Project, structural optimization

Structural optimization, FHLN01

Authors: Kajsa Söderhjelm, Alexander Jörud

March 13, 2017

Contents

Nomenclature

C	Compliance	$\boldsymbol{K}$	Stiffness matrix
$\boldsymbol{D}$	Tangent stiffness tensor	$\boldsymbol{u}$	Nodal displacements
$\boldsymbol{k}^0$	Normalized stiffness matrix	E	Young's modulus
R	Radius	r	Length scale parameter
λ	Lagrangian multiplier	V	Volume
l	Length of a bar	ρ	Density
p	Penalization factor	ν	Poisson's ratio
$\boldsymbol{M}$	Mass matrix	$\boldsymbol{a}$	Area vector
$\widetilde{\nabla}$	Linear elasticity operator	$\widetilde{\boldsymbol{\rho}}$	Filtered density vector
$\boldsymbol{F}$	External force vector	$\boldsymbol{y}$	Intervening variable
$\boldsymbol{N}_e$	Neighborhood vector	$\boldsymbol{w}$	Weight function vector
$\boldsymbol{v}$	Volume vector	$\boldsymbol{x}$	Design variable
L_j	Moving asymptote		

1 Introduction

1.1 Problem formulation

The objective consists of analyzing different optimization methods of the structure depicted in figure 1. The optimization methods include CONLIN, MMA and SIMP. Two different filters are implemented and discussed regarding the SIMP method. The equipment provided for analyzing the assignment is MATLAB software and CALFEM toolbox. The compliance of the beam illustrated in figure 1 should be minimized where the material in the beam is linear elastic, homogeneous and isotropic.

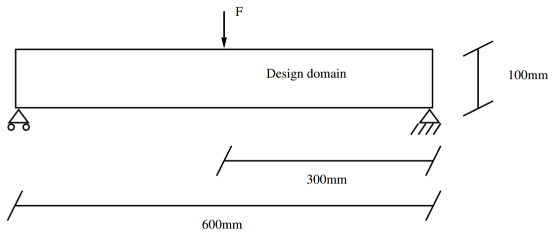


Figure 1: Represented is a illustration of the geometry and boundary condition, i.e. the supports in the corners of the beam used in the analyze.

The first task is to perform a size optimization where the beam is built up with a truss structure of circular bars. The maximum cross section diameter is $d_{max} = 20mm$ and the maximum volume of the structure is $V_{max} = 2000mm^3$. The methods used to optimize the cross sectional areas of the bars in the provided geometry are the two similar algorithms CONLIN and MMA for a range of d_{min} and initial values. The second task is a topology optimization that should be solved for two different finite element discretization. Both are built up with isoparametric four node elements, one coarser and one finer. The maximum volume of the structure is $V_{max} = 0.4V_{box}$ where V_{box} is the volume defined by figure 1, with a thickness of $20 * 10^{-3}m$. Poisson's ratio ν is set to 0.3 and plane stress is used. To solve this task the SIMP algorithm will be used with and without filters. The filters that will be used are a density filter and the Helmholtz' PDE based filter. For simplicity the tasks uses a symmetry plane at $x = 300mm$. This means that only one half of the geometry will be analyzed using boundary conditions connecting to the second part, where the second part consist of the rolling support.

2 Optimization problem

The compliance which is a good measurement of the stiffness in the structure should be minimized. The simultaneous optimization problem for task one, where a truss structure is to be analyzed, becomes

$$(\boldsymbol{P})_{sf} = \begin{cases} \min\limits_{\boldsymbol{x}\in\chi} & C = \boldsymbol{F}^T\boldsymbol{u} \\ s.t. & \begin{cases} \boldsymbol{K}(\boldsymbol{x})\boldsymbol{u} = \boldsymbol{F} \\ \sum l_j x_j - V_{max} \leq 0 \\ \boldsymbol{x} \in \chi = x_j^{min} \leq x_j \leq x_j^{max}, j = 1, .., n \end{cases} \end{cases} \tag{1}$$

Where $C, \boldsymbol{F}, \boldsymbol{u}, \boldsymbol{K}$, is the compliance, the external force, the nodal displacement and the stiffness matrix respectively. x_j is the design variable as a function of the cross sectional area for the trusses in the structure and l_j is the length of the bar. The simultaneous formulation is a disadvantage for large scale problems due to the number of constraints from the equilibrium equations. In the case where the stiffness matrix is nonsingular, i.e. there exist an inverse to the matrix, the displacement can be rewritten as a function of the design variables. This provides the nested version of the problem and becomes

$$(\boldsymbol{P})_{nf} = \begin{cases} \min\limits_{\boldsymbol{x}\in\chi} & C = \boldsymbol{F}^T\boldsymbol{u}(\boldsymbol{x}) \\ s.t. & \begin{cases} \sum l_j x_j - V_{max} \leq 0 \\ \boldsymbol{x} \in \chi = x_j^{min} \leq x_j \leq x_j^{max}, j = 1, .., n \end{cases} \end{cases} \tag{2}$$

The simultaneous formulation is not convex however the nested formulation is. In task two a topology optimization is to be performed. The design variable when solving task two where a topology optimization is to be performed will be a density-type parameter ρ, and can be related to the thickness. As well as for the truss structure there will be an upper bound ρ_{max} and a lower bound ρ_{min}, where the lower bound will be close to zero. When identifying global minimums of optimization problems the Lagrangian function is used and is defined as

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) = g_0(\boldsymbol{x}) + \sum_{i=1}^{l} \lambda_i g_i(\boldsymbol{x}) \tag{3}$$

λ_i are called Lagrange multipliers and the KKT, Karush-Kuhn-Tucker, conditions are defined as

$$\begin{aligned} &\frac{\partial \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_j} \leq 0 && \text{if} && x_j = x_j^{max} \\ &\frac{\partial \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_j} = 0 && \text{if} && x_j^{min} \leq x_j \leq x_j^{max} \\ &\frac{\partial \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})}{\partial x_j} \geq 0 && \text{if} && x_j = x_j^{min} \end{aligned} \tag{4}$$

$$\lambda_i g_i(\boldsymbol{x}) = 0 \qquad g_i(\boldsymbol{x}) \leq 0 \qquad \lambda_i \geq 0 \qquad \boldsymbol{x} \in \chi$$

For convex problems the KKT points provide global optima, however for nonconvex problems these may only be local optima. For large scale structures it might be time consuming to solve the KKT conditions defined in equation (4) instead a method called Lagrangian Duality is used. The optimization problem can solved with

$$\min_{\boldsymbol{x}\in\chi} \max_{\boldsymbol{\lambda}\geq 0} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\chi} \max_{\boldsymbol{\lambda}\geq 0} \left(g_0(\boldsymbol{x}) + \sum_{i=1}^{l} \lambda_i g_i(\boldsymbol{x}) \right) \tag{5}$$

The Lagrangian $\mathcal{L}$ is to be maximized with respect to $\boldsymbol{\lambda} \geq 0$ for a given $\boldsymbol{x}$ which is after minimized with respect to $\boldsymbol{x}$. The objective function and constraints are often approximated in such a way where it will be more computationally efficient to solve the so called dual Lagrangian problem which corresponds to equation (5). The dual objective function is defined as

$$\varphi(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\chi} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) \tag{6}$$

φ is always concave which means that it is easy to maximize. If $\min_{\boldsymbol{x}\in\chi} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda})$ has one solution for a given $\boldsymbol{\lambda}$, φ is differentiable at $\boldsymbol{\lambda}$ and can be solved

$$\frac{\partial \varphi(\boldsymbol{\lambda})}{\partial \lambda_i} = g_i(\boldsymbol{x}^*(\boldsymbol{\lambda})) \qquad i = 1, ..l \qquad \text{where} \qquad \boldsymbol{x}^*(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}\in\chi} \mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) \tag{7}$$

The dual Lagrangian problem is especially convenient to use for CONLIN and MMA. See reference [1].

3 CONLIN and MMA

A truncated Taylor approximation of the objective function (Compliance C) and the volume constraint function, $g_1(\boldsymbol{x})$, are written as

$$C(\boldsymbol{x}) \approx C(\boldsymbol{x}^k) + \sum_{e=1}^{n} \left. \frac{\partial C}{\partial y_e} \right|_{\boldsymbol{x}=\boldsymbol{x}^k} (y_e - y_e^k) \tag{8}$$

In CONLIN the intervening variable $y_e(x_j)$ are defined as

$$\begin{cases} y_j = x_j & \text{if} \quad \frac{\partial C}{\partial x_j} > 0 \\ y_j = \frac{1}{x_j} & \text{if} \quad \frac{\partial C}{\partial x_j} \leq 0 \end{cases} \tag{9}$$

Where $\frac{\partial C(\boldsymbol{x})}{\partial y_j}$ becomes

$$\frac{\partial C(\boldsymbol{x})}{\partial y_j} = \frac{\partial C(\boldsymbol{x})}{\partial x_j} \frac{\partial x_j}{\partial y_j} \tag{10}$$

The derivative of the compliance C with respect to the design variable becomes

$$\frac{\partial C(\boldsymbol{x})}{\partial x_j} = -\boldsymbol{u}(\boldsymbol{x})^T \boldsymbol{k}_j^0 \boldsymbol{u}_j(\boldsymbol{x}) \tag{11}$$

Equation (11) is negative since $\boldsymbol{k}_j^0$ is positive semidefinite , in turn with a objective function $g_0^{C,k}$, i.e. the compliance, the CONLIN approximation becomes

$$g_0^{C,k}(\boldsymbol{x}) = \sum_{j=1}^{n} \frac{\partial g_0(\boldsymbol{x}^k)}{\partial x_j} \frac{x_j^k(x_j - x_j^k)}{x_j} \tag{12}$$

The intervening variable is $y_j = \frac{1}{x_j}$ from equation (9). Then, $(\boldsymbol{P})_{nf}$ becomes

$$(\boldsymbol{P})_{nf}^{C,k} = \begin{cases} \min\limits_{\boldsymbol{x}} & g_0^{C,k}(\boldsymbol{x}) \\ s.t. & \begin{cases} g_1(\boldsymbol{x}) = \sum\limits_{j=1}^{n} l_j x_j - V_{max} \leq 0 \\ \boldsymbol{x} \in \chi \end{cases} \end{cases} \tag{13}$$

$g_0^{C,k}$ is strictly convex if the strain energy of all bars is nonzero. Solving the subproblem with Lagrangian duality and evaluating the partial derivative $\frac{\partial \mathcal{L}}{\partial x_j} = 0$ provides

$$x_j^t = \sqrt{\frac{(x_j^k)^2 \boldsymbol{u}_j(\boldsymbol{x}^k)^T \boldsymbol{k}_j^0 \boldsymbol{u}_j(\boldsymbol{x}^k)}{\lambda l_j}} \qquad \text{where} \qquad x_j^t = \begin{cases} \alpha_j^k & \text{if} \quad x_j^t < \alpha_j^k \\ x_j^{max} & \text{if} \quad x_j^t > x_j^{max} \\ x_j^t & \text{otherwise} \end{cases} \tag{14}$$

Solving the dual problem, where the gradient of φ^k is the constraint condition g_1 in equation (13).

$$\frac{\partial \varphi^k(\lambda)}{\partial \lambda} = \sum_{j=1}^{n} l_j x_j^*(\lambda) - V_{max} \tag{15}$$

If the CONLIN approximation converges slowly or not at all MMA, method of moving asymptotes, is an alternative. The name explains the method quite well, the asymptotes are altered during the interactions which can improve the performance. The intervening variable $y(x_j)$ is defined as

$$\begin{cases} y_j = \frac{1}{x_j - L_j} & \text{if} \quad \frac{\partial C}{\partial x_j} \geq 0 \\ y_j = \frac{1}{U_j - x_j} & \text{if} \quad \frac{\partial C}{\partial x_j} < 0 \end{cases} \tag{16}$$

Continuing in the same manner as for CONLIN, the derivative of the compliance C for MMA becomes

$$\frac{\partial C(\boldsymbol{x})}{\partial x_j} = -\boldsymbol{u}(\boldsymbol{x})^T \boldsymbol{k}_j^0 \boldsymbol{u}_j(\boldsymbol{x}) \tag{17}$$

Equation (17) is negative since $\boldsymbol{k}_j^0$ is positive semidefinite , in turn with a objective function $g_0^{M,k}$, i.e. the compliance, the MMA approximation becomes

$$g_0^{M,k}(\boldsymbol{x}) = r_0^k + \sum_{j=1}^{n} \frac{q_{0j}^k}{x_j - L_j^k} \tag{18}$$

where q_{0j}^k and r_0^k are defined as

$$q_{0j}^k = (x_j^k - L_j^k)^2 \boldsymbol{u}_j(\boldsymbol{x}^k)^T \boldsymbol{k}_j^0 \boldsymbol{u}_j(\boldsymbol{x}^k) \tag{19}$$

$$r_0^k = g_0(\boldsymbol{x^k}) - \sum_{j=1}^{n} (x_j - L_j^k) \boldsymbol{u}_j(\boldsymbol{x}^k)^T \boldsymbol{k}_j^0 \boldsymbol{u}_j(\boldsymbol{x}^k) \tag{20}$$

The intervening variable is $y_j = \frac{1}{x_j - L_j}$ according to (16). Then, $(\boldsymbol{P})_{nf}$ becomes

$$(\boldsymbol{P})_{nf}^{M,k} = \begin{cases} \min\limits_{\boldsymbol{x}} & g_0^{M,k}(\boldsymbol{x}) \\ s.t. & \begin{cases} g_1(\boldsymbol{x}) = \sum\limits_{j=1}^{n} l_j x_j - V_{max} \leq 0 \\ \alpha_j^k \leq x_j \leq x_j^{max}, j = 1, .., n \end{cases} \end{cases} \tag{21}$$

The move limit α_j^k is implemented as $\alpha_j^k = max(x_j^{min}, L_j^k + \mu(x_j^k - L_j^k))$. Note that μ is restricted to $\mu \in (0,1)$. $g_0^{M,k}$ is strictly convex if the strain energy of all bars is nonzero. The asymptote L_j is for the first two iterations calculated with

$$L_j = x_j - s_{init}(x_j^{max} - x_j^{min}) \tag{22}$$

A typical value for the variable $s_{init} = 0.1$, however this is based on the structural optimization problem. For the following iterations when calculating L_j the product of $Z = (x_j^k - x_j^{k-1})(x_j^{k-1} - x_j^{k-2})$ are studied and L_j is calculated with

$$\begin{cases} L_j^k = x_j^k - s_{slower}(x_j^{k-1} - L_j^{k-1}) & \text{if} \quad Z \leq 0 \\ L_j^k = x_j^k - s_{faster}(x_j^{k-1} - L_j^{k-1}) & \text{if} \quad Z > 0 \end{cases} \tag{23}$$

Typical values for $s_{slower} = 0.6$ and $s_{faster} = 1.1$ however this is as well based on the structural problem and could be changed. If L_j becomes negative it should be set to $L_j = 0$. Solving the subproblem with Lagrangian duality and evaluating the partial derivative $\frac{\partial \mathcal{L}}{\partial x_j} = 0$ provides

$$x_j^t = L_j^k + \sqrt{\frac{q_{0j}^k}{\lambda l_j}} \qquad \text{where} \qquad x_e^*(\lambda) = \begin{cases} \alpha_j^k & \text{if} \quad x_j^t < \alpha_j^k \\ x_j^{max} & \text{if} \quad x_j^t > x_j^{max} \\ x_j^t & \text{otherwise} \end{cases} \tag{24}$$

Solving the dual problem, where the gradient of φ^k is the constraint condition g_1.

$$\frac{\partial \varphi^k(\lambda)}{\partial \lambda} = \sum_{j=1}^{n} l_j x_j^*(\lambda) - V_{max} \tag{25}$$

Reference for theory regarding MMA and CONLIN approximations is found in [1].

