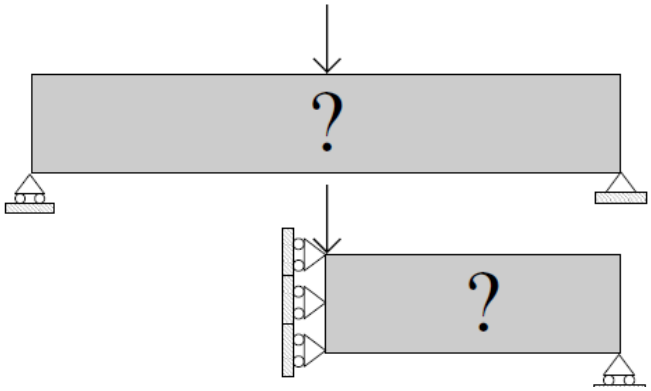
$$\hat{H}_f = r_{\min} - \operatorname{dist}(e, f)\,,$$

$$\{f \in N \mid \operatorname{dist}(e, f) \leq r_{\min}\}, \quad e = 1, \ldots, N\,.$$

Where the operator dist (e,f) is the distance between centre of elements e and f. The convolution operator makes the value of area outside the filter as zero. The decrease in the value is linear from the centre of element.

Topology Optimization for Structures program breakdown

1. Case of Simply supported Beam:



The main input arguments required to be specified by the user for topology optimization can be reduced to just 4 main arguments viz:

Topology (nelx, nely, volfrac, penal, rmin)

Where nelx and nely are number of elements in horizontal and vertical direction each of 1 unit, thus they also denote the horizontal and vertical dimensions.
Volfrac is the volume fraction required to be kept in the optimized structure.
Penal is the penalization factor incorporated because of the use of SIMP method and rmin is the filter size (radius about which a gradient effect is kept in, to avoid the checker-board effect).The remaining boundary conditions like force and fixed nodes are directly incorporated in the Finite element part of the code, which makes the code specific to the nature of problem addressing.
The code is divided into 4 parts as discussed earlier:

1. Main program (Objective function definition and iteration schemes):
   It distributes the material evenly within the design space (which means equal stiffness throughout the geometry). It will make a function cal to FE function defined which returns a displacement vector. The element K function defined below is also called upon. For the assemblage of the element matrices, n1 and n2 are defined to obtain element displacement vector from Global displacement vector. The sensitivity analysis is followed by Mesh-Independency Filter
2. OC based optimizer (Optimality Criteria)
   Here the langrange multiplier satisfying the volume constraint is obtained using bisection algorithm by initial guess of lower l1 and upper l2. The bisection continues until the convergence criteria is achieved.
3. Mesh independency filter:
   The rmin radius defined in the argument, helps defining the mesh independency of the density distribution. For r>rmin the density value is set to 0. For r<rmin density value scales from 1 to 0 based on the location of centre of element
4. The finite element part makes use of MATLAB function called sparse, where it neglects the density 0 locations straight away by just storing non-zero elements
   The FE part loops over element stiffness matrices to obtain a global stiffness matrix. Here the convention of node numbering used is Columnwise left to right.

Every node has 2 degrees of freedom ux and uy, thus a square element having four such nodes at each corner ill have total of 8 elements.
A Boundary condition in terms of force acting at the mid-span of 1 unit. This is given as F(2,1)=-1, which is a vertical force in upper left corner.
Supports (Zero displacements) are eliminated from total degrees of freedom by categorizing them as fixed degrees of freedom.
Thus we evaluate for Free Degrees of freedom

Free DoF=Total DoF-Fixed DoF

From this we calculate Global displacement vector as:

U(Free DoF)=F(Free DoF)/K(Free doF)

