Institut Supérieur de l'Aéronautique et de l'Espace

CONDENSATION BASED FINITE ELEMENT MODEL REDUCTION STRATEGY FOR STRUCTURAL OPTIMIZATION PROBLEM

Due Date of Report: 25/03/2019
Actual Submission Date: 25/03/2019

Ishaan Kakadé

Tutors: CONIGLIO Simone, MORLIER Joseph

ISAE Supaero

Contents

Abstract

The project deals with topology optimization with respect to the structure of an engine pylon on a standard aircraft. Using the finite element method, the engine is representative modeled to then project a representative load onto the design zone i.e. the pylon, which will then undergo a topological optimization. To account for engine architectures with variable thicknesses, we first integrate geometrical aspects of the engine to the decision variables of the design zone. To further simplify the representative engine, we use static condensation techniques to suppress certain degrees of freedom to reduce the time complexity associated with the optimization problem.

Keywords: topology optimization, static condensation, dynamic condensation, speed up, engine pylon.

1 Introduction

Aircraft engine, pylon and engine mounts design is often conducted by different teams and even different companies. Both engine and mounts architecture contribute to the final assembly mass and stiffness of the pylon. In a preliminary phase one would like to know if sizing the engine structure and pylon architecture simultaneously could improve the final assembly. In this context the engine structure can be considered as an input of the problem and it's FEM model is depicted in Fig. 1L, since it is often fixed by primary vein fluid-dynamic constraints. On the other hand, the pylon and engine mounts architecture can be considered as more free. The engine structure may be optimized choosing casing thicknesses; on the other hand pylon architecture can be optimized from sketches using topology optimization (see Fig. 1R).

In order to efficiently deal with this problem the condensation of engine is often required. This is simple for engine with fixed design but is not as straightforward for variable thickness case and a suitable condensation strategy should be developed here. To develop and test our strategy a very simple structure composed by 2D planar stress elements and a beam truss is studied (see Fig. 1R). This model will be employed for a simultaneous optimization of beam cross sections and 2D topology optimization. The objective will be to minimize the assembly compliance with different mass constraints in a static load cases.

To that end, a MATLAB was scripted and executed with modifications to introduce more complexity to the optimization problem while keeping an eye on the time complexity of the code. The Method of Moving Asymptotes (commonly known as MMA) is implemented under the generally outline of the widely used top88.m and top99.m code available as an open-source code at this link. In these codes, an MMA optimization algorithm by Svanberg, KTH Stockholm implemented while maintaining the SIMP (Solid Isotropic Material with Penalization) approach.

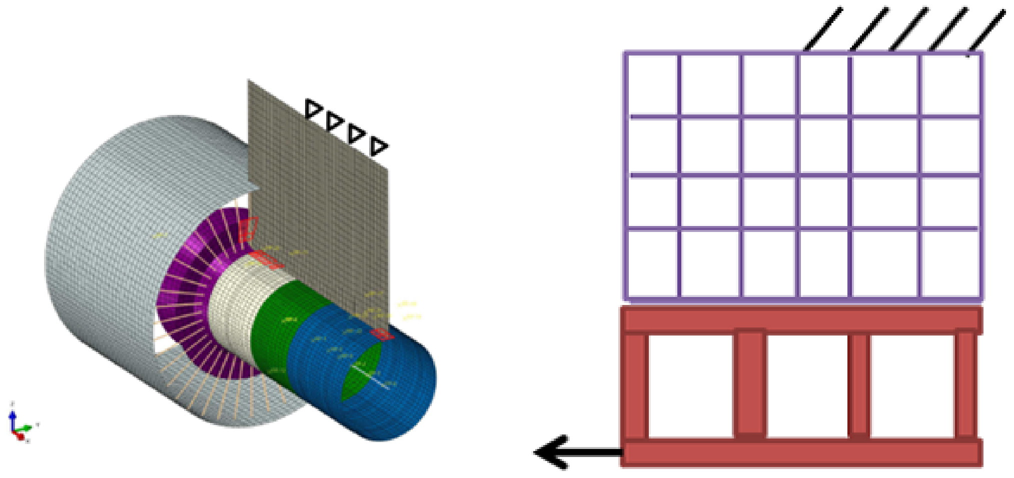


Figure 1: L - R: Left - Original FEM engine model; Right - Simplified FEM model used for testing of Condensation Schemes.

2 Background

2.1 Modified SIMP Approach

The design domain is discretized by square finite elements and a "density-based approach to topology optimization" or a "modified solid isotropic material with penalization" is followed; i.e. each element e is assigned a density $\mathrm{x_e}$ that determines its Young's modulus $\mathrm{E_e}$:

$$E_e(x_e) = E_{min} + x_e^p(E_0 + E_{min}) \qquad x_e \in [0,1] \tag{1}$$

where E0 is the stiffness of the material, E_{min} is a very small stiffness assigned to void regions in order to prevent the stiffness matrix from becoming singular, and p is penalization factor (typically p = 3) introduced to ensure black-and-white solutions. Equation (1) corresponds to the modified SIMP approach, which differs from the classical SIMP approach used in the original paper in the occurrence of the term E_{min}. In the classical SIMP approach, elements with zero stiffness are avoided by imposing a lower limit slightly larger than zero on the densities x_e. The mathematical formulation of the optimization problem reads as follows:

$$Minimize: \qquad c(x) = U^TKU = \Sigma_{e=1}^{N} E_e(x_e) u_e^T k_0 u_e \tag{2}$$

$$Subject\,to: \quad \frac{V(x)}{V_0} = f \tag{3}$$

$$KU = F \tag{4}$$

$$0 \leq x \leq 1 \tag{5}$$

Where c is the compliance, U and F are the global displacement and force vectors, respectively, K is the global stiffness matrix, ue is the element displacement vector, k0 is the element stiffness matrix for an element with unit Young's modulus, x is the vector of design variables (i.e. the element densities), N is the number of elements used to discretize the design domain, V(x) and V0 are the material volume and design domain volume, respectively, and f is the prescribed volume fraction.

2.2 Method of Moving Asymptotes

Ideally, a method for structural optimization should be flexible and general. It should be able to handle not only element sizes as design variables, but also, for instance, shape variables and material orientation angles. It should also be able to handle 'all kinds' of constraints provided only that the derivatives of the constraint functions with respect to the design variables could be calculated (analytically or numerically). Thus, the method should be able to handle general nonlinear programming problems. In addition, it should take into consideration the characteristics of structural optimization problems, e.g. usually very expensive function evaluations but still the possibility to calculate gradients. Further, the method should be 'stable' and generate a sequence of improved feasible (or almost feasible) solutions of the considered problem.

The Method of Moving Asymptotes is a mathematical programming method which has been implemented in several large systems for structural optimization (for example in OPTSYS at the Aircraft division of Saab-Scania and in OASIS at ALF-GAM Opt. AB). MMA is an iterative method. In each iteration, a convex sub problem which approximates the original prob-lem is generated and solved. An important role in the generation of these sub problems is played by a set of parameters which influence the "curvature" of the approximations and also act as "asymptotes" for the sub problem. By moving these asymptotes, between each iteration, the convergence of the overall process can be stabilized.

The MATLAB version of the author's MMA code is based on the assumption that the users optimization problem is written on the following form, where the optimization variables are $x = (x_1, \dots, x_n)^T$, $y = (y_1, \dots, y_m)^T$ and z:

$$Minimize: \qquad f_o(x) + a_o z + \Sigma_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \tag{6}$$

$$Subject\, to: \quad f_i(x) - a_i z - y_i \leq 0 \qquad i = 1, ..., m \tag{7}$$

$$x_j^{min} \leq x_j \leq x_j^{max} \qquad j = 1, ..., n \tag{8}$$

$$y_i \geq \qquad z \geq 0 \tag{9}$$

Here, $x_1 \dots x_n$ are the "true" optimization variables, while $y_1, \dots, y_m$ and z are "artificial" optimization variables; $f_0, f_1 \dots f_m$ are given, continuously differentiable, real-valued functions and constitute the constraint functions evaluated at the current values of the variable x; x_{min} and x_{max} are given real numbers which satisfy $x^j_{min} < x^j_{max}$; a_0 and a_i are given real numbers which satisfy $a_0 > 0$ and $a_i 0$; c_i and d_i are given real numbers which satisfy $c_i 0, d_i 0$ and $c_i + d_i > 0$.

The flexible nature of the MMA problem description and its MATLAB implementation by Svanberg makes it suitable for structural engineering problems which are riddled with non-linearities in constraint functions. To converge, a simple maxi-mum iteration value or KKT optimality criteria are used.