4 Solid Isotropic Material Penalization, (SIMP)

The same method as for CONLIN and MMA are used. A truncated Taylor approximation of the objective function provides the following relation

$$C(\boldsymbol{x}) \approx C(\boldsymbol{x}^k) + \sum_{e=1}^{n} \frac{\partial C}{\partial y_e}\bigg|_{\boldsymbol{x}=\boldsymbol{x}^k} (y_e - y_e^k) \tag{26}$$

The intervening variable is $y_e = x_e^{-\alpha}$. Where $\alpha > 0$, and $\alpha = 1$ represents the CONLIN linearizion. Using this, the derivative of the compliance becomes

$$\frac{\partial C(\boldsymbol{x})}{\partial x_e} = -(\boldsymbol{u}_e^k)^T \boldsymbol{k}_e^0 \boldsymbol{u}_e^0 \quad at\, \boldsymbol{x} = \boldsymbol{x}^k \tag{27}$$

where

$$\boldsymbol{u}(\boldsymbol{x}^k) = \boldsymbol{K}(\boldsymbol{x}^k)^{-1}\boldsymbol{F} \tag{28}$$

Constitutive matrix in SIMP uses Hooke's law with penalization implemented.

$$\boldsymbol{D} = \frac{\rho^p E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{(1-\nu)}{2} \end{bmatrix} \tag{29}$$

A typical value of the constant parameter p is $p = 3$ and $\rho^p E$ is the effective Young's modulus. In the SIMP method $\underline{\rho} \approx 0$ and $\overline{\rho} = 1$ is set in equation (33). The result is illustrated in figure 2 for different values of p. The optimal solution will provide a almost zero to one solution of the problem which practically means areas of holes where $\rho = 0$ and regions where Young's modulus is E where $\rho = 1$.

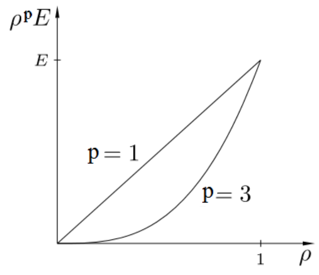


Figure 2: Youngs modulus as a function of ρ for different values of p.

The global stiffness matrix in equation (28) is calculated with equation (29) and becomes

$$\boldsymbol{K}(\boldsymbol{x}) = \sum_{e=1}^{n} x_e^p \boldsymbol{K}_e^0 \tag{30}$$

Evaluation of $\frac{\partial C}{\partial y_e}$ with equation (27) and (26) provides the subproblem:

$$\boldsymbol{P} = \begin{cases} \min\limits_{\boldsymbol{x}} \sum\limits_{e=1}^{n} b_e^k x_e^{-\alpha} \\ s.t \qquad \begin{cases} \boldsymbol{x}^T \boldsymbol{a} = V \\ \underline{\rho} \le x_e \le \overline{\rho}, \quad e = 1, ..., n \end{cases} \end{cases} \tag{31}$$

where

$$b_e^k = \frac{(x_e^k)^{1+\alpha}}{\alpha}((\boldsymbol{u}_e^k)^T p(x_e^k)^{p-1} \boldsymbol{k}_e^0 \boldsymbol{u}_e^k) \tag{32}$$

$\boldsymbol{P}$ is a convex problem and using Lagrangian duality one arrives after finding a stationary point of $\frac{\partial \varphi_e}{x_e}$ at

$$x_e = \left(\frac{\alpha b_e^k}{\lambda a_e}\right)^{\frac{1}{1+\alpha}} \qquad \text{where} \qquad x_e(\lambda) = \begin{cases} \underline{\rho} & \text{if } \left(\frac{\alpha b_e^k}{\lambda a_e}\right)^{\frac{1}{1+\alpha}} < \underline{\rho} \\ \left(\frac{\alpha b_e^k}{\lambda a_e}\right)^{\frac{1}{1+\alpha}} & \text{if } \underline{\rho} \le \left(\frac{\alpha b_e^k}{\lambda a_e}\right)^{\frac{1}{1+\alpha}} \le \overline{\rho} \\ \overline{\rho} & \text{if } \left(\frac{\alpha b_e^k}{\lambda a_e}\right)^{\frac{1}{1+\alpha}} > \overline{\rho} \end{cases} \tag{33}$$

Investigation of stationary point of $\varphi(\lambda)$ provides

$$\frac{\partial \varphi(\lambda)}{\partial \lambda} = \sum_{e=1}^{n} a_e x_e(\lambda) - V = 0 \tag{34}$$

Equation (34) is the volume constraint. Optimization iteration scheme and pseudo code is found in appendix, subsection 7.1. Reference for theory regarding the SIMP is found in [1].

5 Filters

To ensure a more mesh independent solution a filter may be added to the structural optimization algortihms. A density filter and the Helmholtz' PDE based filter will be introduced and further discussed below.

5.1 Density filter

Computing neighbor elements to an element $\boldsymbol{x}_e$ with a prescribed filter radius R and where the neighbor elements are defined as $\boldsymbol{x}_i$, results in the following relation

$$N_e = \{i| \, ||\boldsymbol{x}_i - \boldsymbol{x}_{\boldsymbol{e}}|| \le R\} \tag{35}$$

Filtered density is calculated by

$$\widetilde{\rho}_e = \frac{\sum\limits_{i \in N_e} w(\boldsymbol{x}_i) v_i \rho_i}{\sum\limits_{i \in N_e} w(\boldsymbol{x}_i) v_i} \tag{36}$$

v_i is the volume. A constant weighting function is represented to the left in equation (37) and a cone-shaped weighting function is represented to the right.

$$w(\boldsymbol{x}_i) = 1 \qquad , w(\boldsymbol{x}_i) = R - \|\boldsymbol{x}_i - \boldsymbol{x}_e\| \tag{37}$$

The sensitivity of the objective function with respect to the design variables and use of the chain rule becomes

$$\frac{\partial C}{\partial \rho_e} = -p\widetilde{\rho}_e^{p-1}\boldsymbol{u}_e^T\boldsymbol{k}_e^0\boldsymbol{u}_e\frac{w(\boldsymbol{x}_e)v_e}{\sum\limits_{j\in N_i} w(\boldsymbol{x}_j)v_j} \tag{38}$$

i in equation (38) represents neighbor elements. Equation (32) is instead computed as

$$b_e^k = \frac{(x_e^k)^{1+\alpha}}{\alpha}((\boldsymbol{u}_e^k)^T p(\widetilde{\rho}_e^k)^{p-1}\boldsymbol{k}_e^0\boldsymbol{u}_e^k)\frac{w(\boldsymbol{x}_e)v_e}{\sum\limits_{j\in N_i} w(\boldsymbol{x}_j)v_j} \tag{39}$$

Equations and information used regarding the density filter are acquired from reference [3].

A control of the implemented algorithm can be performed by calculating $C_1 = F^T * a(\rho)$ and $C_2 = F^T * a(\rho + e_e\epsilon)$ for a chosen element. The derivative of the compliance for the same element should be $\frac{\partial C}{\partial \rho_e} = \frac{C_2 - C_1}{\epsilon}$. Where e_e is a basis vector and ϵ is a small value.

5.2 Helmholtz' PDE based filter

Helmholtz equation is defined as

$$\nabla^T\boldsymbol{K}_d\nabla\widetilde{\rho} + \widetilde{\rho} = \rho \qquad \text{where} \qquad \boldsymbol{K}_d = \sum_{i=1}^{d} r_i^2\boldsymbol{v}_i\boldsymbol{v}_i^T \tag{40}$$

Note that the vector $\boldsymbol{v}_i$ is represented as the direction of the length scale r_i in equation (40) and the number of dimensions are defined as d. If r_i has different values for different i, then anisotropy is introduced. The implemented filter regarding the assignment is isotropic, i.e. $r_i = r_{i+1} = r_{i+d}$ The optimization problem, equation (40) is solved using FEM. In turn the information about neighbor cells is not required in the PDE filter compared to the Density filter. This is because the PDE filter has properties which are parallel to the implementation of FEM.

$$\widetilde{\rho}_e = \boldsymbol{N}_e\widetilde{\boldsymbol{\rho}}_e \tag{41}$$

$\widetilde{\rho}_e$ is the filtered density per element while $\widetilde{\boldsymbol{\rho}}_e$ is a vector with the nodal values for element e. $\boldsymbol{N}_e$ consists of finite element interpolation functions. The filtered field is represented by equation (42) and is obtained by using Green's formula, multiplying the weight functions to the PDE filter and integrating over the design domain.

$$\sum_{i\in N_e}\int_{\Omega_i}[\nabla\boldsymbol{N}_e^T\boldsymbol{K}_d\nabla\boldsymbol{N}_e + \boldsymbol{N}_e^T\boldsymbol{N}_e]d\Omega\widetilde{\boldsymbol{\rho}} = \sum_{i\in N_e}\rho_i\int_{\Omega_e}\boldsymbol{N}_e^T d\Omega \tag{42}$$

Note that the stiffness matrix $\boldsymbol{K}_f$ is positive definite. Rewriting this with $\boldsymbol{K} = \int_{\Omega_i} \nabla\boldsymbol{N}_e^T\boldsymbol{K}_d\nabla\boldsymbol{N}_e d\Omega$, $\boldsymbol{M} = \int_{\Omega_i}\boldsymbol{N}_e^T\boldsymbol{N}_e d\Omega$ and $\boldsymbol{T} = \int_{\Omega_e}\boldsymbol{N}_e^T d\Omega$ becomes

$$(\boldsymbol{K} + \boldsymbol{M})\widetilde{\boldsymbol{\rho}} = \boldsymbol{T}\boldsymbol{\rho} \tag{43}$$

By utilizing the adjoint method the sensitivity of the objective function, i.e. the compliance, is represented as

$$\frac{\partial C}{\partial \rho_e} = \frac{\partial C}{\partial \widetilde{\rho}_i}\frac{\partial \widetilde{\rho}_i}{\partial \rho_e} \tag{44}$$

Where

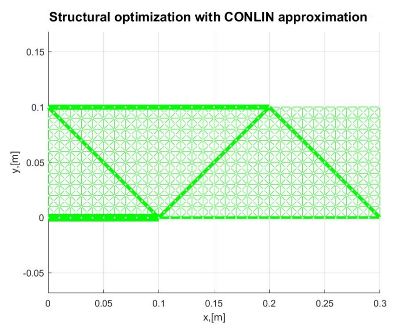
$$\frac{\partial C}{\partial \widetilde{\rho}_i} = \sum_{i\in N_e} -p\int_{\Omega_i}\boldsymbol{N}_e\widetilde{\rho}_i^{p-1}\boldsymbol{u}_e^T\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\boldsymbol{u}_e d\Omega \quad ,\boldsymbol{B} = \widetilde{\nabla}\boldsymbol{N} \qquad \text{and} \qquad \frac{\partial \widetilde{\rho}_i}{\partial \rho_e} = (\boldsymbol{K} + \boldsymbol{M})^{-1}\boldsymbol{T} \tag{45}$$

$\boldsymbol{T}$ is a *[nnod x nelm]* matrix and i is the degree of freedom for one element. $\widetilde{\nabla}$ is the linear elasticity operator. $\frac{\partial \widetilde{\rho}_i}{\partial \rho_e}$ = *constant* and is calculated only one time regarding the optimization loop. Equations and information used regarding the PDE filter are acquired from reference [2].

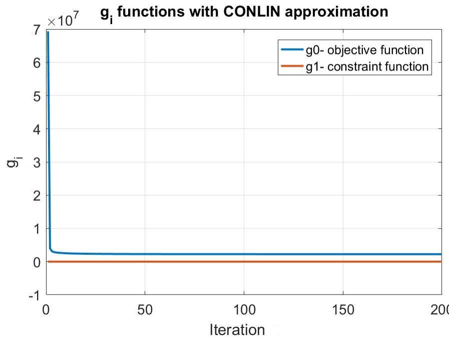
6 Result and discussion

6.1 CONLIN and MMA

Represented in figure 3a is the structural optimization performed with CONLIN. In figure 3b the objective function and constraint function is observed to alter with increasing iterations. Only one result regarding the actual structure as observed in figure 3a is included in the result section since similar results were obtained with different values of $Area_{min}$ and $Area_{initial}$. The $Area$ parameter is used instead of diameter d and is defined as the cross sectional area of the bars. Note that a Young's modulus $E = 1Pa$ is used regarding the CONLIN and MMA calculations.



(a)

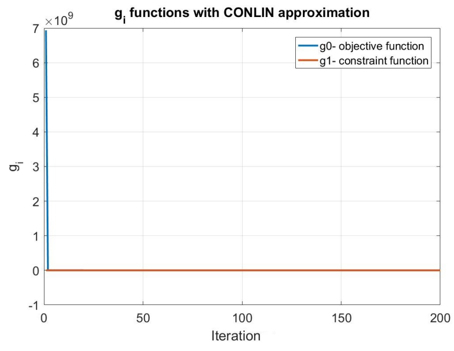


(b)

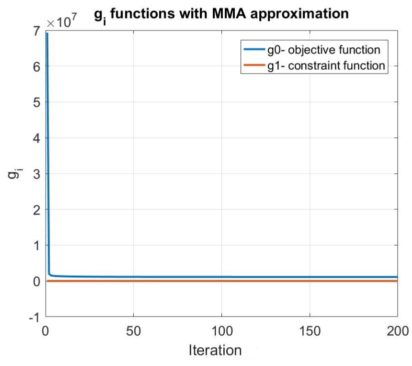
Figure 3: (a) Represents the structure after the structural optimization with CONLIN approximation, where $Area_{min} = 3.14 * 10^{-14}$ and $Area_{initial} = 10^{-8}$. (b) Represents the corresponding objective function and constraint function.

.

The most interesting aspect is the similarity between CONLIN and MMA with identical initial values regarding $Area_{min}$ and $Area_{initial}$, which is observed by comparing figure 3b and 4b. Comparing the time it takes to converge with a tolerance (tolerance equals the change from an optimized area from the previous iteration to the current iteration) of 10^{-9} between CONLIN and MMA with identical initial values presents a time difference of 16.7 seconds, where MMA is less time consuming. Although solving the equilibrium equation (28) is the most time consuming part. Therefore, MMA is more preferable to use regarding the performed optimization. Observed in figure 4a is the result when altering the initial area to $Area_{min} = 10^{-10}$. An altering of the parameter $Area_{min}$ in the CONLIN approximation provides a very similar behavior regarding structural visualization, objective function and constraint function results when comparing to the provided results for CONLIN. Therefore the result is not provided in this report. As some of the cross-sectional areas of the elements equals either $Area_{min}$ or $Area_{max}$ the stresses will vary in the trusses and in turn not result in a fully stressed design.