Simply Supported Beam Structural Topology Optimization

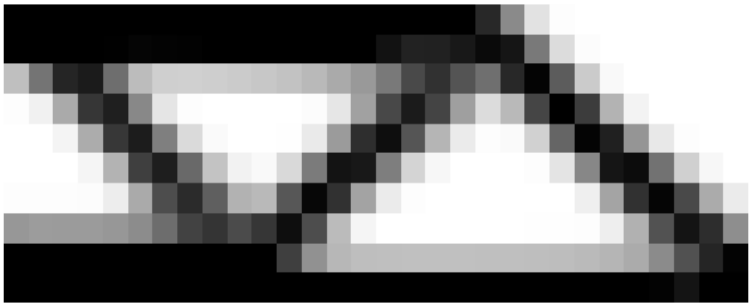
In the code provided in **Appendix A** topology (nelx,nely,volfrac,penal,rmin) where nelx and nely are the number of elements in the horizontal and vertical directions, respectively, volfrac
is the volume fraction, penal is the penalization power and rmin is the filter size (divided by element size).

The Sigmund code implemented on structural mechanic problem as follows:
A Point load is applied at the centre of beam supported at two supports can be simplified utilizing its symmetry as follows:

Results:

>> topology (nelx,nely,volfrac,penal,rmin)

>> topology (30, 10, 0.5, 3.0, 1.5)



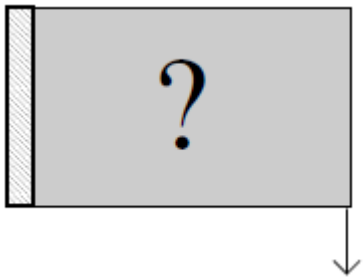
Further refinement can be done as follows:

>> topology (80, 50, 0.5, 3.0, 1.5)

The optimized topology obtained is as follows:

Extension of Code to various Structures problems

1. **Cantilever Problem:**

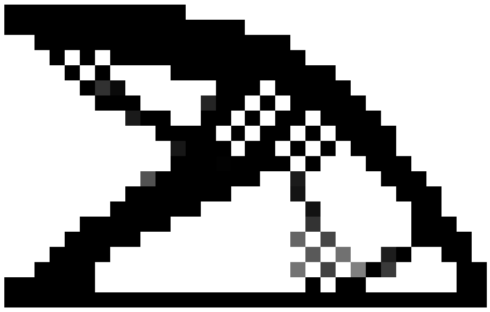


The only changes that are required to be made in the code are the Force boundary conditions which is now at the right bottom.

Also the fixed degrees of freedom will change which is now at the left position throughout.

Rest remains the same, unless you change the element type OR interpolation function

Checkerboard effect:

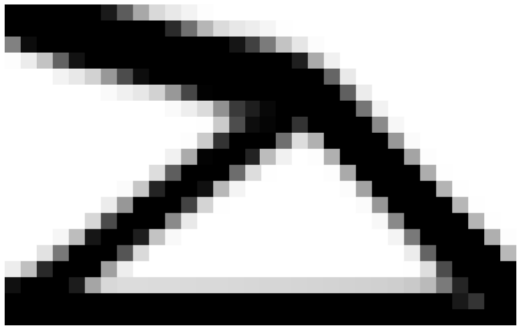


This effect is highly undesirable in topology optimization. It can be observer when rmin value is kept too low. To avoid this one can always increase the rmin value

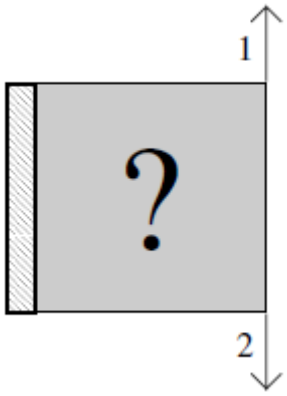
The correct argument go as:

Cantilever(32,20,0.4,3.0,1.2)

Result (Code Provided in Appendix B):