2.3 Static Condensation (Sub-Structuring)

The process of reducing the degrees of freedom is known as static condensation. The same process is also applied to dynamic problems although, in that case, it is only approximate and in general may result in large errors. The static condensation method has recently been modified for applications to dynamic problems. This method is known as the dynamic condensation method (Paz, M. 1997); its application to dynamic problems gives solution that is virtually exact.

In a beam element in the finite element method, the displacements of neighboring nodes are related by the coefficients of the stiffness matrix. It is thus possible, by some mathematical conditioning, that the displacements or rotations of some of these nodes may be suppressed and represented through other coefficients in the stiffness matrix. Sub structuring requires the reduction of nodal coordinates to allow the independent analysis of portions of the structure (sub structuring). The process of reducing the number of free displacements or degrees of freedom is known as static condensation.

A practical method of accomplishing the reduction of the number of degrees of freedom and hence the reduction of the stiffness matrix, is to identify those degrees of freedom to be condensed as secondary degrees of freedom and 'suppress' them, and to express them in terms of the remaining primary degrees of freedom as seen in Fig. 2. The relationship between secondary and primary degrees of freedom is found by establishing the static relation between them, hence the name Static Condensation Method. This relationship provides the means to reduce the number of unknowns in the system stiffness matrix equation with limited error in the resulting model. In order to describe the Static Condensation Method, assume that those secondary degrees of freedom to be reduced or condensed are arranged in the first s nodal coordinates and the remaining primary degrees of freedom are the last p nodal coordinates. With such an arrangement the stiffness equation for a structure may be written using partitioned matrices as:

$$[K]_{ss}[u]_s + [K]_{sp}[u]_p = [F]_s \tag{10}$$

$$[K]_{ps}[u]_s + [K]_{pp}[u]_p = [F]_p \tag{11}$$

Solving eqn. 11 for the vector $[u]_s$ and subsequently substituting it in eqn.11 results in:

$$[u]_s = [K]_{ss}^{-1}([F]_s - [K]_{sp}[u]_p) \tag{12}$$

$$[K]_{ps}[u]_s + [K]_{pp}[u]_p = [F]_p \tag{13}$$

$$([F]_p - [K]_{ps}[K]_{ss}^{-1}[F]_s) = ([K]_{pp} - [K]_{ps}[K]_{ss}^{-1}[K]_{sp})[u]_p \tag{14}$$

Equation 14 may conveniently be written as:

$$[\overline{F}] = [\overline{K}][u]_p \tag{15}$$

where:

$$[\overline{F}] = [F]_p - [K]_{ps}[K]_{ss}^{-1}[F]_s \tag{16}$$

$$[\overline{K}] = [K]_{pp} - [K]_{ps}[K]_{ss}^{-1}[K]_{sp} \tag{17}$$

Thus, eqn. 15 is the condensed stiffness equation relating through the condensed stiffness matrix $[\overline{K}]$, the condensed force vector $[\overline{F}]$ and the primary nodal coordinates $[u]_p$. The solution of the condensed stiffness equation 15 gives the displacement vector $[u]_p$ at the primary nodal coordinates. The displacements $[u]_s$ are fully recoverable from the primary displacements, $[u]_p$ from eqn. 12.

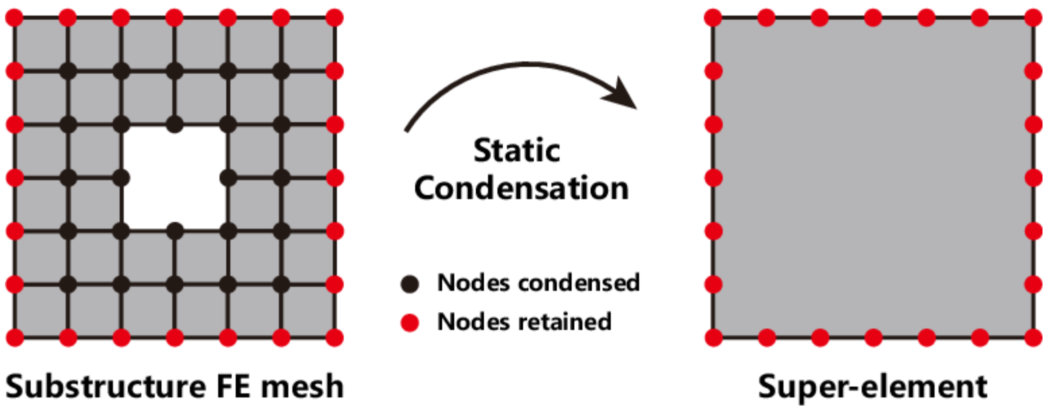


Figure 2: Illustration of static condensation of lattice substructure.

2.4 Dynamic Condensation

To understand the fundamentals of dynamic condensation, it is easy to imagine the redistribution and concentration of the mass of any object (say a cantilever beam) to fewer nodes on the body in such a way that the dynamic behavior of the body is largely unchanged and the mass matrix of the object is condensed in size, this reducing computation time in finite element problems; because the inertia effects are ignored in static condensation, the accuracy of the resulting reduced model is generally very low for dynamic problems.

To achieve reasonably accurate results, the 'master nodes' must be chosen with great care and the number of master nodes should be greater than the number of modes interested. To alleviate the limitations, the inertia effects could be partially or fully included in the condensation. The corresponding condensation approaches are generally called dynamic condensation. We can use exact condensation, classical dynamic condensation, high-order Guyan condensation, dynamic sub structuring scheme, and modal-type condensation to include these effects.

2.5 Numerical Aspects of MMA

For the structural engineering optimization problem in consideration, the problem looks as follows:

$$Minimize : f_o(x) \tag{18}$$

$$Subject\, to : f_i(x) \leq 0 \qquad i = 1, \ldots, m \tag{19}$$

$$x_j^{min} \leq x_j \leq x_j^{max} \qquad j = 1, \ldots, n \tag{20}$$

Where, from the general MMA optimization problem description in section 2.2: $a_0 = 1, a_i = 0, d_i = 0$ and $c_i = \backslash largenumber$". The reason c_i is large is because we force the variable y_i to be "expensive" such that generally, $y = 0$ for any optimal solution for f_o.

In many applications, the constraints are on the form $\sigma_i(x) \leq \sigma_i^{max}(x)$, where $\sigma_i(x)$ stands for example, a certain stress, while imax is the largest permitted value on this stress. This means that $f_i(x) = \sigma_i(x) - \sigma_i^{max}(x)$. The user should then preferably scale the constraints in such a way that $1 \leq \sigma_i^{max}(x) \leq 100$ for each i (and not $\sigma_i^{max}(x) =$ 1010). The objective function $f_0(x)$ should preferably be scaled such that $1 f_0(x) 100$ for reasonable values on the variables. The variables x_j should preferably be scaled such that $0.1 x_{max}^j - x_{min}^j 100$, for all j. Concerning the "large numbers" on the coefficients c_i (mentioned above), the user should for numerical reasons try to avoid "extremely large" values on these coefficients (like 1010). It is better to start with "reasonably large" values and then, if it turns out that not all $y_i = 0$ in the optimal solution, increase the corresponding values of c_i by a factor 100 (for example) and solve the problem again, and so on. If the functions and the variables have been scaled according to above, then "reasonably large" values on the parameters c_i could be, say, c_i = 1000 or 10000.

2.6 Karush-Kuhn-Tucker Stop Criteria

Many iterative solvers in nonlinear optimization provide natural Lagrange multipliers approximations at each iteration (Sequential Quadratic Programming, Augmented Lagrangian Methods, Interior Point Methods). These approximations are often used to stop the execution of the algorithm when the Karush-Kuhn-Tucker (KKT) optimality conditions are approximately satisfied.