(a)



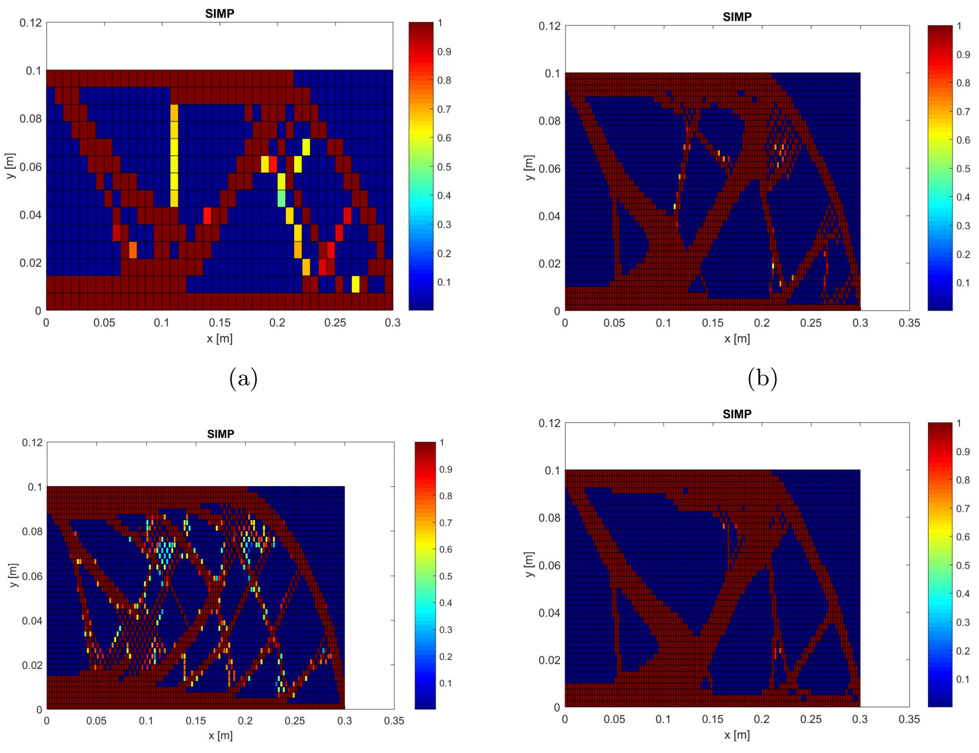
(b)

Figure 4: (a) Represents the objective function and constraint function for CONLIN where, $Area_{min} = 3.14*10^{-14}$ and $Area_{initial} = 10^{-10}$. (b) Represents the objective function and constraint function for MMA, where $Area_{min} = 3.14 * 10^{-14}$ and $Area_{initial} = 10^{-8}$

.

6.2 SIMP

The colorbar represents the distribution of density in the result section for SIMP, density filter and PDE filter, i.e. red/brown color corresponds to high density elements while blue color corresponds to low density elements. Also a Young's modulus of $E = 200 * 10^9 Pa$ is used regarding the SIMP result and the filters. Initial parameters used regarding the SIMP method are $\rho_{min} = 10^{-3}$ and $\rho_{initial} = 10^{-3}$. The SIMP method provides an almost zero to one solution. A significant disadvantage using the SIMP method is the numerical difficulties that may occur. The main concern is the mesh-dependency which means if the resolution of a design is altered. For instance with a finer mesh, and optimized again the result could provide another design comparing with a coarser mesh. Figure 5a and figure 5b illustrates a mesh dependent algorithm where all parameters are the same but figure 5b consists of a finer mesh than figure 5a. The two results are similar but not entirely similar. What could also be observed are so called checkerboarders which is represented as designs with alternating solid and void cells ordered in checkerboarder-like patterns. These checkerboarders are usually not acceptable and they exist due to bad numerical modeling which overestimates the stiffness of the checkerboarders.



(a) (b)

(c) (d)

Figure 5: The optimization problem is solved using the SIMP method. (a) A coarse mesh with $p = 3$, $\alpha = 3$. (b) A fine mesh with $p = 3$, $\alpha = 3$. (c) A fine mesh with $p = 2$, $\alpha = 3$. (d) A fine mesh with $p = 3$, $\alpha = 3$.

.

One more difficulty using the SIMP-method is the parameter p can make the convex problem of the thickness of the sheet into a nonconvex problem. This can result in where the algorithm terminates to different optimas for different starting values. To solve this the algorithm can be exectued several times for different starting values or increasing the value of p gradually from one which provides a convex solution to higher which will provide a more zero to one solution. Figure 5c and figure 5b illustrates this phenomenon. A higher p-value provides a more zero to one solution than with a lower p-value as there is less "half values" for a higher p-value. The exponent in equation (33) is called the damping factor, η. The name is derived from where elements with high strain energy is expected to have low stiffness and this leads to that they become thicker. When η is less than unity this modification is damped, thus the name damping factor. $\alpha = 1$ corresponds to the CONLIN linearization, illustrated in figure 5d. When comparing different values of α, a $\alpha = 3$ yields more checkerboarder structure than $\alpha = 1$.

6.3 Density filter

Using the SIMP method with the results in figure 5 as discussed above some difficulties have been noticed, such as checkerboarders and mesh dependency. What should be noted by investigating the filtered result is that the filtered densities, $\widetilde{\rho}$, are plotted in the following figures. The density filter used in figure 6 and figure 7 improves the result from the SIMP algorithm by removing checkerboarders and mesh dependency. This is obtained with no use of extra constraints, a zero to one solution, stable and fast convergence and a relatively easy implementation. Density filtering makes use of preserving the volume, this means that the volume of material should be the same before and after being filtered. This would have been necessary if the structure would contain details smaller than the neighborhood area. Calculating this gave a sufficiently close result for the filtered and non-filtered elements. Initial parameters used regarding the density filter are $\rho_{min} = 10^{-3}$, $\rho_{initial} = 10^{-1}$ and $\alpha = 3$.

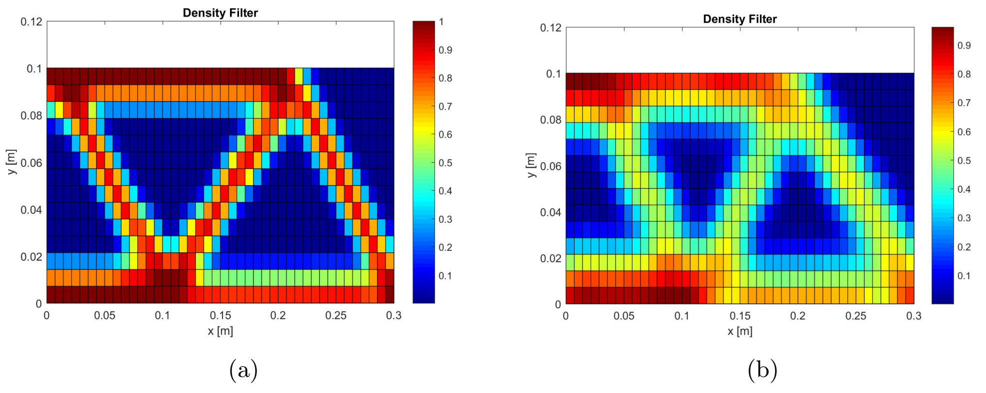


Figure 6: The optimization problem is solved using the SIMP method with a density filter. (a) Coarse mesh at a length scale $r = 2*$ element side, where one element side regarding the coarse mesh is ≈ 0.0071 (b) Coarse mesh at a length scale of $r = 3.5*$element side is used.

What can be noticed comparing figure 6a with figure 7a is some kind of mesh dependency. However what is meant with mesh dependency is described as where the details are defined by the radius used to define the neighbors. The "bridges" present for the fine mesh in figure 7a can easily be removed by increasing the radius and this is performed in figure 7b. However using the same length scale, $r = 3.5*$element side, for the coarse mesh will filter the structure too much which will result in a blurry image, this is illustrated in figure 6b. This is due to the density being defined as a weighted average of the densities in a mesh independent neighborhood which in turn are defined by the radius used. The effect of the weighting is highest for a constant weighting and less for the linear weighting (cone-shaped). However the constant weighting function does result in a more computationally efficient algorithm.

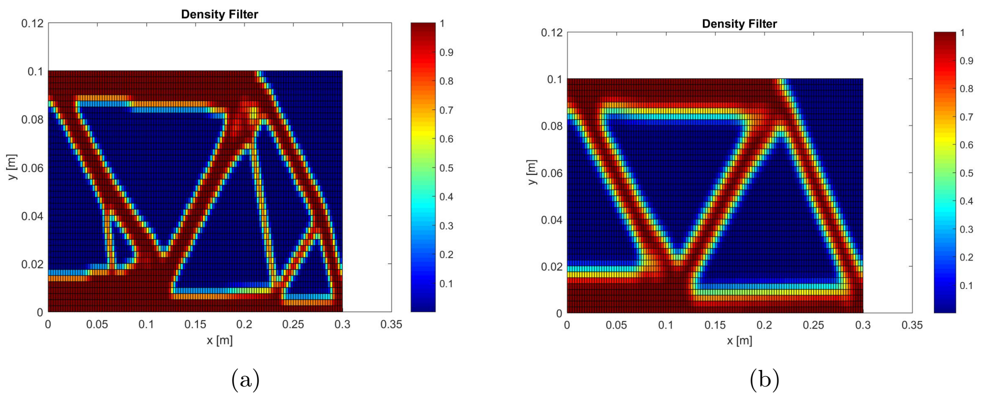
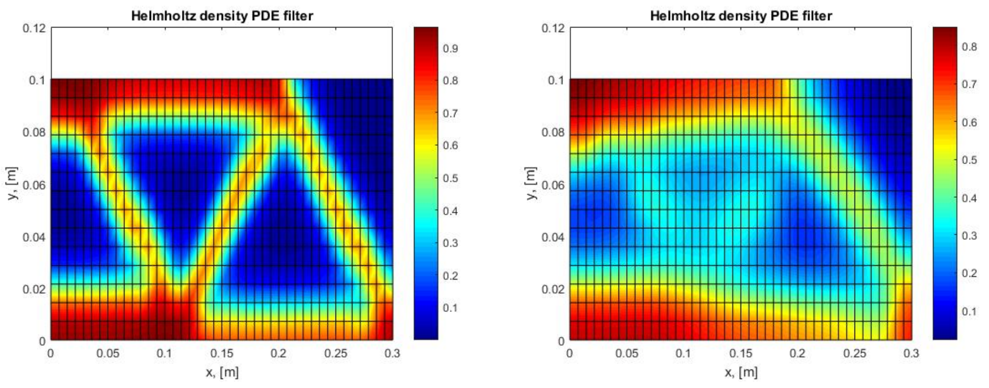


Figure 7: The optimization problem is solved using the SIMP method with a density filter. (a) Fine mesh at a length scale $r = 2*$ element side, where one element side regarding the fine mesh is ≈ 0.0025 (b) Fine mesh at a length scale of $r = 3.5*$element side is used.

6.4 PDE filter

The PDE filter is applied as a density filter according to [2]. The main difference with the PDE method is to avoid large matrices storing neighborhood elements. As well as for the density filter the PDE filter is volume preserving. Initial parameters used regarding the PDE filter are $\rho_{min} = 10^{-5}$ (lower bound), $\alpha = 2$ and $\rho_{initial} = 10^{-3}$.

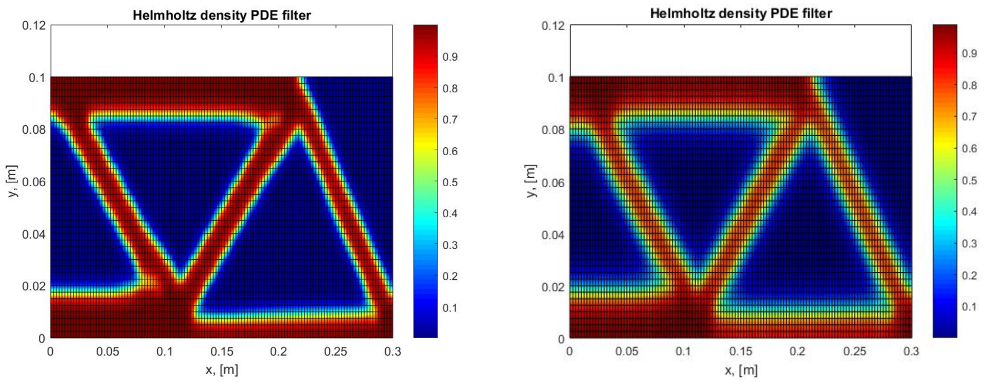


(a) (b)

Figure 8: (a) The use of PDE filter for a coarse mesh at a length scale $r = 1*$ element side, where one element side regarding the coarse mesh is ≈ 0.0071. At figure (b) a length scale of $r = 2*$element side is used. Both figure (a) and (b) has the same tolerance regarding convergence.

.

In figure 8a the use of a less sizable length scale r is used compared to in figure 8b regarding the coarse mesh. In a similar manner, the length scale r less sizable in figure 9a than in figure 9b. Comparing the result in figure 8a and figure 8b represents the importance to designate a not to great r-value when applying a PDE filter to the structural optimization. Where a too big r-value provides a very blurry result of the density distribution, this phenomenon is depicted in figure 8b. Comparing figure 9a and 9b depicts the difference of different values for r, although with a finer mesh it is not as evident that figure 9b is more blurry. It might be important to further investigate the r-value more. If the r-value is chosen as to small, indicating a smaller surrounding of the element thus closer to the boarders of the element,it might be to small and switch over to the negative side. If it becomes to small the derivative of the compliance with respect to the design variable, ρ, becomes positive and the algorithm will be incorrect. By examine the sign of the derivative of the compliance with respect to the design variable this may be detected.



(a) (b)

Figure 9: (a) The use of PDE filter for a fine mesh at a length scale $r = 1*$ element side, where one element side regarding the fine mesh is ≈ 0.0071. At figure (b) a length scale of $r = 2*$element side is used. Both figure (a) and (b) has the same tolerance regarding convergence.

.

In order to speed up the algorithms sparse-matrices and scalar products have been used. However there could be more done in order to optimize the algorithm. This is mainly necessary in the filtering algorithms which for the fine mesh was very time consuming.

References

[1] Peter W. Christensen Anders Klarbring. *An Introduction to Structural Optimization*, volume 153. 2009.

[2] B. S. Lazarov and O. Sigmund. Filters in topology optimization based on helmholtz-type differential equations. *International Journal for Numerical Methods in Engineering*, 86(6):765–781, 2011.