2. Multiple Load Problem:

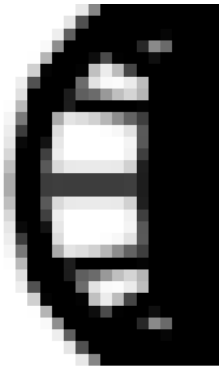


The problem definition now changes, so the objective function for minimal compliance (max. stiffness) needs to be modified accordingly:

$$c(\mathbf{x}) = \sum_{i=1}^{2} \mathbf{U}_i^T \mathbf{K} \mathbf{U}_i$$

This follows few modifications in the code, primarily the Force Boundary condition and Sensitivity loop of iteration. The result obtained for the following case is :

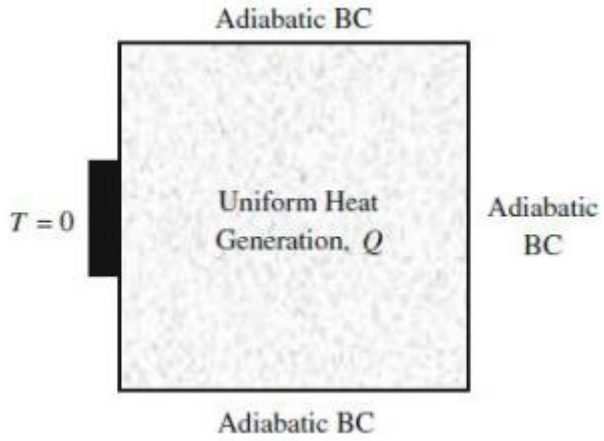
Multiload(30,30,0.3,0.4,1.2)



Note that rmin is selected such that we prevent the checker board effect to maximum extent. The result also can be logically explained, such that material stiffness should be more at the point of application (rightmost),which also can be seen in the code.

The code for the same is attached in Appendix C

Topology Optimization for heat conduction problem



Consider a 2D heat conduction problem as shown above:

Here the problem statement is to find the optimal path for heat conduction from a square plate 2D. The problem can be broken down to minimizing the mean temperature in design domain. Once again the SIMP interpolation method can be adopted to find the topology. However instead of density value, thermal conductivity value is associated in the optimization. The constraint over volume is also applied along ith uniform heat generation over the distributed domain. All edges are adiabatic except the heat sink with temperature 0K at the centre of the left edge of domain.

The governing equation which is Poisson's equation is given as:

$$-\nabla k\rho\nabla T=Q$$

The SIMP interpolation can constructed between density and thermal conductivity variable:

$$k(\rho) = (0.001 + 0.999\rho^p)k_0$$

Problem definition:

$$\text{Find:} \quad \rho$$

$$Minimize: D = \frac{1}{2}\int_\Omega k(\nabla T)^2 d\Omega$$

$$Subject\ to: -\nabla[k(\rho)\nabla T] = Q$$

$$\int_{\Omega_d} \rho d\Omega_d - V_{max} \leq 0$$

$$0 \leq \rho \leq 1$$

$$Given: \quad k(\rho) = (0.001 + 0.999\rho^p)k_0$$

The FE equation used for programing is given as follows:

$$\boldsymbol{KT=F}$$

Where K is global conductivity matrix, T is global nodal temperature and F is applied heat load. The objective function can be given in the form of summation for FE as follows:

$$D = \frac{1}{2}\boldsymbol{T^T K T} = \frac{1}{2}\sum_{e=1}^{N_e} \boldsymbol{T_e^T K_e T_e}$$