3 Initial Code Configuration

3.1 Mathematics and Implementation

Here, the top88.m basis and variable nomenclature is used throughout the project. The basic flow of the code is detailed in Fig. 3. The main feature of this implementation is that the A and I have not been integrated into the decision variable vector. The advantage of a such an implementation is the consistent stiffness matrix for the representative engine FEM model that does not need to be reconstructed at each iteration rendering the code to be faster. The code is written as a function:

>>top88_plustruss_mma_nocond_internship(nelx,nely,volfrac,penal,rmin,ft)

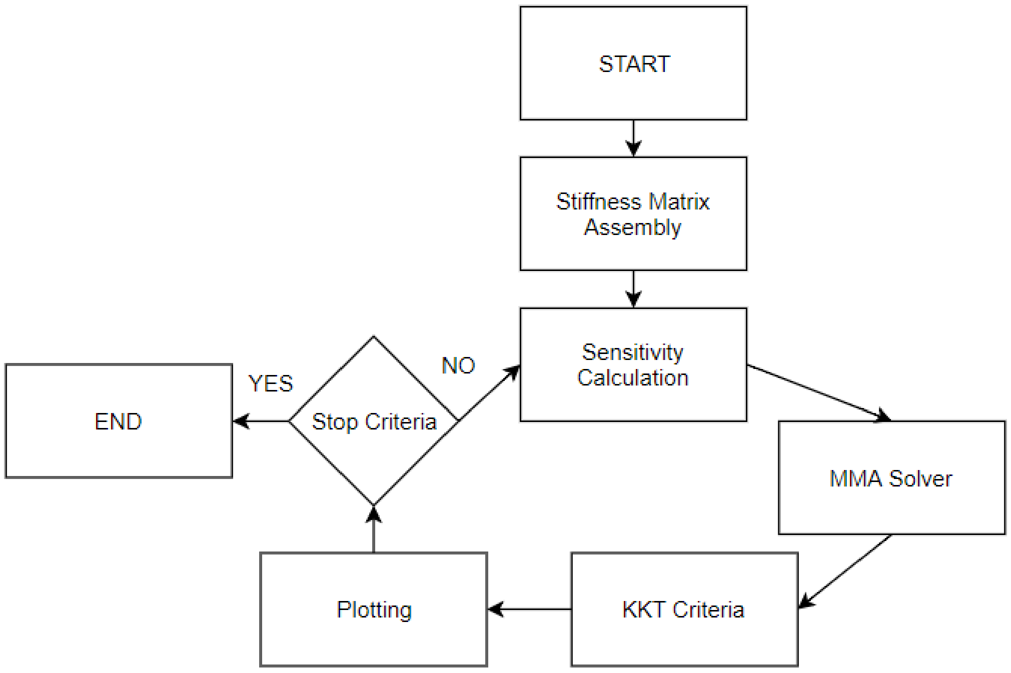


Figure 3: In order to achieve adequate speed-up, the code architecture is of vital importance and the above flow chart helps the user track the function call process and allows one to see where the most time is consumed.

The results may be reproduced by using the following line in the command box of MATLAB:

```
>>top88_plustruss_mma_nocond_internship(50,40,0.3,3,3.0,2)
```

The MATLAB code for optimization used to simulate our problem is based on the top88.m format for topology optimization with all the variables named accordingly. The change is in the optimizer which is now the MMA optimizer and the new KKT optimality criteria. We initialize the problem by calling the following function:

```
>>top88_plustruss_mma_nocond_internship(nelx,nely,volfrac,penal,rmin,ft)
```

Where *nelx* is the number of elements in the x-direction; *nely* is the number of elements in the y-direction; $volfrac$ is the desired reduction in the volume or volume fraction; *penal* is the penalty in the modified SIMP algorithm; *rmin* is the filter radius and ft, the additional argument, specifies whether sensitivity filtering ($ft = 1$) or density filtering ($ft = 2$) should be used.

For the non-design zone, the stiffness matrix and the corresponding force and displacement vectors is obtained by calling the following function:

```
>>truss_stiffness_no_condensation(nelx,nely,E,A,I)
```

Where *nelx* is the number of elements in the x-direction; *nely* is the number of elements in the y-direction; E is the elemental stiffness; A is the area of cross-section; I is the area moment of inertia. This function uses the generalized beam stiffness matrix with 3 degrees of degree per node locally and globally such that each element has 2 axial displacements, 2 vertical displacements and 2 in-plane rotations. This function returns the following vector:

```
[Kcc,Kce,Kee,Fe,Fc,Recovery_matrixc,Recovery_matrixe]
```

Where K_{cc}, K_{ee} and K_{ce} are sub-matrices used to create the 'projection matrix' that is used to affect a new stiffness on the design zone at the interface elements.

For the optimizer i.e. MMA, the following function is called in which all variables follow the exact nomenclature mentioned in section 2.2 and it prepares the the bounds and other necessary conditions to solve the convex sub-problem:

```
>>mmasub(m,n,outeriter,xval,xmin,xmax,...
xold1,xold2,f0val,df0dx,fval,dfdx,low,upp,a0,a,C,d);
```

Where m is the number of constraints; n in the number of decision variables which in our case is the number of elements; *outriter* is the iteration number; *xval* is the current value of the x vector; $xold1$ and $xold2$ are storage variables used to pass the x vector values from 1 and 2 iterations before respectively; $f0val$ is the current objective function value, $df0dx$ is the current value of the derivative of the objective function with respect to the decision variables; $fval$ is the value of the constraint functions at the present x vector value; $dfdx$ is the the value of the derivative of the constraint functions with respect to the decision variables; *low* is the lower asymptotic value from the previous iteration; *up* is the upper asymptotic value from the previous iteration, a is a_i; c is c_i and d is d_i as in section 2.2 which then returns the following vector:

```
[y,ymma,zmma,lam,xsi,eta,mu,zet,S,low,upp]
```

Where the new varibles: *ymma* and *zmma* are the y and z values respectively; lam, xsi, eta, mu and *zet* are Lagrange multipliers; S is constraint slack variables. These will be used for KKT stopping criteria. For the actual solution of the MMA convex subproblem the following function is called within the mmasub function:

```
>>subsolv(m,n,epsimin,low,upp,alfa,beta,p0,q0,P,Q,a0,a,b,c,d);
```

Where the new variables: *epsimin* is a very small number to avoid matrix singularities; $alfa$ and *beta* are 'move limits' to avoid the possibility of a division by zero while solving the subproblem; $p0, q0, P$ and Q are the values calculated in mmasub as specified by Svanberg and calculated using the differential of the constraint functions . This function returns the vector:

```
[xmma,ymma,zmma,lam,xsi,eta,mu,zet,s]
```

This is in turn returned by the mmasub function to the main script.

For the stopping optimiality criteria we use the KKT conditions by calling the following the following function:

```
>>kktcheck(m,n,x,y,z,lam,xsi,eta,mu,zet,s,xmin,xmax,df0dx,...
fval,dfdx,a0,a,c,d);
```

With the input variables as we have already seen with the other functions and it returns the following vector:

```
[residu,residunorm,residumax]
```

Where *residunorm* $=$ *kktnorm* is used as a stopping criterion along with others as seen below:

```
while kktnorm > kkttol && outit < maxoutit && change>0.001
```

Where *outit* is the current iteration value whereas *maxoutit* is the maximum permissible number of iterations and where change is the difference in the x vector values between two iterations.

3.2 The Projection Matrix

The projection matrix in this code is the mathematical tool used to 'project' or transfer the stiffness of the representative engine to the design zone and it is used extensively throughout the course of this body of work. The projection matrix changes dramatically when the structure of the problem is modified and attention needs to be drawn to this element of the code.

3.3 Results

Input Parameter	**Value**
No. of x elements	50
No. of y elements	40
Penalization	3
Filter Radius	3.0
Filter Type	2
No. of Iterations	1500
$[\overline{K}]$	sparse(4737 x 4737)
CPU Time	71124.997s
Speed Up	1

Table 1: Input - output values for the initial configuration run showing a speed-up of **1**.

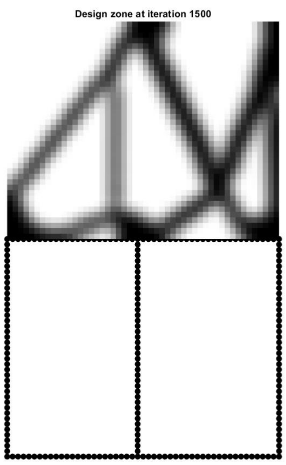


Figure 4: L: Topology Output from initial configuration with $\nu = 0.3$ which will remain as the test case for the rest of this project.