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

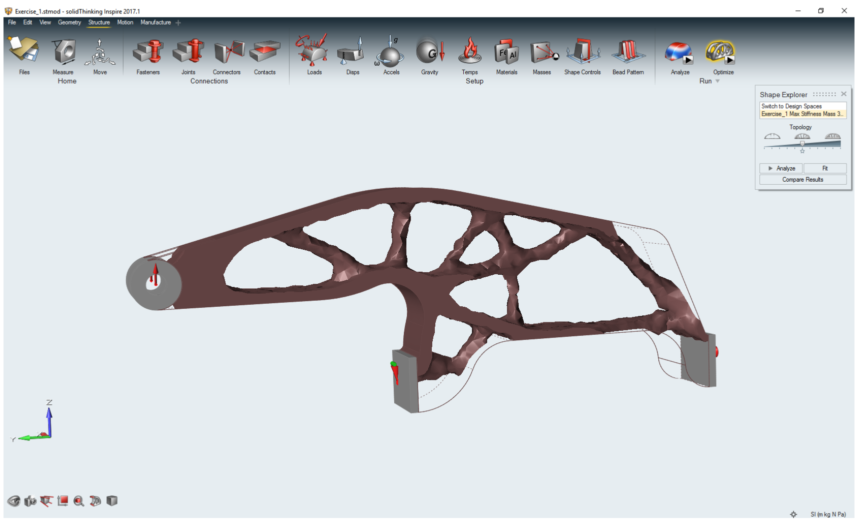
7 Appendix

7.1 Pseudo code

```
Optimization procedure
- Initiation of quantities
• Iteration k = 0, 1 ,2 ... until the change is smaller then a set tolerance, i.e $change = norm(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k)$
    - Calculate the tangent stiffness matrix $\boldsymbol{D}$ with the CALFEM function hooke.
    - Calculate the element stiffness matrix $\boldsymbol{K}$, with e.g. CALFEM function plani4e
      for a four node element.
    - Calculate the displacement increment $\boldsymbol{u}$ with CALFEM function solveq.
    - Extract displacements with CALFEM function extract.
    - Compute sensitivities of the compliance with respect to the design variable per element $x_e$, $\frac{\partial C}{\partial x_e}$.
    - Solve the dual problem by computing $\frac{\partial \varphi(\lambda)}{\partial \lambda}$ and obtain an optimal value $\lambda^*$.
      Subroutine texttttdphidlambda can be used.
    - Compute an optimal design variable per element j, $x_j^*(\lambda^*)$. $\boldsymbol{x}^* = \boldsymbol{x}^{k+1}$ .
      Subroutine textttgetAstar can be used.
    - Compute stresses if needed with e.g. CALFEM function bar2s for a bar element.
    - Check the change between current and previous iteration design
      variables and compare with tolerance.
• End iteration loop
- Accept quantities
• End load step loop
```

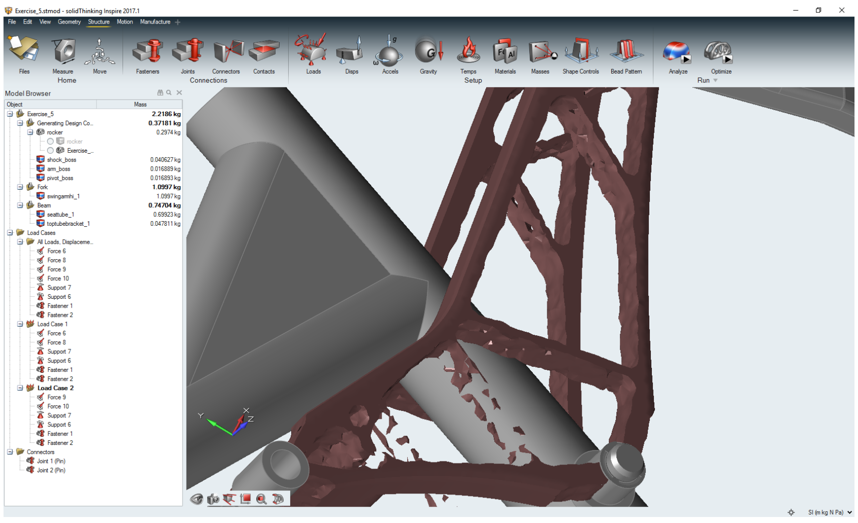
Table 1: Optimization iteration scheme. Depending on what filter and optimization algorithm used, the pseudo code presented may be altered to the desired algorithm provided in the report.

7.2 Inspire


Exercise_1.stmod - solidThinking Inspire 2017.1
File Edit View Geometry Structure Motion Manufacture
Files Measure Move Home
Fasteners Joints Connectors Contacts Connections
Loads Disps Accels Gravity Temps Materials Masses Shape Controls Bead Pattern Setup
Analyze Optimize Run
Topology
Analyze Fit
Compare Results
SI (m kg N Pa)

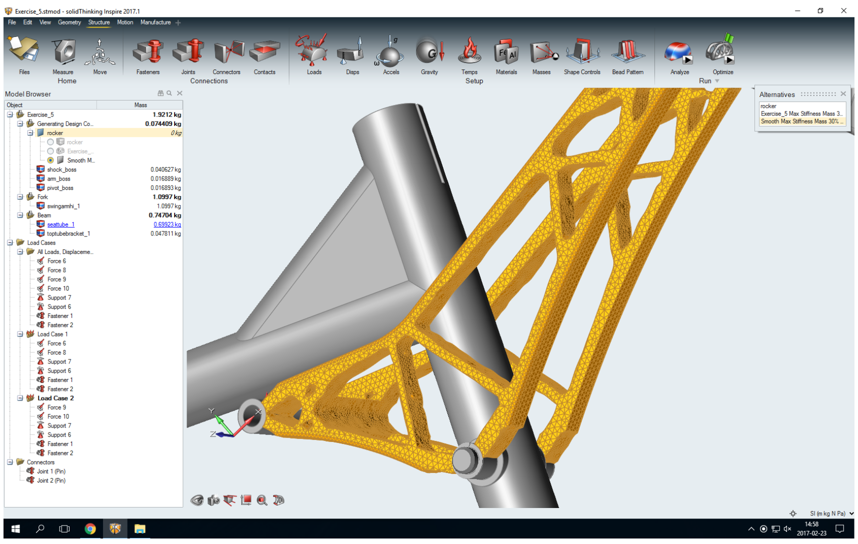
Shape Explorer

Switch to Design Spaces
Exercise_1 Max Stiffness Mass 3...


Exercise_5.stmod - solidThinking Inspire 2017.1
File Edit View Geometry Structure Motion Manufacture
Files Measure Move Home
Fasteners Joints Connectors Contacts Connections
Loads Disps Accels Gravity Temps Materials Masses Shape Controls Bead Pattern Setup
Analyze Optimize Run
SI (m kg N Pa)

Model Browser

```
Object                              Mass
Exercise_5                          2.2186 kg
  Generating Design Co...           0.37181 kg
    rocker                          0.2974 kg
      rocker
      Exercise_...
    shock_boss                      0.040627 kg
    arm_boss                        0.016889 kg
    pivot_boss                      0.016893 kg
  Fork                              1.0997 kg
    swingarmhi_1                    1.0997 kg
  Beam                              0.74704 kg
    seattube_1                      0.69923 kg
    toptubebracket_1                0.047811 kg
Load Cases
  All Loads, Displaceme...
    Force 6
    Force 8
    Force 9
    Force 10
    Support 7
    Support 6
    Fastener 1
    Fastener 2
  Load Case 1
    Force 6
    Force 8
    Support 7
    Support 6
    Fastener 1
    Fastener 2
  Load Case 2
    Force 9
    Force 10
    Support 7
    Support 6
    Fastener 1
    Fastener 2
Connectors
  Joint 1 (Pin)
  Joint 2 (Pin)
```


Exercise_5.stmod - solidThinking Inspire 2017.1
File Edit View Geometry Structure Motion Manufacture
Files Measure Move Home
Fasteners Joints Connectors Contacts Connections
Loads Disps Accels Gravity Temps Materials Masses Shape Controls Bead Pattern Setup
Analyze Optimize Run
SI (m kg N Pa)
14:58
2017-02-23

Model Browser

```
Object                              Mass
Exercise_5                          1.9212 kg
  Generating Design Co...           0.074409 kg
    rocker                          0 kg
      rocker
      Exercise_...
      Smooth M...
    shock_boss                      0.040627 kg
    arm_boss                        0.016889 kg
    pivot_boss                      0.016893 kg
  Fork                              1.0997 kg
    swingarmhi_1                    1.0997 kg
  Beam                              0.74704 kg
    seattube_1                      0.69923 kg
    toptubebracket_1                0.047811 kg
Load Cases
  All Loads, Displaceme...
    Force 6
    Force 8
    Force 9
    Force 10
    Support 7
    Support 6
    Fastener 1
    Fastener 2
  Load Case 1
    Force 6
    Force 8
    Support 7
    Support 6
    Fastener 1
    Fastener 2
  Load Case 2
    Force 9
    Force 10
    Support 7
    Support 6
    Fastener 1
    Fastener 2
Connectors
  Joint 1 (Pin)
  Joint 2 (Pin)
```

Alternatives

rocker
Exercise_5 Max Stiffness Mass 3...
Smooth Max Stiffness Mass 30% ...

7.3 Software

CONLIN

Main

```
clc
clear all
close all
format long

load geomSO.mat

%Material parameters
E = 1;       %Young's modulus
Ae_max = (((20*10^-3)^2)*pi)/4;          %Maximum cross sectional area
Ae_min = (((20*10^-8)^2)*pi)/4;          %Minimum cross sectional area

V_max = (2*10^-6)/2; % Maximum volume allowed


 %Number of elements
 nelm = max(edof(:,1));

 %Initiate
 K = sparse(ndof,ndof);             %Introduce stiffness matrix, Ke [8x8]
  A_global = ones(nelm,1)*1*10^-10; %Introduce element Area vector, with all Area being

%Initate Global cell for normalized stiffness matrix
 Cell_K0e = cell(nelm,1);

     for n=1:nelm


      %Initate constant norm element stiffness matrix
      Cell_K0e{n} = bar2e(ex(n,:),ey(n,:),[E 1]);

      %Obtain a global length vector for each element.
        ec =[ex(n,:),ey(n,:)];

      x_0=[ec(1,2) - ec(1,1);
           ec(1,4) - ec(1,3)];

        l_0 = sqrt(x_0'*x_0);


      %Global length vector
      l_0_global(n,:) = l_0;

     end


 V = A_global'*l_0_global;

 %Initate Tolerance
 Tol = 10^-9;

 %Initate quantaties
 iter = 0;
 Nres = 0;
 q0 = 0;

 %Input min and max values for fzero function.
 lambda_min=10^10;
 lambda_max=10^15;


%Optimization loop
 while Nres > Tol || iter == 0

     iter = iter + 1;
     disp(['Iteration ' ,num2str(iter),'-----------------------------'])


     %Reset global stiffness matrix
      K=sparse(ndof,ndof);

     %Element loop to assemble global stiffness matrix
       for i=1:nelm

           ep = [E A_global(i,:)];

           Ke = bar2e(ex(i,:),ey(i,:),ep);

          %Assemble element matrices to global matrices:
              indx = edof(i,2:end);
              K(indx,indx) = K(indx,indx)+Ke;
       end

       %Obtain global displacements
           a = solveq(K,f,bc);
           ed = extract(edof,a);

     for j = 1:nelm

       %Compute q0j according to eq 5.37, applied for CONLIN approx
       q0e(j,:) = (ed(j,:)*Cell_K0e{j}*ed(j,:)'*(A_global(j,:)^2));

     end


         %Solve the dual function
         lambdastar = fzero(@(lambda) dphidlambda...
         ( lambda,l_0_global,V_max,Ae_min, Ae_max,nelm,q0e )...
          ,[lambda_min lambda_max]);


 for z=1:nelm

        %Find new rho value per element based on new lambda from the dual
        %function
         Ae_star(z,:)= getAstar( lambdastar , Ae_min, Ae_max,...
                                l_0_global(z,:),q0e(z,:) );

        %Compute element forces
         stress(z,:) = bar2s(ex(z,:),ey(z,:),[E Ae_star(z,:)],ed(z,:))/Ae_star(z,:);

 end

      K=sparse(ndof,ndof);
      %Element loop to assemble global stiffness matrix
       for i=1:nelm

           ep = [E A_global(i,:)];
```

```
            Ke = bar2e(ex(i,:),ey(i,:),ep);


          %Assemble element matrices to global matrices:
              indx = edof(i,2:end);
              K(indx,indx) = K(indx,indx)+Ke;
       end

       %Obtain displacement
            a = solveq(K,f,bc);
            ed = extract(edof,a);

 for z=1:nelm

     %Compute element stress
     stress(z,:) = bar2s(ex(z,:),ey(z,:),[E Ae_star(z,:)],ed(z,:))/Ae_star(z,:);

 end

    %Compute the change between current and previous iteration
    %regarding the updated area.
     Nres = norm(Ae_star-A_global);
     A_global = Ae_star;

     %Plot vectors
     Ae_star_plot(:,iter) = Ae_star;
     Iter_plot(:,iter) = iter;
     g0(:,iter) = f'*a;
     stress_global(:,iter) = stress;

     g1(:,iter) = A_global'*l_0_global-V_max;

     V = A_global'*l_0_global;

     fprintf('Iter %i Res %e\n',iter,Nres)

     end

%%

figure
fac = 4*10^6;      %Magnification of Cross Section area.
magnfac = 1*10^-10; %Magnification of Deformed structure.
myeldisp2(ex,ey,ed,[1 2 0],magnfac,A_global,fac);
grid on
xlabel('x,[m]')
ylabel('y,[m]')
title(['Structural optimization with CONLIN approximation'],'FontSize', 14)

figure
plot(Iter_plot,g0,'LineWidth',2)
hold on

plot(Iter_plot,g1,'LineWidth',2)
grid on
xlabel('Iterationer','FontSize', 14)
ylabel('g_i','FontSize', 14)
legend('g0- objective function','g1- constraint function')
title('g_i functions with CONLIN approximation','FontSize', 14)
set(gca,'FontSize',14)
```

```
%Check Area fulfills Box constraint

A_max_check = max(A_global);
A_min_check = min(A_global);

    if A_max_check <= Ae_max
        disp('Valid Box constraint regarding A_max')
    else
        disp('Box constraint not fulfilled ')
    end


     if A_min_check >= Ae_min
        disp('Valid Box constraint regarding A_min')
        else
        disp('Box constraint not fulfilled ')
     end

  %Check procentage of elements in A_global which have the values Ae_min
  %and Ae_max.
  procent_min=length(find(A_global == Ae_min))/nelm
  procent_max=length(find(A_global == Ae_max))/nelm
```

Dphidlambda

```
function [ dphidlambda ] = dphidlambda( lambda,l_0_global,Vmax,Ae_min, Ae_max,nelm,q0e
%Compute dphidlambda, solving the dual function.
%
%Input: Lambda
%        l_0 , Global length vector
%        Vmax, Maximum volume
%        Ae_min, Minimum area
%        Ae_max, Maximum area
%        q0e   , qo array, computed in the main m-file.

%Output: dphidlambda



     h = 0;
    for z=1:nelm
        Ae_star(z,:) = getAstar( lambda , Ae_min, Ae_max,l_0_global(z,:),q0e(z,:) );

        he = l_0_global(z,:)*Ae_star(z,:);
        h = he + h;
    end

%Constraint function
dphidlambda = h-Vmax;

end
```

GetAstar

```
function [ Ae_star ] = getAstar( lambda, Ae_min, Ae_max,l_0,q0e )
%Calculate A* for a given lambda.

%Input: lambda
%        Ae_min, Minimum area
%        Ae_max, Maximum area
%        l_0 , Global length vector
%        q0e   , qo(e) (per element), computed in the main m-file.
```

```
%Output: A*

 Ae_trial = sqrt(q0e/(lambda*l_0));


    if   Ae_trial < Ae_min;


            Ae_star = Ae_min;

    elseif  Ae_trial > Ae_max



          Ae_star = Ae_max;
    else


          Ae_star = Ae_trial;
    end



end
```