Thus the entire problem definition can be converted in terms of Finite Elements as:

$$\text{Find: } x = (\rho_1, \rho_2, \dots . \rho_i, \dots \dots \rho_{N_e}), \quad \rho_i = 0 \text{ or } 1$$

$$Minimize: D = \frac{1}{2}\boldsymbol{T^T K T} = \frac{1}{2}\sum_{e=1}^{N_e} \boldsymbol{T_e^T K_e T_e}$$

$$Subject\ to: \boldsymbol{K(\rho)T} = \boldsymbol{F}$$

$$\sum \rho_i V_i \leq V_{max}$$

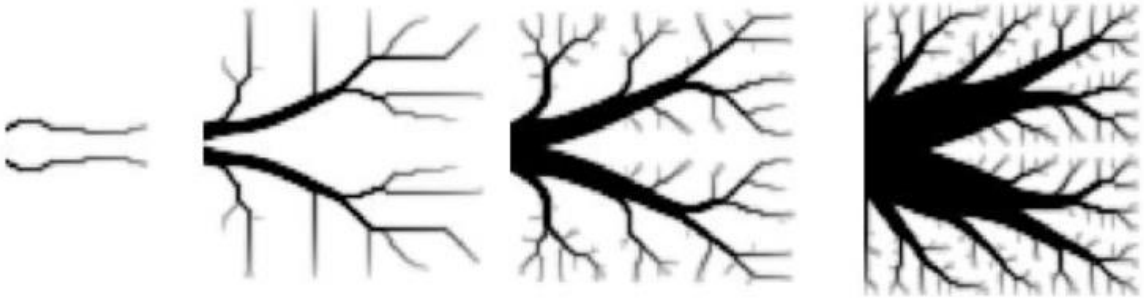
$$0 \leq x_{min} \leq x_i \leq 1$$

$$Given: \ K_e(\rho) = (0.001 + 0.999\rho^p)K_e^0$$

Other inputs that can be given to the program are as follows:

Number of elements	Volume fraction	Filter Radius	Boundary conditions	Loading
6400 quadrilateral elements	0.4	1.2	Adiabatic boundaries, heat sink at the center of the left side	Uniform heat source

Results for different volume fractions:



a) b) c) d)

Variation in the results with change in volume fraction f varying from a) 0.05, b) 0.1, c) 0.5, d) 0.6

The modified Sigmund –Bendsoe code for the above thermal problem is provided in Appendix D

Conclusion

The Topology Optimization method using finite element methods based on optimality criteria can be successfully implemented for both Structural and thermal problems.
The Sigmund-Bendsoe approach to divide the approach using 4 subparts as explained above can be extended to various cases as demonstrated.
The method can very well be extended to various structures problem like Simply Supported, Cantilever and Multi-load case by suitable modifications in the code.
Also the same method can handle various thermal conduction problems for various Boundary conditions. The fluid mechanic optimization problems for minimum resistance (viscous) paths can be also be explored by the same approach where the objective function needs to be modified.
The various future extensions multi-physics extensions to this technique are:

1. Thermo-structural problems (Structures+Thermal)
2. Fluid-structure interaction problems
3. 3D structural optimization
4. 3D thermal conductivity optimization

Various other techniques that can be compared along with OC criteria are:

Method of Moving average, Density based filtering

Based on the literature survey and sources referred, the proposed method is efficient than rest of the methods mentioned above.

Appendix A

Following the above mentioned algorithm and the techniques, a MATLAB code proposed by Sigmund using Finite elements for Simply Supported Beam:

```
function top(nelx,nely,volfrac,penal,rmin);
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk;
  c = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  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 DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end
%%%%%%%%% OPTIMALITY CRITERIA UPDATE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));
  if sum(sum(xnew)) - volfrac*nelx*nely > 0;
    l1 = lmid;
  else
    l2 = lmid;
  end
end
%%%%%%%%% MESH-INDEPENDENCY FILTER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
```

```
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
%%%%%%%%% FE-ANALYSIS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk;
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F(2,1) = -1;
fixeddofs   = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
%%%%%%%%% ELEMENT STIFFNESS MATRIX
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1.;
nu = 0.3;
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
   -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
                  k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
                  k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
                  k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
                  k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
                  k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix B

Modified Sigmund-Bendsoe Code for Cantilever Beam Problem-