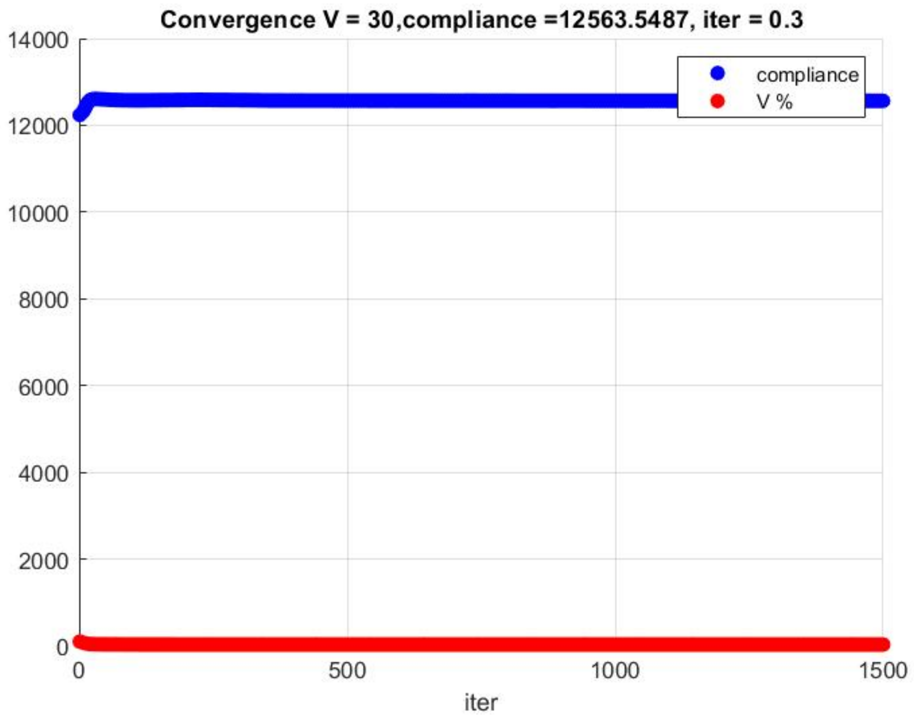


Figure 5: Compliance evolution from initial configuration.

4 Condensed Uniform Optimization

In order to acquire an optimal engine - pylon design, in the first approach, integration of the area of cross-section of the representative engine truss is required. The resulting mass ratio between the pylon and the representative engine is the primary output. Reduction of the size of the stiffness matrix is achieved for the truss that is representative of the engine. For the first case, we have reduced the stiffness matrix such that the retained or master DOFs are interface DOFs of the truss and the design zone as in fig 6

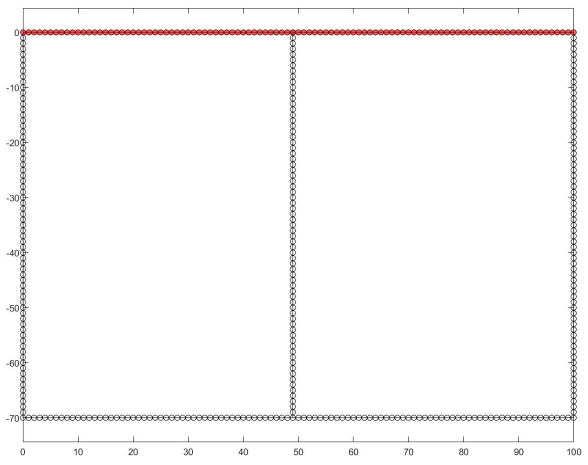


Figure 6: Representative truss as seen on MATLAB with red points showing the master nodes and hollow black points showing the slave nodes.

The stiffness matrix is split into master and slave nodes and the following condensation process is adopted:

$$\begin{bmatrix} K_{cc} & K_{ce} \\ K_{ec} & K_{ee} \end{bmatrix} \times \begin{bmatrix} u_c \\ u_e \end{bmatrix} = \begin{bmatrix} F_c \\ F_e \end{bmatrix}$$

$$B = \frac{K_{ce}}{K_{ee}} \tag{21}$$

$$K_{cond} = K_{cc} - B \times K'_{ce} \tag{22}$$

$$F_{cond} = F_c - B \times F_e \tag{23}$$

$$u_{cond} = \frac{F_{cond}}{K_{cond}} \tag{24}$$

Where K, F and U correspond to the stiffness, force and displacement matrices and the subscript *cond* represents the condensed form of the stiffness matrix. Subscript c corresponds to the master nodes and e is the slave node notation.

4.1 Mathematical Formulation

For the optimization of the condensed stiffness matrix two tasks are foreseen:

1. Integration of compliance due to force at truss into total system compliance.
2. Sensitivity of the condensed stiffness matrix with respect to the decision variable i.e. area of the beam element.

In order to achieve the requisite optimization model that is compatible with the SIMP MATLAB implementation. Thus the static condensation equations from section 2.3 are modified. For the calculation of the differentials with respect to area, we require the differential of the matrix B which may be calculated by the chain rule:

$$\frac{dB}{dA} = K_{ce}\frac{\delta K_{ee}}{\delta A} + \frac{\delta K_{ce}}{\delta A}K_{ee} \tag{25}$$

$$K_{ce}\frac{\delta K_{ee}}{\delta A} = -K_{ee}^{-1}K_{ce}K_{ee}^{-1} \tag{26}$$

Here, the differential of the slave sub-matrix with respect to area is given by K_{ee_A}

And the differential of the cross-dependency sub-matrix with respect to area is given by K_{ce_A}

Hence,

$$\frac{dK_{cond}}{dA} = \frac{dK_{cc}}{dA} + \frac{dB}{dA}K_{ce} + B\frac{dK_{ce}}{dA} \tag{27}$$

For the truss compliance, we define a new quantity:

$$u_0 = K_{ee}^{-1} \times F_e \tag{28}$$

Therefore, for the calculation of the truss compliance for a predefined force vector at slave node x, we get:

$$C_{truss} = F_e(u_0(x) - B(x,:)u_c) \tag{29}$$

4.2 Constraints and Optimization Formulation

The problem has two distinct sets of constraints, one of the topology design variables and the other of the area of cross-section of the beam truss elements. To drive the former, we use the derivatives and objective functions in the proven top88.m code whereas for the latter we use use total mass as a driver for the optimization problem. The derivatives of the two sets of the design variables are mentioned below:

1. For the design zone decision variables,we have:

$$\frac{\delta c}{\delta x_e} = -px_e^{p-1}(E_0 - E_{min})u_e^T k_0 u \tag{30}$$

$$\frac{\delta V}{\delta x_e} = 1 \tag{31}$$

$$\frac{\delta M_{total}}{\delta x_e} = \frac{1}{\rho} \tag{32}$$

2. For the area of cross-section we have from the chain rule:

$$\frac{\delta C_{truss}}{\delta A} = F_e(\frac{\delta K_{ee}}{\delta A} - \frac{\delta B}{\delta A}u_c) \tag{33}$$

3. The combined mass constraint of the truss-design zone system is:

$$M_{total} = M_{DZ} + M_{truss} \tag{34}$$

$$M_{total} = mean(x) \times rho \times N_x \times N_y + (3N_y + 2N_x) \times rho \times L \times A \tag{35}$$

Therefore:

$$\frac{\delta M_{total}}{\delta A} = (3N_y + 2N_x) \times rho \times L \tag{36}$$

The final optimization problem is as follows:

Minimize:

$$c(x) = U^T KU = \Sigma_{e=1}^N E_e(x_e)u_e^T k_0 u_e \tag{37}$$

$$M(x, A) = mean(x) \times rho \times N_x \times N_y + (3N_y + 2N_x) \times rho \times L \times A \tag{38}$$

Subject to:

$$\frac{V(x)}{V_0} = f_v \tag{39}$$

$$\frac{M(x,A)}{M_{target}} = f_m \tag{40}$$

$$KU = F \tag{41}$$

$$0 \leq x \leq 1 \tag{42}$$

$$0 \leq A \leq 2 \tag{43}$$

4.3 Implementation of Code

In order to optimize the code for time complexity, care was taken to ensure that the fastest methods were implemented to unleash the full advantage of condensation. The sensitivities are introduced as seen in Fig. 7 and the stiffness matrix assembly is not within the optimization loop and changes the function call loop to Fig. 8.

Some challenges faced in this part of the optimization are as follows:

1. To avoid singularities, the factor that matters the most is the scaling between the stiffness of the truss elements and the design zone variables and by verifying the eigen values of the stiffness matrix after projection.
2. The compliance of the truss which is based on the displacement of the nodes of the beam truss is calculated at the point of action of the force but in our case that node is condensed and suppressed but this value is required to compute the total compliance of the system, which is ultimately the objective function of he total optimization problem. It is recovered from eq. 29. Therefore it is:
$$C = C_{DZ} + C_{truss} \tag{44}$$
3. The choice of the master and slave nodes is of vital importance to represent the stiffness of the truss. Here, we choose nodes such that the projection is simple and leads us to choose the interface nodes.