MMA

Main

```
clc
clear all
close all
format long

load geomSO.mat

%Material parameters
E = 1;        %Young's modulus
Ae_max = (((20*10^-3)^2)*pi)/4;      %Maximum cross sectional area
Ae_min = (((20*10^-8)^2)*pi)/4;     %Minimum cross sectional area
V_max = 2*10^-6;                     %Maximum allowed volume


%Number of elements
nelm = max(edof(:,1));

%Initiate
K = sparse(ndof,ndof);
A_global = ones(nelm,1)*10^-8; %Introduce element Area vector

%Initate Global cell for normalized stiffness matrix
Cell_K0e = cell(nelm,1);


%Initiate MMA parameters
s_slower = 0.6;
s_faster = 1.1;
s_init = 0.1;
A_last = 0;
A_lastlast = 0;
my = 0.5;
```

```
for n=1:nelm


    %Initate constant norm element stiffness matrix
    Cell_K0e{n} = bar2e(ex(n,:),ey(n,:),[E 1]);

    %Obtain a global length vector for each element.
    ec =[ex(n,:),ey(n,:)];

    x_0=[ec(1,2) - ec(1,1);
        ec(1,4) - ec(1,3)];

    l_0 = sqrt(x_0'*x_0);

    %Global length vector
    l_0_global(n,:) = l_0;

end


V = A_global'*l_0_global;



%Initate Tolerance
Tol = 10^-9;

%Initate quantaties
iter = 0;
Nres = 0;
q0 = 0;


lambda_min=10^10;
lambda_max=10^15;


%Optimization loop
while Nres > Tol || iter == 0

    iter = iter + 1;
    disp(['Iteration ',num2str(iter),'----------------------------'])

    K=sparse(ndof,ndof);
    %Element loop to assemble global stiffness matrix
    for i=1:nelm

        ep = [E A_global(i,:)];

        Ke = bar2e(ex(i,:),ey(i,:),ep);


        %Assemble element matrices to global matrices:
        indx = edof(i,2:end);
        K(indx,indx) = K(indx,indx)+Ke;
    end

    %Obtain displacement
    a = solveq(K,f,bc);
    ed = extract(edof,a);
```

```
for j=1:nelm


    %Modifying the asymptotes during the iterations
    %according to page 68 in the course literature.
    if iter < 3
        Lj(j,:) = A_global(j,:)-s_init*(Ae_max-Ae_min);
    else

        sign1 = A_global(j,:) - A_last(j,:);
        sign2 = A_last(j,:) - A_lastlast(j,:);

        if sign1*sign2 > 0
            Lj(j,:) = A_global(j,:) - s_faster*(A_last(j,:)-Lj(j,:));

        else
            Lj(j,:) = A_global(j,:)-s_slower*(A_last(j,:)-Lj(j,:));

        end
    end


    %If Lj is negative, give it a value of zero.
    if sign(Lj(j,:)) ~= 1
        Lj(j,:) = 0;
        %disp('Negative Lj')
    end

    %Update alpha_j per element
    alpha_j(j,:) = max([Ae_min, Lj(j,:)+my*(A_global(j,:)-Lj(j,:))]);

    %Compute q0j according to eq 5.37
    q0e(j,:) =((A_global(j,:)-Lj(j,:))^2)*ed(j,:)*Cell_K0e{j}*ed(j,:)';



end

%Solve the dual function
lambdastar = fzero(@(lambda) dphidlambda...
    ( lambda,l_0_global,V_max,Ae_min, Ae_max,nelm,q0e, Lj, alpha_j )...
    ,[lambda_min lambda_max]);

for z=1:nelm

    %Find new rho value per element based on new lambda from the dual
    %function
    Ae_star(z,:)= getAstar( lambdastar , Ae_min, Ae_max,l_0_global(z,:),q0e(z,:),

    %Compute element forces
    stress(z,:) = bar2s(ex(z,:),ey(z,:),[E Ae_star(z,:)],ed(z,:))/Ae_star(z,:);


end

%Compute the change between current and previous iteration
%regarding the updated area.
res=Ae_star-A_global;
Nres = norm(res);
A_lastlast = A_last;
```

```
    A_last = A_global;
    A_global = Ae_star;


    %Plot vectors
    Ae_star_plot(:,iter) = Ae_star;
    Iter_plot(:,iter) = iter;
    g0(:,iter) = f'*a;
    stress_global(:,iter) = stress;
    res_plot(:,iter) = Nres;
    g1(:,iter) = A_global'*l_0_global-V_max;


    V = A_global'*l_0_global;

    fprintf('Iter %i Res %e\n',iter,Nres)


end

figure
plot(Iter_plot,res_plot,'LineWidth',1.5)
xlabel('Iteration')
ylabel('Residual')



%%
fac = 4*10^6;       %Magnification of Cross Section area.
magnfac = 1*10^-10; %Magnification of Deformed structure.
figure
myeldisp2(ex,ey,ed,[1 2 0],magnfac,A_global,fac);
grid on
xlabel('x,[m]')
ylabel('y,[m]')
title(['Structural optimization with MMA approximation'],'FontSize', 14)


figure
plot(Iter_plot,g0,'LineWidth',2)
hold on

plot(Iter_plot,g1,'LineWidth',2)'
grid on
xlabel('Iterationer','FontSize', 14)
ylabel('g_i','FontSize', 14)
legend('g0- objective function','g1- constraint function')
title('g_i functions with MMA approximation','FontSize', 14)
set(gca,'FontSize',14)
xlim([0 200])


%Check Area fulfills Box constraint

A_max_check = max(A_global);
A_min_check = min(A_global);

if A_max_check <= Ae_max
    disp('Valid Box constraint regarding A_max')
else
    disp('Box constraint not fulfilled')
```

```
end


if A_min_check >= Ae_min
    disp('Valid Box constraint regarding A_min')
else
    disp('Box constraint not fulfilled ')
end
```

Dphidlambda

```
function [ dphidlambda ] = dphidlambda( lambda,l_0_global,Vmax,Ae_min...
, Ae_max,nelm,q0e,Lj,alpha_j )
%Compute dphidlambda, solving the dual function.
%
%Input: Lambda
%        l_0 , Global length vector
%        Vmax, Maximum volume
%        Ae_min, Minimum area
%        Ae_max, Maximum area
%        q0e    , qo array, computed in the main m-file.
%        Lj     , Moving asymptote,
%        alpha_j, move limit

%Output: dphidlambda

     h = 0;
    for z=1:nelm
        Ae_star(z,:) = getAstar( lambda , Ae_min,...
        Ae_max,l_0_global(z,:),q0e(z,:),Lj(z,:),alpha_j(z,:) );

        he = l_0_global(z,:)*Ae_star(z,:);
        h = he + h;
    end


%Constraint function
dphidlambda = h-Vmax;

end
```

GetAstar

```
function [ Ae_star ] = getAstar( lambda, Ae_min, Ae_max,l_0,q0e,Lj,alpha_j )
%Calculate A* for a given lambda.

%Input: lambda
%        Ae_min, Minimum area
%        Ae_max, Maximum area
%        l_0 , Global length vector
%        q0e    , qo(e) (per element), computed in the main m-file.
%        Lj     , Moving asymptote,
%        alpha_j, move limit

%Output: A*

Ae_trial =Lj + sqrt(q0e/(lambda*l_0));


   if  Ae_trial < alpha_j;

          Ae_star = alpha_j;

   elseif Ae_trial > Ae_max
```

```
            Ae_star = Ae_max;
    else


            Ae_star = Ae_trial;
    end



end
```

SIMP

Main

```
clc
clear all
close all
format long


el_length_Fine = 0.0025;
el_length_Coarse = 0.0071;

%Selection of coarse mesh or fine mesh.
prompt = ['Select Coarse Mesh or Fine Mesh:\n'...
     'A1: Coarse Mesh \nB1: Fine Mesh \n'];
A1 = 1;
B1 = 2;
selection = input(prompt)


if selection == 1

     load MBBCoarseMesh

     %Selection of what length scale R to use.
     prompt = ['Select a length scale R\n'...
         'A2: R = Element Side*1  \nB2: R = Element Side*2 \n'...
         'C2: R = Element Side*2.5\n'];
     A2 = 1;
     B2 = 2;
     C2 = 3;
     selection2 = input(prompt)



     if selection2 == 1

         R = el_length_Coarse;

     elseif selection2 == 2

         R = el_length_Coarse*2;

     elseif selection2 == 3

         R = el_length_Coarse*2.5;
```

```
    end

elseif selection == 2

    load MBBFineMesh

    %Selection of what length scale R to use.
    prompt = ['Select a length scale R\n'...
        'A2: R = Element Side*1  \nB2: R = Element Side*2 \n'...
        'C2: R = Element Side*2.5\n'];
    A2 = 1;
    B2 = 2;
    C2 = 3;
    selection2 = input(prompt)



    if selection2 == 1

        R = el_length_Fine;

    elseif selection2 == 2

        R = el_length_Fine*2;

    elseif selection2 == 3

        R = el_length_Fine*2.5;


    end

end


%Input min and max values for fzero function.
lambda_min=10^-8;
lambda_max=10^-1;


nen = 4; %Number of nodes per element

%Extract x and y coordinates
[exC,eyC]=coordxtr(edof,coord,dof,nen);
ec = [exC,eyC];


nelm = max(edof(:,1));  %Number of elements
ndof = max(max(dof));   %Number of degree of freedom, displacement part
nnod = ndof/2;          %Number of nodes

%Material parameters
E = 200e9;                %Young's modulus
rho_max = 1;               %Maximum density factor
rho_min = 10^-5;           %Minimum density factor
thickness = 20*10^-3;      %Thickness
V_box = 300*100*(10^-6)*thickness;
V_max = 0.4*V_box;         %Maximum allowed volume
v = 0.3;                   %Poissons tal
eq = 1;
```

```
%Initiate
x_e = ones(nelm,1)*10^-3; %Introduce element Area vector, with all Area being
a_e = zeros(nelm,1);       %Introduce Area vector
T = sparse(nnod,nelm);     %Introduce T matrix, [nnod x nelm]
M = sparse(nnod,nnod);     %Introduce Mass matrix
K_dens = sparse(nnod,nnod); %Introduce constant stiffness matrix,Ke [4x4]
K_rho =sparse(ndof,ndof); %Introduce stiffness matrix, Ke [8x8]
dgdrhotilde = sparse(nnod,1);
p = 3;                     %Penalization factor

ptype = 1; %Plane stress
ep =[ptype thickness 2];

%Hooke matrix
D0 = hooke(ptype,E,v);


%Parameters for flw2i4e
ep2 = [thickness 2];
D2 = [1 0;
    0 1];


for n=1:nelm

    a_e(n,:) = ElementArea( ec(n,:) );   %Area per element

    %Compute T matrix
    [Ke,Te] = flw2i4e(exC(n,:),eyC(n,:),ep2,D2,eq);

    %Assemble element matrices to global matrices:
    indx = enod(n,2:end);
    T(indx,n) = T(indx,n)+Te;

    indx = enod(n,2:end);
    K_dens(indx,indx) = K_dens(indx,indx)+Ke.*R^2;


    %Compute Mass matrix
    Me = flw2i4m(exC(n,:),eyC(n,:),thickness);

    %Assemble element matrices to global matrices:
    indx = enod(n,2:end);
    M(indx,indx) = M(indx,indx)+Me;



end

%Compute constant drhoTilde_drho matrix.
drhoTilde_drho = (K_dens + M)\T;


%Initate Tolerance
Tol = 10^-4;

%Initate quantaties
iter = 0;
Nres = 0;
```

```
alpha = 2;


%Optimization loop
while Nres > Tol || iter == 0

    iter = iter + 1;
    disp(['Iteration ',num2str(iter),'-----------------------------'])

    %Reset global matrices
    K_rho = sparse(ndof,ndof);
    dgdrhotilde = sparse(nnod,1);

    %Compute filtered density
    x_e_tilde = drhoTilde_drho*x_e;

    %Extract the densities at the nodal points.
    ed_rho=extract(enod,x_e_tilde);



    %Element loop to assemble global stiffness matrix
    for i=1:nelm

        Ke_rho=plani4e_rho(exC(i,:),eyC(i,:),ep,D0,ed_rho(i,:),p);

        %Assemble element matrices to global matrices:
        indx = edof(i,2:end);
        K_rho(indx,indx) = K_rho(indx,indx)+Ke_rho;


    end

    %Obtain displacement
    a = solveq(K_rho,F,bc);
    ed = extract(edof,a);


    for j=1:nelm

        %Compute dgdrhotilde
        dgdrhotilde_e=getdgdrhotilde_el(exC(j,:),eyC(j,:),ep,D0,ed(j,:)...
            ,ed_rho(j,:),p);

        %Assemble element matrices to global matrices:
        indx = enod(j,2:end);
        dgdrhotilde(indx,1) = dgdrhotilde(indx,1)+dgdrhotilde_e;


    end

    %Compute global dg_drho
    dg_drho = dgdrhotilde'*drhoTilde_drho;

    b = (1/alpha).*dg_drho'.*(x_e.^(1+alpha));


    %Solve the dual function
    lambdastar = fzero(@(lambda) dphidlambda_b...
        (lambda,rho_max,rho_min,alpha,b,a_e,V_max,nelm,thickness )...
        ,[lambda_min lambda_max]);
    for z=1:nelm

        %Find new rho value per element based on new lambda from the dual
        %function
        xe_star(z,:)= getx_e_star( lambdastar ,rho_max,rho_min,alpha,b(z,:),a_e(z,:)

    end

    %Compute the change between current and previous iteration
    %regarding the updated density.
    Nres = norm(xe_star-x_e);
    x_e = xe_star;

    %Plot vectors
    Iter_plot(:,iter) = iter;
    g0(:,iter) = F'*a;
    g1(:,iter) = x_e'*a_e*thickness-V_max;


    fprintf('Res %e\n',Nres)


    if iter == 200

        fprintf('Force break optimization loop at iteration: %i \n',iter)
        break
    end
end

%%

ed_el = extract(enod,x_e_tilde);
figure
fill(exC',eyC',ed_el')
title('Helmholtz density PDE filter')
xlabel('x, [m]')
ylabel('y, [m]')
colormap('Jet')

%%


figure
plot(Iter_plot,g0,'LineWidth',2)
grid on
xlabel('Iterationer')
ylabel('g_0')
title('Compliance history')


figure
plot(Iter_plot,g1,'LineWidth',2)
xlabel('Iterationer')
ylabel('g_1')
grid on
title('Weight history')
```

Dphidlambda

```
function [ dphidlambda ] = dphidlambda_b(lambda,rho_max,rho_min,alpha,...
be,a_e,V_max,nelm,thickness)
%Compute dphidlambda, solving the dual function.
```

```
%
%Input: Lambda
%       l_0 , Global length vector
%       Vmax, Maximum volume
%       rho_min, Minimum density
%       rho_max, Maximum density
%       alpha, set parameter in the main m-file , connected to the
%       intervening variable in the SIMP OC method.
%       be    , b array, computed in the main m-file.
%       a_e   , area of each element
%       nelm  , number of elements
%       thickness
%
%Output: dphidlambda

     h = 0;
    for z=1:nelm
        xe_star(z,:) = getx_e_star( lambda,rho_max,rho_min,alpha,be(z,:),a_e(z,:) );

        he = a_e(z,:)*xe_star(z,:);
        h = he + h;
    end


%Constraint function
dphidlambda = h*thickness-V_max;

end
```

getx-e-star

```
function [ xe_star ] = getx_e_star( lambda,rho_max,rho_min,alpha,be,a_e )
%Calculate x* (= rho*) for a given lambda.

%Input: lambda
%       rho_min, Minimum density
%       rho_max, Maximum density
%       l_0 , Global length vector
%       be    , b array, computed in the main m-file.
%       alpha, set parameter in the main m-file , connected to the
%       intervening variable in the SIMP OC method.
%       a_e   , area of each element

%Output: x*


xe_trial = ((alpha*be)/(lambda*a_e))^(1/(1+alpha));


   if  xe_trial < rho_min;


         xe_star = rho_min;

   elseif xe_trial > rho_max



        xe_star = rho_max;
   else


        xe_star = xe_trial;
```

```
    end



end
```