```
function top(nelx,nely,volfrac,penal,rmin);
    %nelx=elements along x axis (X dimension)
    %volfrac=eg 0.5 means 50% keep 0.3 30% material keep
    %penal= p..... Density^p p=3 (exp)
    %rmin=radius around the element to consider for density (filter radius)
    %to avoid checker board effect
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
%x is density
loop = 0;
change = 1.;
%change=diff in density(design variable) prev. and next iteration
% START ITERATION
while change > 0.01  %convergence criteria
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk; %Single element stiffness matrix
  c = 0.; %compliance
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue; %dc compliance
sensitivity
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  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 DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end %1=white 0=black for x; inverse for -x
%%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);%
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));%Beta=-
dc/lmid
  if sum(sum(xnew)) - volfrac*nelx*nely > 0; %similar to Bisection algo for lmid
to keef volf=0.5
    l1 = lmid;
  else
    l2 = lmid;
```

```
  end
end
%%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk; %1 element stiffnessmatrix
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1)); %K global stiff matrix
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1); %Force matrix
for elx = 1:nelx     %assembling stiffness matrix to global matrix
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)(Generates vector)
F(2*(nelx+1)*(nely+1),1) = -1;
fixeddofs = [1:2*(nely+1)];
alldofs     = [1:2*(nely+1)*(nelx+1)]; %all nodes
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
%%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1.;
nu = 0.3;
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
   -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
                  k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
                  k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
                  k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
                  k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
                  k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

Appendix C

Modified Sigmund-Bendsoe Code for Multiple Loading Problem-

```
function top(nelx,nely,volfrac,penal,rmin);

    %nelx=elements along x axis (X dimension)
    %volfrac=eg 0.5 means 50% keep 0.3 30% material keep
    %penal= p..... Density^p p=3 (exp)
    %rmin=radius around the element to consider for density (filter radius)
    %to avoid checker board effect
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
%x is density
loop = 0;
change = 1.;
%change=diff in density(design variable) prev. and next iteration
% START ITERATION
while change > 0.01  %convergence criteria
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk; %Single element stiffness matrix
  c = 0.; %compliance
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = 0.;
      for i = 1:2
         Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2;2*n2+1;2*n2+2;2*n1+1;2*n1+2],i);
         c  = c + x(ely,elx)^penal*Ue'*KE*Ue;
         dc(ely,elx) = dc(ely,elx) - penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue; %dc
compliance sensitivity
      end
    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  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 DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end %1=white 0=black for x; inverse for -x
%%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);%
```

```
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));%Beta=-
dc/lmid
  if sum(sum(xnew)) - volfrac*nelx*nely > 0; %similar to Bisection algo for lmid
to keef volf=0.5
    l1 = lmid;
  else
    l2 = lmid;
  end
end
%%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk; %1 element stiffnessmatrix
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1)); %K global stiff matrix
F = sparse(2*(nely+1)*(nelx+1),2); U = sparse(2*(nely+1)*(nelx+1),2); %Force
matrix
for elx = 1:nelx     %assembling stiffness matrix to global matrix
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)(Generates vector)
%F(2*(nelx+1)*(nely+1),1) = -1.;
F(2*(nelx)*(nely+1)+2,2) = 1.;
fixeddofs   = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)]; %all nodes
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;
%%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1.;
nu = 0.3;
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
   -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
                  k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
                  k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
```

```
k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

Appendix D

Modified Sigmund-Bendsoe Code for 2D Thermal Conduction Optimization with Adiabatic Boundary Conditions Problem-