From Tab. 4, it is evident that the code is sped up significantly. The simultaneous evaluation of the optimal engine (represented by a beam truss) and the engine pylon that used to take > 70000 seconds now takes < 1600 seconds. The reasons for the speed up may be attributed as follows:

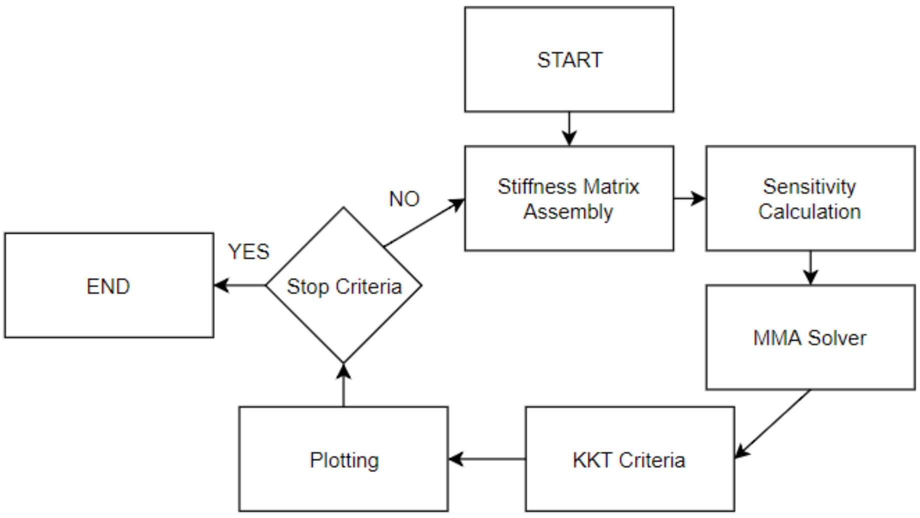
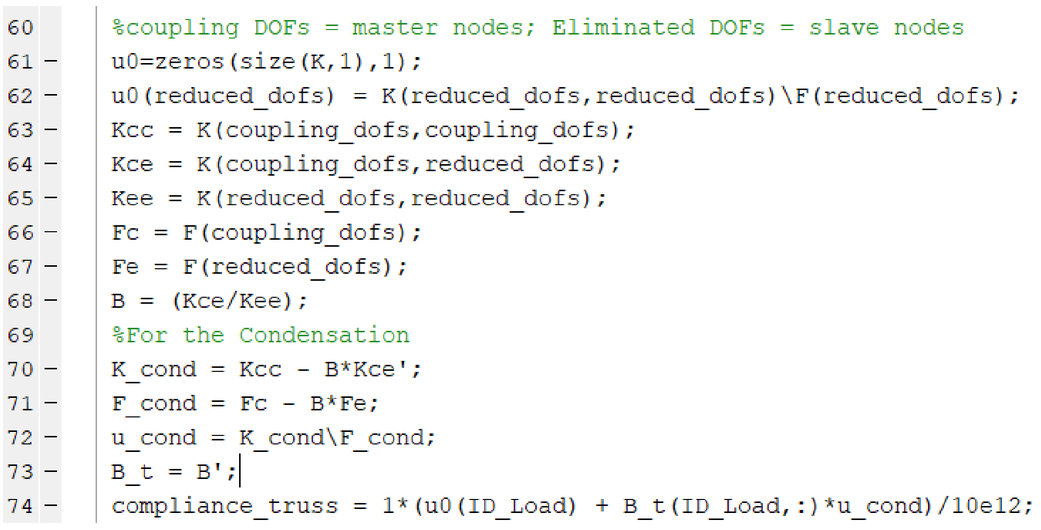


Figure 7: In order to achieve adequate speed-up, the code architecture is of vital importance and the above flow chart helps the user track the function call process and allows one to see where the most time is consumed.



```
%coupling DOFs = master nodes; Eliminated DOFs = slave nodes
u0=zeros(size(K,1),1);
u0(reduced_dofs) = K(reduced_dofs,reduced_dofs)\F(reduced_dofs);
Kcc = K(coupling_dofs,coupling_dofs);
Kce = K(coupling_dofs,reduced_dofs);
Kee = K(reduced_dofs,reduced_dofs);
Fc = F(coupling_dofs);
Fe = F(reduced_dofs);
B = (Kce/Kee);
%For the Condensation
K_cond = Kcc - B*Kce';
F_cond = Fc - B*Fe;
u_cond = K_cond\F_cond;
B_t = B';
compliance_truss = 1*(u0(ID_Load) + B_t(ID_Load,:)*u_cond)/10e12;
```

Figure 8: Modifications to the code used to calculate the sensitivities of the design variables with respect to the area of cross-section of the beam elements. Some scaling of values is required for numerical purposes.

4.4 Results

1. MATLAB is especially expensive in iterative processes that require accessing a large matrix and can be uneconomical. The major reduction in the size of the stiffness matrix results in the

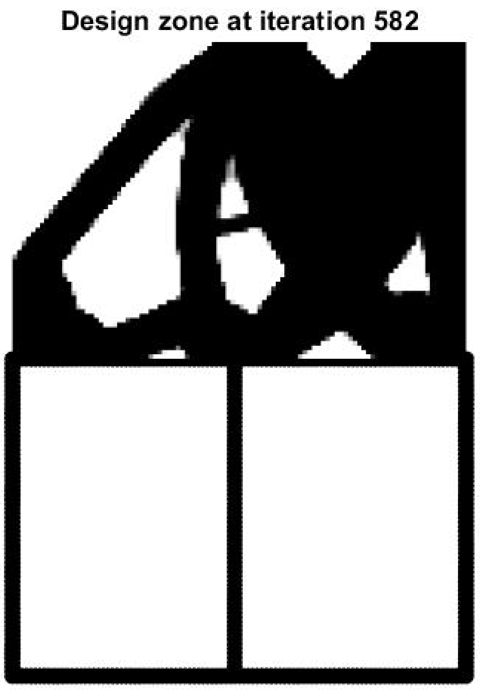


Figure 9: Final converged design zone for the condensed beam truss.

speed up of the code.

2. The evaluation of the derivatives is done only once at every iteration and the derivatives are used in other functions as when required rather than the total re-calculation of the derivatives.
3. Since the stiffness matrix is reduced in size, the calculation of the derivatives is made much faster.
4. Faster KKT convergence criteria allows a faster convergence of the result.

The result may be recreated used the following command in the file:

```
>>top88_plustruss_mma_cond_internship(50,40,0.3,500,3,3.0,2)
```

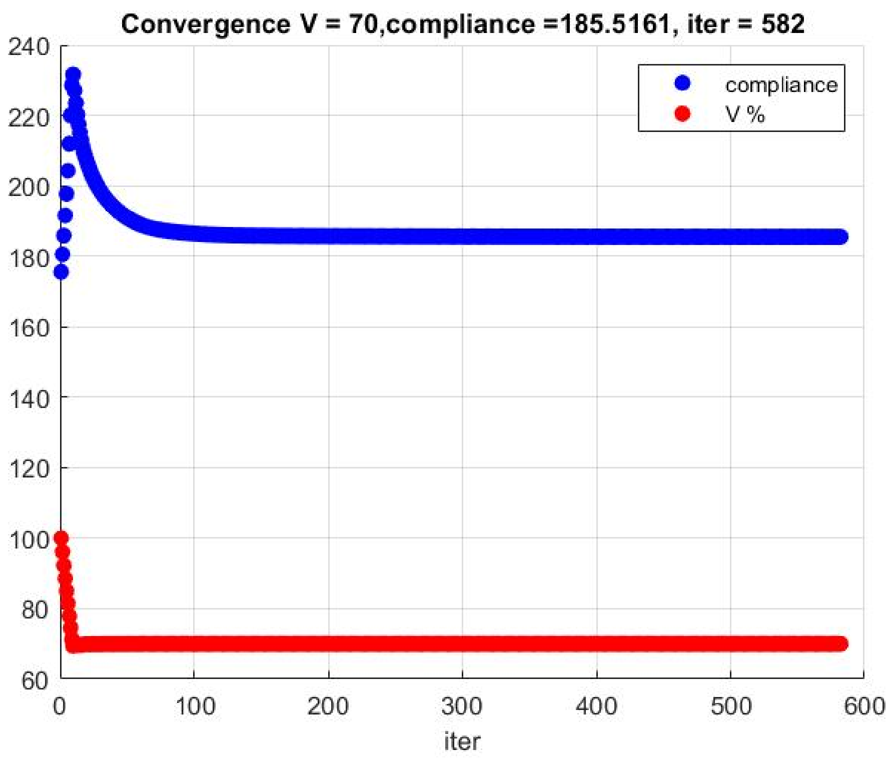
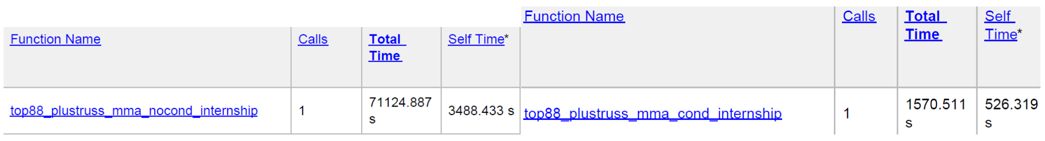


Figure 10: Compliance evolution for condensed truss optimization.