ElementArea

```
function [ A0 ] = ElementArea( ec )
%Compute element area

%Input: ec, coordinate matrix,
%       where ec(e) = [1x8]

%Output: A0, the area of one element.

x1 = ec(1,1);
x2 = ec(1,2);
x3 = ec(1,3);
x4 = ec(1,4);

y1 = ec(1,5);
y2 = ec(1,6);
y3 = ec(1,7);
y4 = ec(1,8);

%Element Area.
A0 =  (1/2)*abs(x1*y2+x2*y3+x3*y4+x4*y1-x2*y1-x3*y2-x4*y3-x1*y4);

end
```

Density filter

Main, constant weight function

```
clc
clear all
close all
format long
tic

% load MBBFineMesh
load MBBCoarseMesh

nen = 4;

[ex,ey]=coordxtr(edof,coord,dof,nen);
ec = [ex,ey];

%Number of elements
nelm = max(edof(:,1));
ndof = max(max(dof));

% Material parameters
E = 200e9;                     % Young's modulus
rho_max = 1;                   % Maximum density factor
rho_min = 10^-5;               % Minimum density factor
thickness = 20*10^-3;          % Thickness
V_box = 300*100*(10^-6)*thickness;
V_max = 0.4*V_box;             % Maximum allowed volume
v = 0.3;                       % Poissons tal
```

```
%Filter radius
%R = 0.0025;
R = 0.0071*2;

%Initiate global matrix
K = sparse(ndof,ndof);              % Stiffness
K0=sparse(ndof,ndof);               % Normalized stiffness
Cell_K0e = cell(nelm,1);            % Normalized stiffness cell
x_e = ones(nelm,1)*10^-1;           % Initial design parameter
a_e = zeros(nelm,1);                % Area vector
p = 3;                              % Penalization factor

ptype = 1;                          % Plane stress
ep =[ptype thickness 2];

% Constitutive matrix
D0 = hooke(1,E,v);

for n=1:nelm

    % Calculate area and volume of each element
    a_e(n,:) = ElementArea( ec(n,:) );
    Vol_el(n,:) = thickness*ElementArea(ec(n,:));

    %Initate constant norm element stiffness matrix
    [Cell_K0e{n}]=plani4e(ex(n,:),ey(n,:),ep,D0);


    % Calculate centrum of current element and put in a vector
    y_centrum=min(ey(n,:))+(max(ey(n,:))-min(ey(n,:)))/2;
    x_centrum=min(ex(n,:))+(max(ex(n,:))-min(ex(n,:)))/2;
    kord_centrum=[x_centrum,y_centrum]';

    % Neigbour matrix
    N(:,1)=edof(:,1);
    var=0;

    for lite=1:nelm

        % Calculate centrum and check if neighbour
        y_centrum_test=min(ey(lite,:))+(max(ey(lite,:))-min(ey(lite,:)))/2;
        x_centrum_test=min(ex(lite,:))+(max(ex(lite,:))-min(ex(lite,:)))/2;
        kord_test=[x_centrum_test,y_centrum_test]';

        if norm(kord_test-kord_centrum)< R %&& lite ~= n

            % Update neigbouring matrix
            N(n,lite)=lite;

            % Distance from centrum of current element cone-shaped function
            weights(n,lite)=1;

            % Global M-matrix
            M(n,lite)=weights(n,lite)*Vol_el(n,:);
            var=weights(n,lite)*Vol_el(n,:)+var;
        end
    end
    M(n,:)=M(n,:)/var;
end

%Initate Tolerance
```

```
Tol = 10^-2;

%Initate quantaties
iter = 0;
Nres = 0;
alpha = 3;

% Interval to fzero
lambda_min=10^-20;
lambda_max=10^20;

%Optimization loop
while Nres > Tol || iter == 0

    iter = iter + 1;
    disp(['Iteration ' ,num2str(iter),'-----------------------------'])

    K = sparse(ndof,ndof);

    % Element loop to assemble global stiffness matrix
    for i=1:nelm

        x_e_tilde(i,:)=weights(i,:)*x_e/sum(weights(i,:));

        % Constitutive matrix
        D = (x_e_tilde(i,:)^p)*D0;

        % Stiffness matrix
        Ke = plani4e(ex(i,:),ey(i,:),ep,D);

        %Assemble element matrices to global matrices:
        indx = edof(i,2:end);
        K(indx,indx) = K(indx,indx)+Ke;
    end

    %Obtain displacement
    a = solveq(K,F,bc);
    ed = extract(edof,a);


    % Assemble and calculate the sensitivities
    for j=1:nelm
        dC_drhotilde(j,:)= ed(j,:)*(-p*x_e_tilde(j,:)^(p-1))*Cell_K0e{j}*ed(j,:)';
    end

    dC_drho=dC_drhotilde'*M;


    be= -(1/alpha)*x_e.^(1+alpha).*dC_drho';

    % Solve the dual Lagrangian function
    lambdastar = fzero(@(lambda) dphidlambda_b...
        (lambda,rho_max,rho_min,alpha,be,a_e,V_max,nelm,thickness )...
        ,[lambda_min lambda_max]);

    % Find KKT point
    for z=1:nelm
        xe_star(z,:)= getx_e_star( lambdastar ,rho_max,rho_min,alpha,be(z,:),a_e(z,:)
    end

    % Calculate the change
    Nres = norm(xe_star-x_e);
    x_e = xe_star;

    %Plot vectors
    Iter_plot(:,iter) = iter;
    g0(:,iter) = F'*a;
    g1(:,iter) = x_e_tilde'*a_e*thickness-V_max;

    fprintf('Res %e\n',Nres)

    if iter == 1000
        fprintf('Force break optimization loop at iteration: %i \n',iter)
        break
    end

    % Check if R is to small
    if any(dC_drho<0)==0
        fprintf('R to small')
        break
    end
end

%%

figure
plot(Iter_plot,g0,'LineWidth',2)
grid on
xlabel('Iterationer')
ylabel('g_0')
title('Compliance history')

figure
plot(Iter_plot,g1,'LineWidth',2)
xlabel('Iterationer')
ylabel('g_1')
grid on
title('Weight history')

figure
colormap('jet')
fill(ex',ey',x_e_tilde)
title('Density Filter')
xlabel('x [m]')
ylabel('y [m]')

toc
```

Main, cone-shaped weight function

```
clc
clear all
close all
format long
tic

% load MBBFineMesh
load MBBCoarseMesh

nen = 4;

[ex,ey]=coordxtr(edof,coord,dof,nen);
ec = [ex,ey];
```

```
%Number of elements
nelm = max(edof(:,1));
ndof = max(max(dof));

% Material parameters
E = 200e9;                          % Young's modulus
rho_max = 1;                        % Maximum density factor
rho_min = 10^-5;                    % Minimum density factor
thickness = 20*10^-3;               % Thickness
V_box = 300*100*(10^-6)*thickness;
V_max = 0.4*V_box;                  % Maximum allowed volume
v = 0.3;                            % Poissons tal


%Filter radius
%R = 0.0025;
R = 0.0071*3.5;

%Initiate global matrix
K = sparse(ndof,ndof);              % Stiffness
K0=sparse(ndof,ndof);               % Normalized stiffness
Cell_K0e = cell(nelm,1);            % Normalized stiffness cell
x_e = ones(nelm,1)*10^-1;           % Initial design parameter
a_e = zeros(nelm,1);                % Area vector
p = 3;                              % Penalization factor

ptype = 1;                          % Plane stress
ep =[ptype thickness 2];

% Constitutive matrix
D0 = hooke(1,E,v);

for n=1:nelm

    % Calculate area and volume of each element
    a_e(n,:) = ElementArea( ec(n,:) );
    Vol_el(n,:) = thickness*ElementArea(ec(n,:));

    %Initate constant norm element stiffness matrix
    [Cell_K0e{n}]=plani4e(ex(n,:),ey(n,:),ep,D0);


    % Calculate centrum of current element and put in a vector
    y_centrum=min(ey(n,:))+(max(ey(n,:))-min(ey(n,:)))/2;
    x_centrum=min(ex(n,:))+(max(ex(n,:))-min(ex(n,:)))/2;
    kord_centrum=[x_centrum,y_centrum]';

    % Neigbour matrix
    N(:,1)=edof(:,1);
    var=0;

    for lite=1:nelm

        % Calculate centrum and check if neighbour
        y_centrum_test=min(ey(lite,:))+(max(ey(lite,:))-min(ey(lite,:)))/2;
        x_centrum_test=min(ex(lite,:))+(max(ex(lite,:))-min(ex(lite,:)))/2;
        kord_test=[x_centrum_test,y_centrum_test]';

        if norm(kord_test-kord_centrum)< R %&& lite ~= n

            % Update neigbouring matrix
            N(n,lite)=lite;

            % Distance from centrum of current element cone-shaped function
            weights(n,lite)=R - norm(kord_test-kord_centrum);

            % Global M-matrix
            M(n,lite)=weights(n,lite)*Vol_el(n,:);
            var=weights(n,lite)*Vol_el(n,:)+var;
        end
    end
    M(n,:)=M(n,:)/var;
end

%Initate Tolerance
Tol = 10^-2;

%Initate quantaties
iter = 0;
Nres = 0;
alpha = 3;

% Interval to fzero
lambda_min=10^-20;
lambda_max=10^20;

%Optimization loop
while Nres > Tol || iter == 0

    iter = iter + 1;
    disp(['Iteration ' ,num2str(iter),'----------------------------'])

    K = sparse(ndof,ndof);

    % Element loop to assemble global stiffness matrix
    for i=1:nelm

        x_e_tilde(i,:)=weights(i,:)*x_e/sum(weights(i,:));

        % Constitutive matrix
        D = (x_e_tilde(i,:)^p)*D0;

        % Stiffness matrix
        Ke = plani4e(ex(i,:),ey(i,:),ep,D);

        %Assemble element matrices to global matrices:
        indx = edof(i,2:end);
        K(indx,indx) = K(indx,indx)+Ke;
    end

    %Obtain displacement
    a = solveq(K,F,bc);
    ed = extract(edof,a);


    % Assemble and calculate the sensitivities
    for j=1:nelm
        dC_drhotilde(j,:)= ed(j,:)*(-p*x_e_tilde(j,:)^(p-1))*Cell_K0e{j}*ed(j,:)';
    end

    dC_drho=dC_drhotilde'*M;
    be= -(1/alpha)*x_e.^(1+alpha).*dC_drho';

    % Solve the dual Lagrangian function
    lambdastar = fzero(@(lambda) dphidlambda_b...
        (lambda,rho_max,rho_min,alpha,be,a_e,V_max,nelm,thickness )...
        ,[lambda_min lambda_max]);

    % Find KKT point
    for z=1:nelm
        xe_star(z,:)= getx_e_star( lambdastar ,rho_max,rho_min,alpha,be(z,:),a_e(z,:)
    end

    % Calculate the change
    Nres = norm(xe_star-x_e);
    x_e = xe_star;

    %Plot vectors
    Iter_plot(:,iter) = iter;
    g0(:,iter) = F'*a;
    g1(:,iter) = x_e_tilde'*a_e*thickness-V_max;

    fprintf('Res %e\n',Nres)

    if iter == 1000
        fprintf('Force break optimization loop at iteration: %i \n',iter)
        break
    end

    % Check if R is to small
    if any(dC_drho<0)==0
        fprintf('R to small')
        break
    end
end

%%

figure
plot(Iter_plot,g0,'LineWidth',2)
grid on
xlabel('Iterationer')
ylabel('g_0')
title('Compliance history')

figure
plot(Iter_plot,g1,'LineWidth',2)
xlabel('Iterationer')
ylabel('g_1')
grid on
title('Weight history')

figure
colormap('jet')
fill(ex',ey',x_e_tilde)
title('Density Filter')
xlabel('x [m]')
ylabel('y [m]')

toc
```