```
function sbesoh(nelx,nely,volfrac,er,rmin)
%initialize
x(1:nely,1:nelx)=1.;
for ely=1:nely
    for elx=1:nelx
        if sqrt((ely-nely/2.)^2+(elx-nelx/2.)^2)<nely/(nelx/2)
            passive(ely,elx)=1.;
            x(ely,elx)=1.;
        else
            passive(ely,elx)=0.;
        end
    end
end
vol=1.;
i=0;
change=1.; penal=3.;
%start iTH ITERATION
% %T(i,1)=i;
while change > 0.00001
    %penal=penal+0.1;
        i = i+1;
        T(i,1)=i;
        vol=max(vol*(1-er),volfrac);
        if i >1; olddc = dc;
        end
        %FE-ANALYSIS
            [U]=FEh(nelx,nely,x,penal);
            [Uj]=FEhj(nelx,nely,x,penal);
        %OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
        [KE] = lkh;
        c(i) = 0.;
    S(i,1)=0;
    for ely=1:nely
        for elx = 1:nelx
            n1=(nely+1)*(elx-1)+ely;
            n2=(nely+1)*elx    +ely;
            Ue = U([n1;n2;n2+1;n1+1],1);
            Uej = Uj([n1;n2;n2+1;n1+1],1);
            Un=U([1+(nelx+1)*nelx/2+nelx/2;1+(nelx+1)*nelx/2+nelx/2;...
                1+(nelx+1)*nelx/2+nelx/2;1+(nelx+1)*nelx/2+nelx/2],1);
            c(i)=c(i)+(0.001+0.999*0.5*x(elyelx)^penal)*Un'*KE*Ur;
            dc(ely,elx)=0.999*0.5*penal*x(ely,elx)^(penal-1)*Uej'*KE*Ue;
        end
    end

    %FILTERING OF SENSITIVITIES
    [dc]=check1(nelx,nely,rmin,x,dc);
    % %STABILIZATION OF EVOLUTIONARY PROCESS
    if i > 1;
        dc = (dc+olddc)/2.;
    end
    %BESO DESIGN UPDATE
    [x]=ADDDEL(nelx,nely,vol,dc,x,passive);
```

```
    %CHANGE
    if i>10;
        change=abs(sum(c(i-9:i-5))-sum(c(i-4:i)))/sum(c(i-4:i));
    end
    %PLOT DENSITIES
    colormap(gray);imagesc(-x);axis equal;axis tight;axis off;pause(1e-6);
end
end
%FE-ANALYSIS%
function[U]=FEh(nelx,nely,x,penal)
    [KE]=lkh;
    K = sparse((nelx+1)*(nely+1), (nelx+1)*(nely+1));
    F = sparse((nely+1)*(nelx+1),1);
    U=sparse(((nely+1)*(nelx+1)),1);
    for elx=1:nelx
        for ely=1:nely
            n1=(nely+1)*(elx-1)+ely;
            n2=(nely+1)*elx    +ely;
            edof = [n1;n2;n2+1;n1+1];
            K(edof,edof) = K(edof,edof) +(0.001+0.999* x(ely,elx).^penal)*KE;
        end
    end
end
%Two Heat Sources
F((((nelx/2)+1)+(nelx+1)),1)=0.1;
F(1+nelx*(nelx+1)+(nelx/2)-(nelx+1),1)=0.1;
%Four Heat Sources
F((((nelx/2)+1)+(nelx+1)),1)=0.1;
F(1+nelx*(nelx+1)+(nelx/2)-(nelx+1),1)=0.1;
F(2+(nelx+1)*(nelx/2),1)=0.1;
F(nelx+(nelx+1)*(nelx/2),1)=0.1;
%Zero Temperature at the Boundaries
a=[1:nelx+1:1+(nelx*(nelx+1))];
b=[1:nelx+1];
c=[nelx+1:nelx+1:(nelx+1)*(nelx+1)];
d=[1+(nelx*(nelx+1)):(nelx+1)*(nelx+1)];
fixeddofs=[a,b,c,d];
alldofs    = (1:(nely+1)*(nelx+1));
freedofs=setdiff(alldofs,fixeddofs);
%SOLVING
U(freedofs,:)=K(freedofs,freedofs)\F(freedofs,:);
U(fixeddofs,:)=0;
function [Uj]=FEhj(nelx,nely,x,penal)
    [KE] = lkh;
    K = sparse((nelx+1)*(nely+1), (nelx+1)*(nely+1));
    Fj = sparse((nely+1)*(nelx+1),1);
    Uj =sparse((nely+1)*(nelx+1),1);
    for ely = 1:nely
        for elx = 1:nelx
            n1 = (nely+1)*(elx-1)+ely;
            n2 = (nely+1)* elx   +ely;
            edof = [n1;n2;n2+1;n1+1];
            K(edof,edof)=K(edof,edof)+(0.001+0.999*x(ely,elx)^penal)*KE;
        end
    end
end
    % DEFINE LOADSAND SUPPORTS(HALF MBB-BEAM)
    Fj(:,1)= 0;
    Fj(1+(nelx+1)*nelx/2+nelx/2,1)=1;
```