Function Name	Calls	Total Time	Self Time*
top88_plustruss_mma_nocond_internship	1	71124.887 s	3488.433 s

Function Name	Calls	Total Time	Self Time*
top88_plustruss_mma_cond_internship	1	1570.511 s	526.319 s

Figure 11: CPU time used for non-condensed (L) and condensed (R) truss as seen using time function of MATLAB.

Input Parameter	**Value**
No. of x elements	50
No. of y elements	40
Penalization	3
Filter Radius	3.0
Starting A	5
Filter Type	2
Target Mass Ratio	0.66
No. of Iterations	582
$[\overline{K}]$	sparse(277 x 277)
CPU Time	1570.511s
Speed Up	45.30

Table 2: Input and output values for the condensed truss using uniform area optimization run showing a speed up of **45.30**.

5 Condensed Differential Optimization

To take things further, the finite element problem that underlies the optimization is modified to pose the problem in a more complex way. The beam truss is then modified to have different areas of cross-section for horizontal and vertical parts of the truss.

5.1 Mathematical Formulation

Identical to section 4.1.1, the mathematical formulation of the optimization problem remains identical whereas the finite problem in this case requires rather major modifications. This modifications follow the logic of matrix separation based on the location of the elements (of the matrix) corresponding to the position of the node of truss. Say the matrix K_{truss} now consists of two parts K_v, representing the vertical beams of the beam truss and K_h, representing the horizontal beams such that:

$$\begin{bmatrix}K_{truss}\end{bmatrix} = \begin{bmatrix}K_h \\ K_v\end{bmatrix}$$

These matrices need to be treated independently in the calculation of the sensitivities of the areas of cross section:

$$\frac{dB}{dA_h} = K_{ce}^h \frac{\delta K_{ee}^h}{\delta A_h} + \frac{\delta K_{ce}^h}{\delta A_h} K_{ee}^h \tag{45}$$

$$K_{ce}^h \frac{\delta K_{ee}^h}{\delta A_h} = -K_{ee}^{h\ -1} K_{ce}^h K_{ee}^{h\ -1} \tag{46}$$

$$\frac{dB}{dA_v} = K_{ce}^v \frac{\delta K_{ee}^v}{\delta A_v} + \frac{\delta K_{ce}^v}{\delta A_v} K_{ee}^v \tag{47}$$

$$K_{ce}^v \frac{\delta K_{ee}^v}{\delta A_v} = -K_{ee}^{v\ -1} K_{ce}^v K_{ee}^{v\ -1} \tag{48}$$

Here, the sub-matrix with respect to horizontal area is given by K_{xx}^h and the sub matrices of the vertical matrices are given by K_{xx}^v

Hence,

$$\frac{dK_{cond}}{dA_h} = \frac{dK_{cc}^h}{dA_h} + \frac{dB}{dA_h} K_{ce}^h + B \frac{dK_{ce}^h}{dA_h} \tag{49}$$

$$\frac{dK_{cond}}{dA_v} = \frac{dK_{cc}^v}{dA_v} + \frac{dB}{dA_v} K_{ce}^v + B \frac{dK_{ce}^v}{dA_v} \tag{50}$$

Truss compliance is recovered in the same way as in the previous exercise.

5.2 Constraint and Optimization Formulation

Essentially, the optimization problem remains unchanged in that the only addition to the structure is the parallel sensitivity calculation with respect to horizontal and vertical parts of the beam truss.

1. For the design zone decision variables,we have:

$$\frac{\delta c}{\delta x_e} = -p x_e^{p-1}(E_0 - E_{min}) u_e^T k_0 u \tag{51}$$

$$\frac{\delta V}{\delta x_e} = 1 \tag{52}$$

$$\frac{\delta M_{total}}{\delta x_e} = \frac{1}{\rho} \tag{53}$$

2. For the area of cross-section we have from the chain rule:

$$\frac{\delta C_{truss}}{\delta A_h} = F_e(\frac{\delta K_{ee}^h}{\delta A_h} - \frac{\delta B}{\delta A_h} u_c) \tag{54}$$

$$\frac{\delta C_{truss}}{\delta A_v} = F_e(\frac{\delta K_{ee}^v}{\delta A_v} - \frac{\delta B}{\delta A_v} u_c) \tag{55}$$

3. The combined mass constraint of the truss-design zone system is:

$$M_{total} = M_{DZ} + M_{truss}^h + M_{truss}^v \tag{56}$$

$$M_{total} = mean(x) \times rho \times N_x \times N_y + 3N_y \times rho \times L \times A_v + 2N_x \times rho \times L \times A_h \tag{57}$$

Therefore:

$$\frac{\delta M_{total}}{\delta A_h} = 2N_x \times rho \times L \tag{58}$$

$$\frac{\delta M_{total}}{\delta A_v} = 3N_y \times rho \times L \tag{59}$$

The final optimization problem is as follows:

Minimize:

$$c(x) = U^T K U = \Sigma_{e=1}^N E_e(x_e) u_e^T k_0 u_e \tag{60}$$

$$M(x, A_h, A_v) = mean(x) \times rho \times N_x \times N_y + 3N_y \times rho \times L \times A_v + 2N_x \times rho \times L \times A_h \tag{61}$$

Subject to:

$$\frac{V(x)}{V_0} = f_v \tag{62}$$

$$\frac{M(x, A_h, A_v)}{M_{target}} = f_m \tag{63}$$

$$KU = F \tag{64}$$

$$0 \leq x \leq 1 \tag{65}$$

$$0 \leq A_h \leq 2 \tag{66}$$

$$0 \leq A_v \leq 2 \tag{67}$$

The mass constraint is determined in the mathematical form in eq. 63 by the mass share although, due to difficulties associated with implementing the code, the areas A_h and A_v are used as target values and then used in the constraint functions to determine the masses of the system at each iteration.

5.3 Sensitivity Analysis

From Tab. 4 it is becomes necessary to study the effect of the choice of a starting point in the optimization problem on the final topology optimization output.

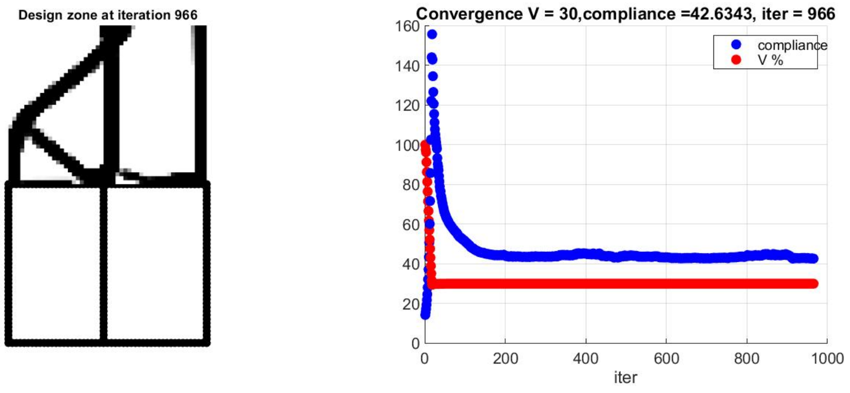


Figure 12: $A_h = 0.62 A_v = 1.432$.

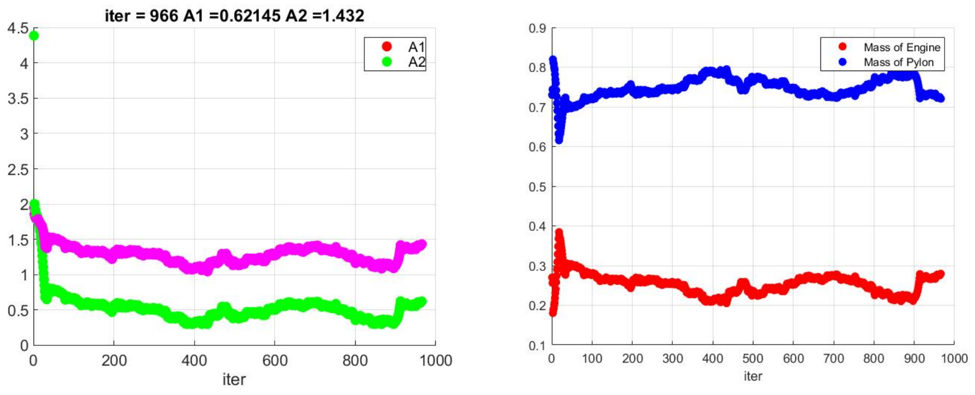


Figure 13: $A_h = 0.62 A_v = 1.432$.