PDE filter

Main

```
clc
clear all
close all
format long


el_length_Fine = 0.0025;
el_length_Coarse = 0.0071;

%Selection of coarse mesh or fine mesh.
prompt = ['Select Coarse Mesh or Fine Mesh:\n'...
     'A1: Coarse Mesh \nB1: Fine Mesh \n'];
A1 = 1;
B1 = 2;
selection = input(prompt)


if selection == 1

    load MBBCoarseMesh

    %Selection of what length scale R to use.
    prompt = ['Select a length scale R\n'...
         'A2: R = Element Side*1  \nB2: R = Element Side*2 \n'...
         'C2: R = Element Side*2.5\n'];
    A2 = 1;
    B2 = 2;
    C2 = 3;
    selection2 = input(prompt)



    if selection2 == 1

        R = el_length_Coarse;

    elseif selection2 == 2

        R = el_length_Coarse*2;

    elseif selection2 == 3

        R = el_length_Coarse*2.5;


    end

elseif selection == 2

    load MBBFineMesh

    %Selection of what length scale R to use.
    prompt = ['Select a length scale R\n'...
         'A2: R = Element Side*1  \nB2: R = Element Side*2 \n'...
         'C2: R = Element Side*2.5\n'];
    A2 = 1;
    B2 = 2;
    C2 = 3;
    selection2 = input(prompt)


    if selection2 == 1

        R = el_length_Fine;

    elseif selection2 == 2

        R = el_length_Fine*2;

    elseif selection2 == 3

        R = el_length_Fine*2.5;


    end

end


%Input min and max values for fzero function.
lambda_min=10^-8;
lambda_max=10^-1;


nen = 4; %Number of nodes per element

%Extract x and y coordinates
[exC,eyC]=coordxtr(edof,coord,dof,nen);
ec = [exC,eyC];


nelm = max(edof(:,1));  %Number of elements
ndof = max(max(dof));   %Number of degree of freedom, displacement part
nnod = ndof/2;          %Number of nodes

%Material parameters
E = 200e9;               %Young's modulus
rho_max = 1;             %Maximum density factor
rho_min = 10^-5;         %Minimum density factor
thickness = 20*10^-3;    %Thickness
V_box = 300*100*(10^-6)*thickness;
V_max = 0.4*V_box;       %Maximum allowed volume
v = 0.3;                 %Poissons tal
eq = 1;


%Initiate
x_e = ones(nelm,1)*10^-3; %Introduce element Area vector, with all Area being
a_e = zeros(nelm,1);      %Introduce Area vector
T = sparse(nnod,nelm);    %Introduce T matrix, [nnod x nelm]
M = sparse(nnod,nnod);    %Introduce Mass matrix
K_dens = sparse(nnod,nnod); %Introduce constant stiffness matrix,Ke [4x4]
K_rho =sparse(ndof,ndof); %Introduce stiffness matrix, Ke [8x8]
dgdrhotilde = sparse(nnod,1);
p = 3;                    %Penalization factor

ptype = 1; %Plane stress
ep =[ptype thickness 2];
```

```
%Hooke matrix
D0 = hooke(ptype,E,v);


%Parameters for flw2i4e
ep2 = [thickness 2];
D2 = [1 0;
    0 1];


for n=1:nelm

    a_e(n,:) = ElementArea( ec(n,:) );   %Area per element

    %Compute T matrix
    [Ke,Te] = flw2i4e(exC(n,:),eyC(n,:),ep2,D2,eq);

    %Assemble element matrices to global matrices:
    indx = enod(n,2:end);
    T(indx,n) = T(indx,n)+Te;

    indx = enod(n,2:end);
    K_dens(indx,indx) = K_dens(indx,indx)+Ke.*R^2;


    %Compute Mass matrix
    Me = flw2i4m(exC(n,:),eyC(n,:),thickness);

    %Assemble element matrices to global matrices:
    indx = enod(n,2:end);
    M(indx,indx) = M(indx,indx)+Me;



end

%Compute constant drhoTilde_drho matrix.
drhoTilde_drho = (K_dens + M)\T;


%Initate Tolerance
Tol = 10^-4;

%Initate quantaties
iter = 0;
Nres = 0;
alpha = 2;


%Optimization loop
while Nres > Tol || iter == 0

    iter = iter + 1;
    disp(['Iteration ',num2str(iter),'----------------------------'])

    %Reset global matrices
    K_rho = sparse(ndof,ndof);
    dgdrhotilde = sparse(nnod,1);

    %Compute filtered density
```

```
x_e_tilde = drhoTilde_drho*x_e;

%Extract the densities at the nodal points.
ed_rho=extract(enod,x_e_tilde);



%Element loop to assemble global stiffness matrix
for i=1:nelm

    Ke_rho=plani4e_rho(exC(i,:),eyC(i,:),ep,D0,ed_rho(i,:),p);

    %Assemble element matrices to global matrices:
    indx = edof(i,2:end);
    K_rho(indx,indx) = K_rho(indx,indx)+Ke_rho;


end

%Obtain displacement
a = solveq(K_rho,F,bc);
ed = extract(edof,a);


for j=1:nelm

    %Compute dgdrhotilde
    dgdrhotilde_e=getdgdrhotilde_el(exC(j,:),eyC(j,:),ep,D0,ed(j,:)...
        ,ed_rho(j,:),p);

    %Assemble element matrices to global matrices:
    indx = enod(j,2:end);
    dgdrhotilde(indx,1) = dgdrhotilde(indx,1)+dgdrhotilde_e;


end

%Compute global dg_drho
dg_drho = dgdrhotilde'*drhoTilde_drho;

b = (1/alpha).*dg_drho'.*(x_e.^(1+alpha));


%Solve the dual function
lambdastar = fzero(@(lambda) dphidlambda_b...
    (lambda,rho_max,rho_min,alpha,b,a_e,V_max,nelm,thickness )...
    ,[lambda_min lambda_max]);

for z=1:nelm

    %Find new rho value per element based on new lambda from the dual
    %function
    xe_star(z,:)= getx_e_star( lambdastar ,rho_max,rho_min,alpha,b(z,:),a_e(z,:)

end

%Compute the change between current and previous iteration
%regarding the updated density.
Nres = norm(xe_star-x_e);
x_e = xe_star;
```

```
    %Plot vectors
    Iter_plot(:,iter) = iter;
    g0(:,iter) = F'*a;
    g1(:,iter) = x_e'*a_e*thickness-V_max;


    fprintf('Res %e\n',Nres)


    if iter == 200

        fprintf('Force break optimization loop at iteration: %i \n',iter)
        break
    end
end

%%

ed_el = extract(enod,x_e_tilde);
figure
fill(exC',eyC',ed_el')
title('Helmholtz density PDE filter')
xlabel('x, [m]')
ylabel('y, [m]')
colormap('Jet')

%%


figure
plot(Iter_plot,g0,'LineWidth',2)
grid on
xlabel('Iterationer')
ylabel('g_0')
title('Compliance history')


figure
plot(Iter_plot,g1,'LineWidth',2)
xlabel('Iterationer')
ylabel('g_1')
grid on
title('Weight history')
```

Extra

myeldisp2

```
function [magnfac]=myeldisp2(ex,ey,ed,plotpar,magnfac,Area,fac)
%
%
%-------------------------------------------------------------
% PURPOSE
%   Draw the deformed 2D mesh for a number of elements of
%   the same type. Supported elements are:
%
%           1) -> bar element            2) -> beam el.
%           3) -> triangular 3 node el.  4) -> quadrilateral 4 node el.
%           5) -> 8-node isopar. element
%  INPUT
%    ex,ey:.........  nen:   number of element nodes
%                     nel:   number of elements
%    ed:      element displacement matrix
%
%    plotpar=[  linetype, linecolor, nodemark]
%
%              linetype=1 -> solid     linecolor=1 -> white
%                       2 -> dashed              2 -> green
%                       3 -> dotted              3 -> yellow
%                                                4 -> red
%              nodemark=1 -> circle
%                       2 -> star
%                       0 -> no mark
%
%    magnfac:  magnification factor for displacements
%
%    Rem. Default if magnfac and plotpar is left out is auto magnification
%         and dashed white lines with circles at nodes -> plotpar=[2 1 1]
%-------------------------------------------------------------

% LAST MODIFIED: P-A Hansson  1994-03-27
% Copyright (c)  Division of Structural Mechanics and
%                Department of Solid Mechanics.
%                Lund Institute of Technology
%-------------------------------------------------------------
%
%  if ~((nargin==3)|(nargin==4)|(nargin==5))
%     error('??? Wrong number of input arguments!')
%
%  end
nargin = 5;

 a=size(ex); b=size(ey);

 if (a-b)==[0 0]
    nen=a(2);
 else
    error('??? Check size of coordinate input arguments!')

 end

 c=size(ed);

 if ~(c(1)==a(1))
    error('??? Check size of displacement input arguments!')

 end

 ned=c(2);

 dxmax=max(max(ex')-min(ex')); dymax=max(max(ey')-min(ey'));
 dlmax=max(dxmax,dymax);
 edmax=max(max(abs(ed)));
 krel=0.1;

 if nargin==3;
      plotpar=[2 1 1]; magnfac=krel*dlmax/edmax;
 elseif nargin==4
      magnfac=krel*dlmax/edmax;
 end

 [s1,s2]=pltstyle(plotpar);
 k=magnfac;
```

```
% ********** Bar or Beam elements *************
    if nen==2
       if ned==4  % -----------  Bar elements -------------
          x=(ex+k*ed(:,[1  3]))';
          y=(ey+k*ed(:,[2  4]))';
          xc=x;
          yc=y;
       elseif ned==6  % -------- Beam elements ------------
          x=(ex+k*ed(:,[1  4]))';
          y=(ey+k*ed(:,[2  5]))';
          [exc,eyc]=beam2crd(ex,ey,ed,k);
          xc=exc';
          yc=eyc';
       end
% ********* 2D triangular elements ************
    elseif nen==3
       x=(ex+k*ed(:,[1  3  5]))';
       y=(ey+k*ed(:,[2  4  6]))';
       xc=[x; x(1,:)];
       yc=[y; y(1,:)];

% ********* 2D quadrilateral elements *********
    elseif nen==4
       x=(ex+k*ed(:,[1  3  5  7]))';
       y=(ey+k*ed(:,[2  4  6  8]))';
       xc=[x; x(1,:)];
       yc=[y; y(1,:)];
 % ********* 2D 8-node quadratic elements *********
    elseif nen==8
       x=(ex+k*ed(:,[1  3  5  7  9  11  13  15]));
       y=(ey+k*ed(:,[2  4  6  8  10  12  14  16]));
%       xc=[x(1); x(5); x(2); x(6); x(3); x(7); x(4); x(8);x(1)];
%       yc=[y(1); y(5); y(2); y(6); y(3); y(7); y(4); y(8);y(1)];
%
% isoparametric elements
%
    t=-1;
    n=0;
    for s=-1:0.4:1
      n=n+1;
      N1=-1/4*(1-t)*(1-s)*(1+t+s);
      N2=-1/4*(1+t)*(1-s)*(1-t+s);
      N3=-1/4*(1+t)*(1+s)*(1-t-s);
      N4=-1/4*(1-t)*(1+s)*(1+t-s);
      N5=1/2*(1-t*t)*(1-s);
      N6=1/2*(1+t)*(1-s*s);
      N7=1/2*(1-t*t)*(1+s);
      N8=1/2*(1-t)*(1-s*s);
      N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

      x1(n,:)=N*x';
      y1(n,:)=N*y';
    end;
    xc=[xc x1];
    yc=[yc y1];
    clear x1
    clear y1
%
    s=1;
    n=0;
```

```
        for t=-1:0.4:1
          n=n+1;
          N1=-1/4*(1-t)*(1-s)*(1+t+s);
          N2=-1/4*(1+t)*(1-s)*(1-t+s);
          N3=-1/4*(1+t)*(1+s)*(1-t-s);
          N4=-1/4*(1-t)*(1+s)*(1+t-s);
          N5=1/2*(1-t*t)*(1-s);
          N6=1/2*(1+t)*(1-s*s);
          N7=1/2*(1-t*t)*(1+s);
          N8=1/2*(1-t)*(1-s*s);
          N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

          x1(n,:)=N*x';
          y1(n,:)=N*y';
        end;
        xc=[xc x1];
        yc=[yc y1];
        clear x1
        clear y1
%
        t=1;
        n=0;
        for s=1:-0.4:-1
          n=n+1;
          N1=-1/4*(1-t)*(1-s)*(1+t+s);
          N2=-1/4*(1+t)*(1-s)*(1-t+s);
          N3=-1/4*(1+t)*(1+s)*(1-t-s);
          N4=-1/4*(1-t)*(1+s)*(1+t-s);
          N5=1/2*(1-t*t)*(1-s);
          N6=1/2*(1+t)*(1-s*s);
          N7=1/2*(1-t*t)*(1+s);
          N8=1/2*(1-t)*(1-s*s);
          N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

          x1(n,:)=N*x';
          y1(n,:)=N*y';
        end;
        xc=[xc x1];
        yc=[yc y1];
        clear x1
        clear y1
%
        s=-1;
        n=0;
        for t=1:-0.4:-1
          n=n+1;
          N1=-1/4*(1-t)*(1-s)*(1+t+s);
          N2=-1/4*(1+t)*(1-s)*(1-t+s);
          N3=-1/4*(1+t)*(1+s)*(1-t-s);
          N4=-1/4*(1-t)*(1+s)*(1+t-s);
          N5=1/2*(1-t*t)*(1-s);
          N6=1/2*(1+t)*(1-s*s);
          N7=1/2*(1-t*t)*(1+s);
          N8=1/2*(1-t)*(1-s*s);
          N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

          x1(n,:)=N*x';
          y1(n,:)=N*y';
        end;
        xc=[xc x1];
        yc=[yc y1];
```

```
    clear x1
    clear y1
%
%*****************************************************
    else
       error('Sorry, this element is currently not supported!')

    end
% ************* plot commands ******************
    axis('equal')
    hold on
    for el=1:size(ex,1)
        w = Area(el)*fac;
        plot(xc(:,el),yc(:,el),s1,'linewidth',w)

    end

    if s2~=' '
      plot(x,y,s2)
    end
    hold off
%--------------------------end--------------------------------
```

myeldraw2

```
function myeldraw2(ex,ey,plotpar,Area,fac)
%-------------------------------------------------------------
% PURPOSE
%   Draw the undeformed 2D mesh for a number of elements of
%   the same type. Supported elements are:
%
%   1) -> bar element              2) -> beam el.
%   3) -> triangular 3 node el.    4) -> quadrilateral 4 node el.
%   5) -> 8-node isopar. elemen
%
% INPUT
%    ex,ey:.......... nen:   number of element nodes
%                     nel:   number of elements
%    plotpar=[ linetype, linecolor, nodemark]
%
%             linetype=1 -> solid     linecolor=1 -> white
%                      2 -> dashed              2 -> green
%                      3 -> dotted              3 -> yellow
%                                               4 -> red
%
%             nodemark=1 -> circle
%                      2 -> star
%                      0 -> no mark
%
%    elnum=edof(:,1) ; i.e. the first column in the topology matrix
%
%    Rem. Default is solid white lines with circles at nodes.
%
%-------------------------------------------------------------

% LAST MODIFIED: P-A Hansson  1994-03-27
% Copyright (c)  Division of Structural Mechanics and
%                Department of Solid Mechanics.
%                Lund Institute of Technology
%-------------------------------------------------------------
%
%  if ~((nargin==2)|(nargin==3)|(nargin==4))
%     disp('??? Wrong number of input arguments!')
%      break
%      error('??? Wrong number of input arguments!')
%  end

nargin = 3;

 a=size(ex); b=size(ey);

 if (a-b)==[0 0]
     nel=a(1);nen=a(2);
 else
    error('??? Check size of coordinate input arguments!')
   %disp('??? Check size of coordinate input arguments!')
  %break
 end
 if nargin==2;
      plotpar=[1 1 1];
 end
 [s1,s2]=pltstyle(plotpar);

% ************************************************
% ************* plot coordinates ******************
% ************************************************
 x0=sum(ex')/nen; y0=sum(ey')/nen;

% ********** Bar or Beam elements *************
 if nen==2
    x=ex' ;
    y=ey';
    xc=x ;yc=y;

% ********* 2D triangular elements ************
 elseif nen==3
    x=ex' ;
    y=ey';
    xc=[x ; x(1,:)];  yc=[y ; y(1,:)];

% ********* 2D quadrilateral elements *********
 elseif nen==4
    x=ex' ;
    y=ey';
    xc=[x ; x(1,:)]; yc=[y ; y(1,:)];
% ********* 2D 8 node quadratic elements *********
 elseif nen== 8
    x=ex;
    y=ey;
   %xc=[x(1);x(5);x(2);x(6);x(3);x(7);x(4);x(8);x(1)];
   %yc=[y(1);y(5);y(2);y(6);y(3);y(7);y(4);y(8);y(1)];
xc=[];
yc=[];
%
% isoparametric elements
%
    t=-1;
    n=0;
    for s=-1:0.4:1
      n=n+1;
      N1=-1/4*(1-t)*(1-s)*(1+t+s);
      N2=-1/4*(1+t)*(1-s)*(1-t+s);
      N3=-1/4*(1+t)*(1+s)*(1-t-s);
      N4=-1/4*(1-t)*(1+s)*(1+t-s);
      N5=1/2*(1-t*t)*(1-s);
      N6=1/2*(1+t)*(1-s*s);
      N7=1/2*(1-t*t)*(1+s);
      N8=1/2*(1-t)*(1-s*s);
      N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

      x1(n,:)=N*x';
      y1(n,:)=N*y';
    end;
    xc=[xc x1];
    yc=[yc y1];
    clear x1
    clear y1
%
    s=1;
    n=0;
    for t=-1:0.4:1
      n=n+1;
      N1=-1/4*(1-t)*(1-s)*(1+t+s);
      N2=-1/4*(1+t)*(1-s)*(1-t+s);
      N3=-1/4*(1+t)*(1+s)*(1-t-s);
      N4=-1/4*(1-t)*(1+s)*(1+t-s);
      N5=1/2*(1-t*t)*(1-s);
      N6=1/2*(1+t)*(1-s*s);
      N7=1/2*(1-t*t)*(1+s);
      N8=1/2*(1-t)*(1-s*s);
      N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

      x1(n,:)=N*x';
      y1(n,:)=N*y';
    end;
    xc=[xc x1];
    yc=[yc y1];
    clear x1
    clear y1
%
    t=1;
    n=0;
    for s=1:-0.4:-1
      n=n+1;
      N1=-1/4*(1-t)*(1-s)*(1+t+s);
      N2=-1/4*(1+t)*(1-s)*(1-t+s);
      N3=-1/4*(1+t)*(1+s)*(1-t-s);
      N4=-1/4*(1-t)*(1+s)*(1+t-s);
      N5=1/2*(1-t*t)*(1-s);
      N6=1/2*(1+t)*(1-s*s);
      N7=1/2*(1-t*t)*(1+s);
      N8=1/2*(1-t)*(1-s*s);
      N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

      x1(n,:)=N*x';
      y1(n,:)=N*y';
    end;
    xc=[xc x1];
    yc=[yc y1];
    clear x1
    clear y1
%
    s=-1;
    n=0;
    for t=1:-0.4:-1
```

```
        n=n+1;
        N1=-1/4*(1-t)*(1-s)*(1+t+s);
        N2=-1/4*(1+t)*(1-s)*(1-t+s);
        N3=-1/4*(1+t)*(1+s)*(1-t-s);
        N4=-1/4*(1-t)*(1+s)*(1+t-s);
        N5=1/2*(1-t*t)*(1-s);
        N6=1/2*(1+t)*(1-s*s);
        N7=1/2*(1-t*t)*(1+s);
        N8=1/2*(1-t)*(1-s*s);
        N=[ N1, N2, N3 ,N4, N5, N6, N7, N8 ];

        x1(n,:)=N*x';
        y1(n,:)=N*y';
      end;
      xc=[xc x1];
      yc=[yc y1];
      clear x1
      clear y1
%**********************************************************
  else
      error('!!!! Sorry, this element is currently not supported!')
     %disp('!!!! Sorry, this element is currently not supported!')
     %break
  end
% **************************************************
% *************** plot commands ******************
% **************************************************
  axis('equal')
  hold on
  plot(xc,yc,s1)

  for el=1:nel
      w = Area(el)*fac;

      plot(xc(:,el),yc(:,el),s1,'linewidth',w)

end

  if s2~=' '
    plot(x,y,s2)
  end
  if nargin==4
      for i=1:nel
          h=text(x0(i),y0(i),int2str(elnum(i)));
          set(h,'fontsize',8);
      end
  end
  xlabel('x'); ylabel('y');
  hold off
%-------------------------end--------------------------------
```