```
    %Zero Temperature at the Boundaries
    a=[1:nelx+1:1+(nelx*(nelx+1))];
    b=[1:nelx+1];
    c=[nelx+1:nelx+1:(nelx+1)*(nelx+1)];
    d=[1+(nelx*(nelx+1)):(nelx+1)*(nelx+1)];
    fixeddofs=[a,b,c,d];
    alldofs     = (1:(nely+1)*(nelx+1));
    freedofs    = setdiff(alldofs,fixeddofs);
    % SOLVING
    Uj(freedofs,:) = K(freedofs,freedofs)\Fj(freedofs,:);
    Uj(fixeddofs,:)= 0;
%OPTIMALITY CRITERIA UPDATE%
function [x]=ADDDEL(nelx,nely,volfra,dc,x,passive)
    l1 = min(min(dc));
    l2 = max(max(dc));
    while ((l2-l1)/l2 > 1.0e-5)
        th=(l1+l2)/2.0;
        x=max(0.001,sign(dc-th));
        x(find(passive))=1.;
        if sum(sum(x))-volfra*(nelx*nely)>0;
            l1=th;
        else
            l2=th;
        end
    end
end
%MESH INDEPENDENCY FILTER%
function[dcf]=check1(nelx,nely,rmin,x,dc)
    dcf=zeros(nely,nelx);
    for i = 1:nelx
        for j = 1:nely
            sum=0.0;
            for k=max(i-floor(rmin),1):min(i+floor(rmin),nelx)
                for l=max(j-floor(rmin),1):min(j+floor(rmin),nely)
                    fac=rmin-sqrt((i-k)^2+(j-l)^2);
                    sum=sum+max(0,fac);
                    dcf(j,i)=dcf(j,i) + max(0,fac)*dc(l,k);
                end
            end
            dcf(j,i)=dcf(j,i)/sum;
        end
    end
end
```

References

1. Sigmund, O. 1994: Design of material structures using topology optimization. Ph.D. Thesis, Department of Solid Mechanics, Technical University of Denmark
2. Sigmund, O. 1997: On the design of compliant mechanisms using topology optimization. Mech. Struct. Mach. 25, 495–526
3. Zhao, C.; Hornby, P.; Steven, G.P.; Xie, Y.M. 1998: A generalized evolutionary method for numerical topology optimization of structures under static loading conditions. Struct. Optimization 15, 251–260
4. Zhou, M.; Rozvany, G.I.N. 1991: The COC algorithm, part II: Topological, geometry and generalized shape optimization. Comp. Meth. Appl. Mech. Engrng. 89, 197–224
5. Sigmund, O.; Petersson, J. 1998: Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. Struct.
   Optim. 16, 68–75
6. Svanberg, K. 1987: The method of moving asymptotes – a new method for structural optimization. Int. J. Numer. Meth. Engrg. 24, 359–373
7. Xie, Y.M.; Steven, G.P. 1997: Evolutionary structural optimization. Berlin, Heidelberg, New York: Springer
8. Zhao, C.; Hornby, P.; Steven, G.P.; Xie, Y.M. 1998: A generalized evolutionary method for numerical topology optimization of structures under static loading conditions. Struct. Optim. 15, 251–260