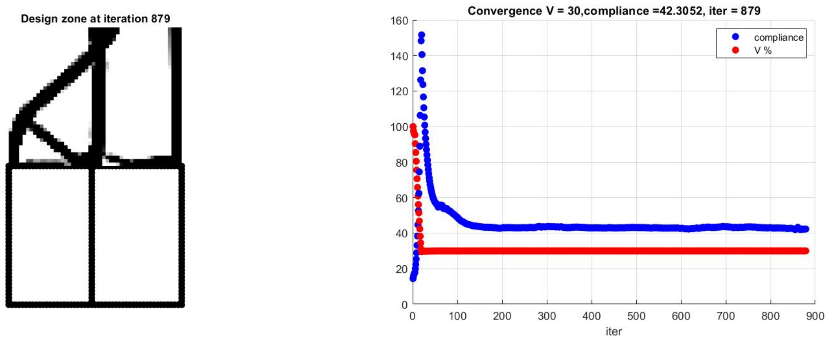


Figure 14: $A_h = 0.707 A_v = 1.35$.

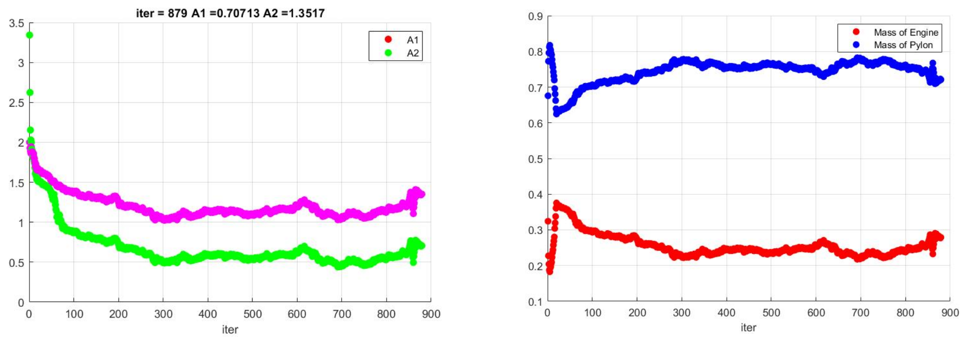
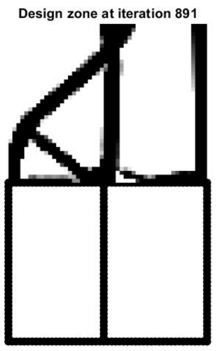


Figure 15: $A_h = 0.707 A_v = 1.35$.



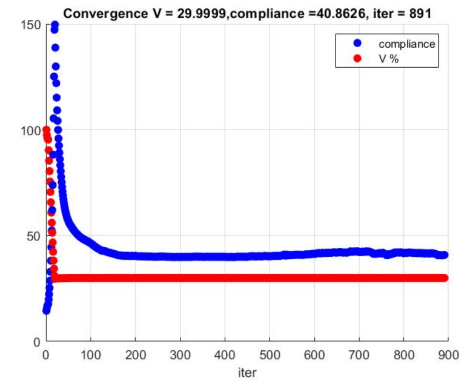
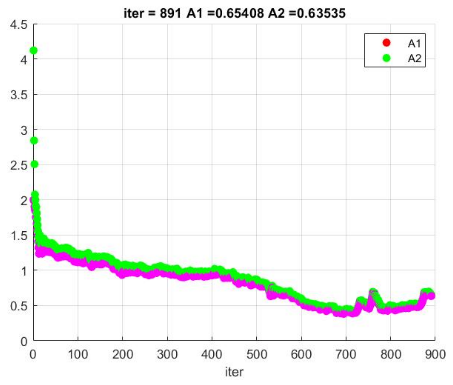


Figure 16: $A_h = 0.654 A_v = 0.635$.



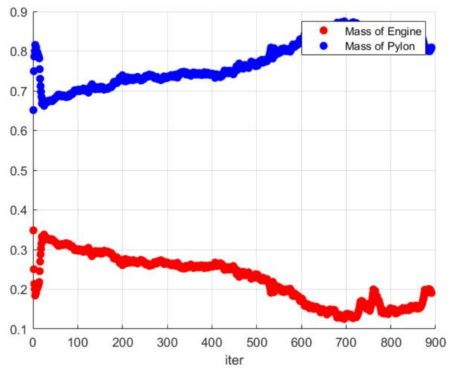


Figure 17: $A_h = 0.654 A_v = 0.635$.

$A_h^{initial}$	$A_v^{initial}$	$\%M_{Engine}$	$\%M_{Pylon}$	A_h^{final}	A_v^{final}
4	1	40%	60%	0.59	1.305
5	2	33%	66%	0.62	1.432
6	3	29%	71%	0.707	1.35
6.5	3.5	20%	80%	0.654	0.635

Table 3: Within a limited range, the optimization is not too sensitive to change in starting values but outside this range, the problem may behave erratically.

5.4 Parametric Analysis

In order to see how the optimization problem behaves with different input parameters the volume fraction required at the output:

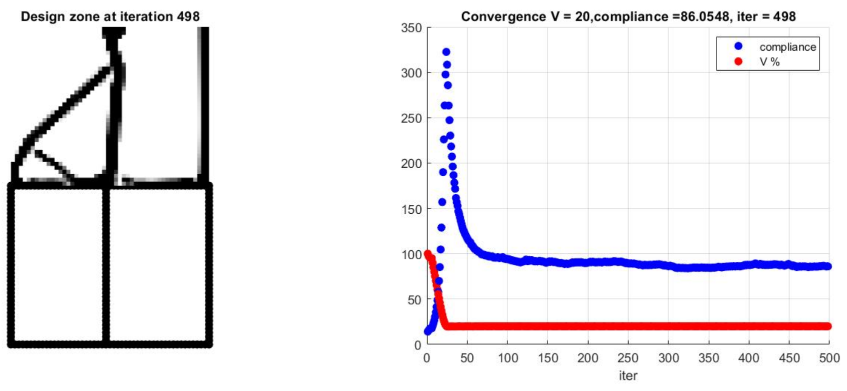


Figure 18: The final layout and statistics for $\nu = 0.2$ after 498 iterations showing a significant speed-up of **35.04**.

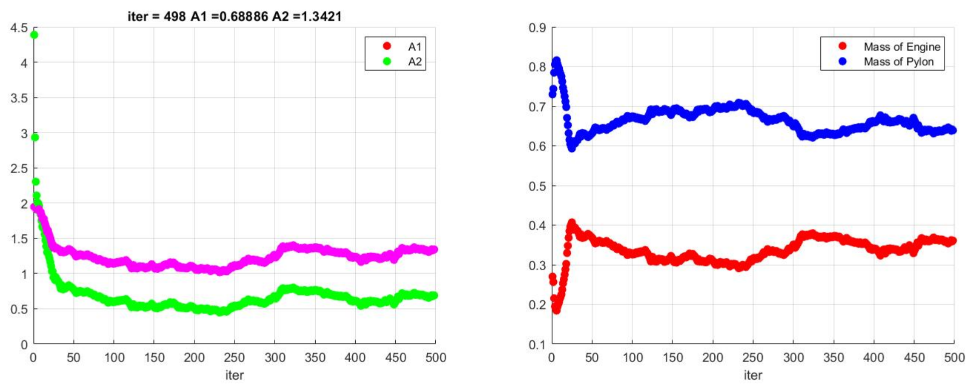
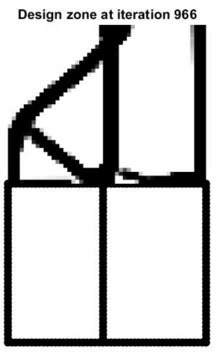


Figure 19: The final areas and mass distribution for $\nu = 0.2$ after 498 iterations showing a significant speed-up of **35.04**.

As seen in Fig. 19, the right-most image shows the variation of the mass share between the pylon (blue) and the representative engine (red). Here, the pylon mass share is initially decreased (due to volume reduction constraints) and then further increased as the mass constraints kick-in as the area of the truss beam elements are reduced and the pylon is reinforced to reduced the compliance of the truss. The left most image shows a plausible topological distribution of a pylon and the stabilization of the compliance of the system is seen. The final areas for A_h and A_v is 0.40603 and 1.804 respectively which seems to conform with the idea that the code chose to reduced axial force bearing truss sections before reducing the bending force bearing truss sections. The mass share at the end of the optimization process stood at 66% for the pylon and 33% for the representative engine.