plani4e-rho

```
function Ke=plani4e_rho(ex,ey,ep,D,ed_rho,q)
%
%-------------------------------------------------------------
% PURPOSE
%  Calculate the stiffness matrix for a 4 node isoparametric
%  element in plane strain or plane stress with a distributed
%  density field.
%
% INPUT:   ex = [x1 x2 x3 x4]  element coordinates
%          ey = [y1 y2 y3 y4]
```

```
%
%          ep =[ptype  t  ir]     element property
%                                   ptype: analysis type
%                                   ir: integration rule
%                                   t : thickness
%
%          D                      constitutive matrix
%
%          eq = [bx; by]          bx: body force in x direction
%                                 by: body force in y direction
%
%          ed_rho = [rho1, rho2, rho3, rho4]   : The densities at the
%                                                nodal points of the element.
%
% OUTPUT: Ke : element stiffness matrix (8 x 8)
%         fe : equivalent nodal forces (8 x 1)
%-------------------------------------------------------------

% LAST MODIFIED: M Ristinmaa   1995-10-25
%                Eric Borgqvist  2014-03-04
% Copyright (c)  Division of Structural Mechanics and
%                Department of Solid Mechanics.
%                Lund Institute of Technology
%-------------------------------------------------------------
  ptype=ep(1);  t=ep(2);   ir=ep(3);   ngp=ir*ir;

%--------- gauss points --------------------------------------
  if ir==1
    g1=0.0; w1=2.0;
    gp=[ g1 g1 ];   w=[ w1 w1 ];
  elseif ir==2
    g1=0.577350269189626; w1=1;
    gp(:,1)=[-g1; g1;-g1; g1];   gp(:,2)=[-g1;-g1; g1; g1];
    w(:,1)=[ w1; w1; w1; w1];    w(:,2)=[ w1; w1; w1; w1];
  elseif ir==3
    g1=0.774596669241483; g2=0.;
    w1=0.555555555555555; w2=0.888888888888888;
    gp(:,1)=[-g1;-g2; g1;-g1; g2; g1;-g1; g2; g1];
    gp(:,2)=[-g1;-g1;-g1; g2; g2; g2; g1; g1; g1];
    w(:,1)=[ w1; w2; w1; w1; w2; w1; w1; w2; w1];
    w(:,2)=[ w1; w1; w1; w2; w2; w2; w1; w1; w1];
  else
    disp('Used number of integration points not implemented');
    return
  end
  wp=w(:,1).*w(:,2);
  xsi=gp(:,1);   eta=gp(:,2);   r2=ngp*2;

%--------- shape functions -----------------------------------
  N(:,1)=(1-xsi).*(1-eta)/4;   N(:,2)=(1+xsi).*(1-eta)/4;
  N(:,3)=(1+xsi).*(1+eta)/4;   N(:,4)=(1-xsi).*(1+eta)/4;

  dNr(1:2:r2,1)=-(1-eta)/4;       dNr(1:2:r2,2)= (1-eta)/4;
  dNr(1:2:r2,3)= (1+eta)/4;       dNr(1:2:r2,4)=-(1+eta)/4;
  dNr(2:2:r2+1,1)=-(1-xsi)/4;     dNr(2:2:r2+1,2)=-(1+xsi)/4;
  dNr(2:2:r2+1,3)= (1+xsi)/4;     dNr(2:2:r2+1,4)= (1-xsi)/4;


  Ke=zeros(8,8);
  fe=zeros(8,1);
  JT=dNr*[ex;ey]';
```

```
%--------- plane stress ----------------------------------------
if ptype==1

  colD=size(D,2);
  if colD>3
    Cm=inv(D);
    Dm=inv(Cm([1 2 4],[1 2 4]));
  else
    Dm=D;
  end

  for i=1:ngp
      indx=[ 2*i-1; 2*i ];
      detJ=det(JT(indx,:));
      if detJ<10*eps
        disp('Jacobideterminant equal or less than zero!')
      end
      JTinv=inv(JT(indx,:));
      dNx=JTinv*dNr(indx,:);

      B(1,1:2:8-1)=dNx(1,:);
      B(2,2:2:8)  =dNx(2,:);
      B(3,1:2:8-1)=dNx(2,:);
      B(3,2:2:8)  =dNx(1,:);

      N2(1,1:2:8-1)=N(i,:);
      N2(2,2:2:8)  =N(i,:);

      N1 = N(i,:);
      rhogp = N1*ed_rho';
      Ke=Ke+rhogp^q*B'*Dm*B*detJ*wp(i)*t;
  end
%--------- plane strain ----------------------------------------
elseif ptype==2

  colD=size(D,2);
  if colD>3
    Dm=D([1 2 4],[1 2 4]);
  else
    Dm=D;
  end

  for i=1:ngp
      indx=[ 2*i-1; 2*i ];
      detJ=det(JT(indx,:));
      if detJ<10*eps
        disp('Jacobideterminant equal or less than zero!')
      end
      JTinv=inv(JT(indx,:));
      dNx=JTinv*dNr(indx,:);

      B(1,1:2:8-1)=dNx(1,:);
      B(2,2:2:8)  =dNx(2,:);
      B(3,1:2:8-1)=dNx(2,:);
      B(3,2:2:8)  =dNx(1,:);

      N2(1,1:2:8-1)=N(i,:);
      N2(2,2:2:8)  =N(i,:);

      N1 = N(i,:);
      rhogp = N1*ed_rho';
      Ke=Ke+rhogp^q*B'*Dm*B*detJ*wp(i)*t;
   end

else
    error('Error ! Check first argument, ptype=1 or 2 allowed')
    return
end
%-------------------------end--------------------------------
```

getdgdrhotilde-el

```
function fe=getdgdrhotilde_el(ex,ey,ep,D,ed_u,ed_rho,q)
%
%-------------------------------------------------------------
% PURPOSE
%   Computes the sensitivity of the compliance with respect to a
%   continues density field (rhotilde)
%
% INPUT:   ex = [x1 x2 x3 x4]   element coordinates
%          ey = [y1 y2 y3 y4]
%
%          ep =[ptype t ir]     element property
%                                 ptype: analysis type
%                                 ir: integration rule
%                                 t : thickness
%
%          D                    constitutive matrix
%
%     ed_u:            Element displacement vector = [u1, u2, ... u8];
%
%     ed_rho:          Element density vector = [rho1, rho2, rho3, rho4]
%
%     q:               Exponent in density field.
%
%
%
% OUTPUT: fe : Nodal sensitivites (4 x 1)
%-------------------------------------------------------------

% LAST MODIFIED: M Ristinmaa   1995-10-25
%                Eric Borgqvist  2014-03-04
% Copyright (c)  Division of Structural Mechanics and
%                Department of Solid Mechanics.
%                Lund Institute of Technology
%-------------------------------------------------------------
  ptype=ep(1); t=ep(2);  ir=ep(3);  ngp=ir*ir;

%--------- gauss points --------------------------------------
  if ir==1
    g1=0.0; w1=2.0;
    gp=[ g1 g1 ];  w=[ w1 w1 ];
  elseif ir==2
    g1=0.577350269189626; w1=1;
    gp(:,1)=[-g1; g1;-g1; g1];  gp(:,2)=[-g1;-g1; g1; g1];
    w(:,1)=[ w1; w1; w1; w1];   w(:,2)=[ w1; w1; w1; w1];
  elseif ir==3
    g1=0.774596699241483; g2=0.;
    w1=0.555555555555555; w2=0.888888888888888;
    gp(:,1)=[-g1;-g2; g1;-g1; g2; g1;-g1; g2; g1];
    gp(:,2)=[-g1;-g1;-g1; g2; g2; g2; g1; g1; g1];
    w(:,1)=[ w1; w2; w1; w1; w2; w1; w1; w2; w1];
    w(:,2)=[ w1; w1; w1; w2; w2; w2; w1; w1; w1];
  else
    disp('Used number of integration points not implemented');
    return
  end
  wp=w(:,1).*w(:,2);
  xsi=gp(:,1);  eta=gp(:,2);  r2=ngp*2;

%--------- shape functions -----------------------------------
  N(:,1)=(1-xsi).*(1-eta)/4;  N(:,2)=(1+xsi).*(1-eta)/4;
  N(:,3)=(1+xsi).*(1+eta)/4;  N(:,4)=(1-xsi).*(1+eta)/4;

  dNr(1:2:r2,1)=-(1-eta)/4;     dNr(1:2:r2,2)= (1-eta)/4;
  dNr(1:2:r2,3)= (1+eta)/4;     dNr(1:2:r2,4)=-(1+eta)/4;
  dNr(2:2:r2+1,1)=-(1-xsi)/4;   dNr(2:2:r2+1,2)=-(1+xsi)/4;
  dNr(2:2:r2+1,3)= (1+xsi)/4;   dNr(2:2:r2+1,4)= (1-xsi)/4;


  fe=zeros(4,1);
  JT=dNr*[ex;ey]';

%--------- plane stress --------------------------------------
if ptype==1

  colD=size(D,2);
  if colD>3
    Cm=inv(D);
    Dm=inv(Cm([1 2 4],[1 2 4]));
  else
    Dm=D;
  end

  for i=1:ngp
      indx=[ 2*i-1; 2*i ];
      detJ=det(JT(indx,:));
      if detJ<10*eps
        disp('Jacobideterminant equal or less than zero!')
      end
      JTinv=inv(JT(indx,:));
      dNx=JTinv*dNr(indx,:);

      B(1,1:2:8-1)=dNx(1,:);
      B(2,2:2:8)  =dNx(2,:);
      B(3,1:2:8-1)=dNx(2,:);
      B(3,2:2:8)  =dNx(1,:);

      N2(1,1:2:8-1)=N(i,:);
      N2(2,2:2:8)  =N(i,:);

      N1 = N(i,:);
      rhogp = N1*ed_rho';

      fe=fe+N1'*q*rhogp^(q-1)*(ed_u*B'*Dm*B*ed_u')*detJ*wp(i)*t;
  end
%--------- plane strain --------------------------------------
elseif ptype==2

  colD=size(D,2);
  if colD>3
    Dm=D([1 2 4],[1 2 4]);
  else
    Dm=D;
  end

  for i=1:ngp
      indx=[ 2*i-1; 2*i ];
      detJ=det(JT(indx,:));
      if detJ<10*eps
        disp('Jacobideterminant equal or less than zero!')
      end
      JTinv=inv(JT(indx,:));
      dNx=JTinv*dNr(indx,:);

      B(1,1:2:8-1)=dNx(1,:);
      B(2,2:2:8)  =dNx(2,:);
      B(3,1:2:8-1)=dNx(2,:);
      B(3,2:2:8)  =dNx(1,:);

      N2(1,1:2:8-1)=N(i,:);
      N2(2,2:2:8)  =N(i,:);

      N1 = N(i,:);
      rhogp = N1*ed_rho';

      fe=fe+N1'*q*rhogp^(q-1)*(ed_u*B'*Dm*B*ed_u')*detJ*wp(i)*t;
  end

else
   error('Error ! Check first argument, ptype=1 or 2 allowed')
   return
end
%-------------------------end--------------------------------
```

flw2i4m

```
function Me = flw2i4m(ex,ey,x)
% Me=flw2i4m(ex,ey,x)
%-------------------------------------------------------------
% PURPOSE
%  Compute the quantity: Ce=x*int(N^T*N)dA with full integration, i.e. 3x3
%  gauss points
%
% INPUT:  ex,ey;        Element coordinates
%
%         x
%
% OUTPUT: Theta :       Matix 4 x 4
%-------------------------------------------------------------

w1 = [0.5555555555555555555,0.888888888888888,0.55555555555555555555,0.5555555555555
w2 = [0.5555555555555555555,0.55555555555555555555,0.55555555555555555555,0.88888888
gp1 = -0.774596669241483;
gp2 = 0;
gp3 = -gp1;

gp_m = [gp1 gp1;
    gp2 gp1;
    gp3 gp1;
    gp1 gp2;
    gp2 gp2;
    gp3 gp2;
    gp1 gp3;
    gp2 gp3;
    gp3 gp3;];
```

```
Me = zeros(4);
for gp=1:9

     xsi = gp_m(gp,1);
     eta = gp_m(gp,2);

     dN1dxsi = 0.25*(eta-1);
     dN1deta = 0.25*(xsi-1);
     dN2dxsi = -0.25*(eta-1);
     dN2deta = -0.25*(xsi+1);
     dN3dxsi = 0.25*(eta+1);
     dN3deta = 0.25*(xsi+1);
     dN4dxsi = -0.25*(eta+1);
     dN4deta = -0.25*(xsi-1);

     dNdxsi = [dN1dxsi, dN2dxsi, dN3dxsi, dN4dxsi];
     dNdeta = [dN1deta, dN2deta, dN3deta, dN4deta];
     ax = [ex(1); ex(2); ex(3); ex(4)];
     ay = [ey(1); ey(2); ey(3); ey(4)];

    dxdxsi = dNdxsi*ax; dxdeta= dNdeta*ax;
    dydxsi = dNdxsi*ay; dydeta = dNdeta*ay;

    J = [dxdxsi, dxdeta; dydxsi, dydeta];
    detJ = det(J); %ska vara lika med Areaan av elementen*4

   N1 = 1/4*(xsi-1)*(eta-1);
   N2 = -1/4*(xsi+1)*(eta-1);
   N3 = 1/4*(xsi+1)*(eta+1);
   N4 = -1/4*(xsi-1)*(eta+1);

   N = [N1, N2,N3,N4];

   Me = Me + N'*N*detJ*x*w1(gp)*w2(gp);

end
```