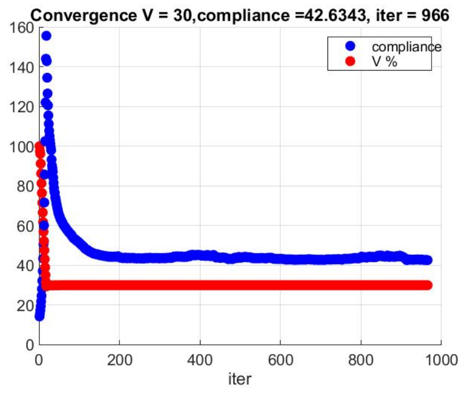
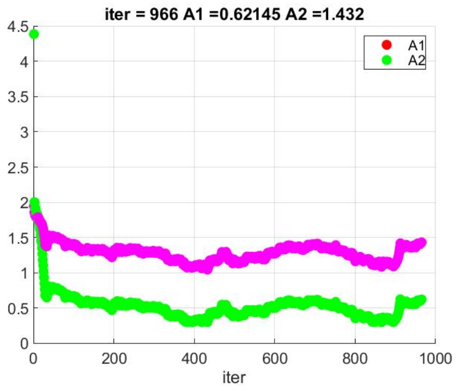


Figure 20: L: The final layout and statistics for $\nu = 0.3$ after 968 iterations showing a significant speed-up of **24.430**.



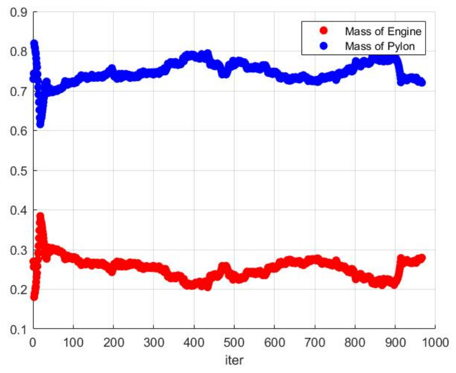


Figure 21: L: The final area and mass distribution for $\nu = 0.3$ after 968 iterations showing a significant speed-up of **24.430**.

The behaviour of the simulation shown in Fig. 21 is similar to Fig. 19, but what changes is the final areas for A_h and A_v and they are 0.62145 and 1.432 respectively with an engine to pylon mass share of 29% - 71%.

Fig. 23 represents a scenario where a large mass is requested as a target value for the pylon and the result is as expected with a stronger pylon, the structure of the engine may be relaxed further and the areas for A_h and A_v are 0.80893 and 0.80964 respectively. The convergence

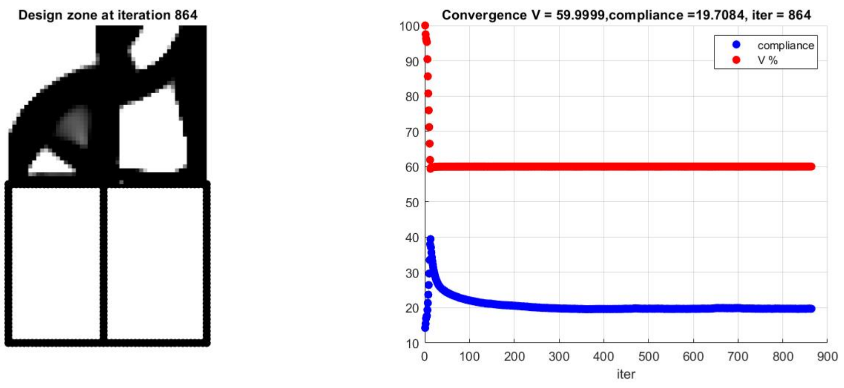


Figure 22: L: The final layout and statistics for $\nu = 0.6$ after 864 iterations showing a significant speed-up of **25.868**.

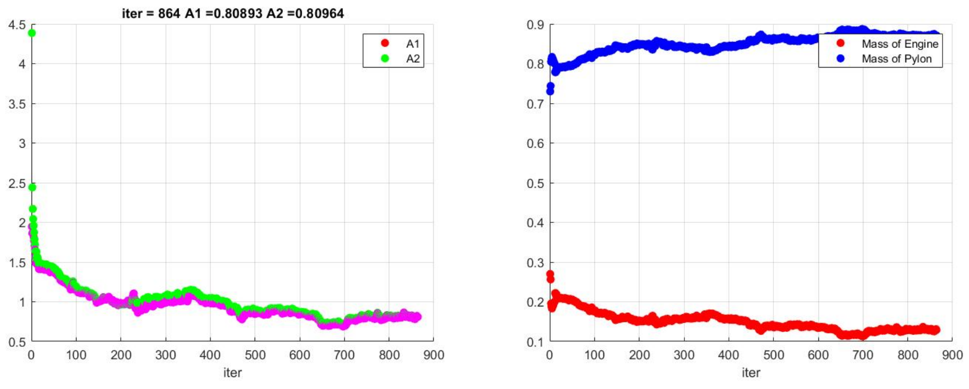


Figure 23: L: The final area and mass distribution for $\nu = 0.6$ after 864 iterations showing a significant speed-up of **25.868**.

of the areas is expected as the pylon has a higher stiffness and the optimization is primarily focused on reduction of the area of the beam truss elements. Engine to pylon mass share stood at 12% - 88% as the function call demanded a high volume fraction in the pylon.

5.5 Results

The following results are for the standard test case showing the various parameters of interest graphically. The results may be reproduced using the following command on MATLAB:

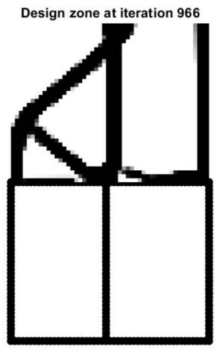
```
>>top88_plustruss_mma_cond_internship_by_set(50,40,1,0.3,3,...
3,1.5,1.5,1.5,3,3.0,2)
```

The results obtained with this technique show possible topology layouts of the pylons that could potentially lead to designs for the pylon as seen in Fig. 25.

Input Parameter	**Value**
No. of x elements	50
No. of y elements	40
Penalization	3
Filter Radius	3.0
Starting A_h	5
Starting A_v	2
Filter Type	2
Target Mass Ratio	0.66
No. of Iterations	582
$[\overline{K}]$	sparse(277 x 277)
CPU Time	2911.350s
Speed Up	24.430

Table 4: Input and output values for the condensed truss using differential area optimization configuration run with a speed up of **24.430**.

As seen in Fig. 25, the image shows the variation of the mass share between the pylon (blue) and the representative engine (red). Here, the pylon mass share is initially decreased (due to volume reduction constraints) and then further increased as the mass constraints kick-in as the area of the truss beam elements are reduced and the pylon is reinforced to reduced the compliance of the truss. The left most image shows a plausible topological distribution of a pylon and the



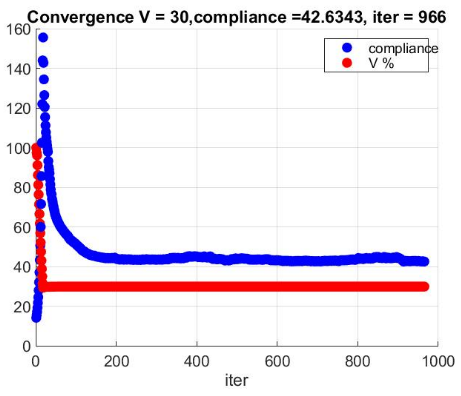
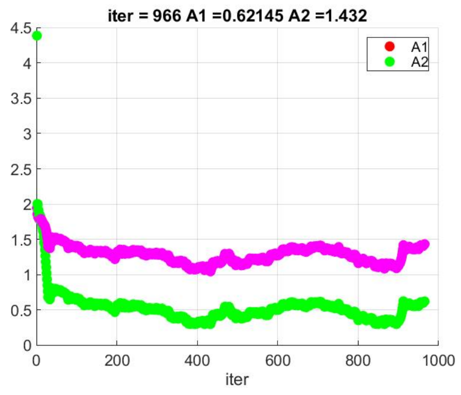


Figure 24: L: The final layout for $\nu = 0.3$ (standard test case) after 966 iterations showing a significant speed-up. R: Compliance evolution for this case.



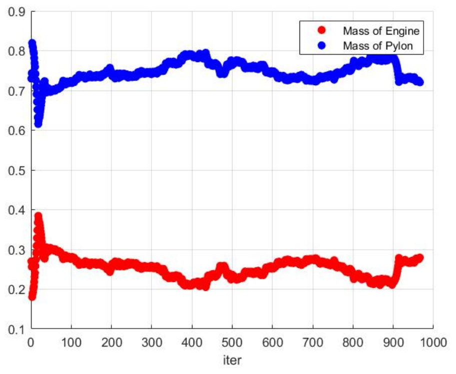


Figure 25: L: The evolution of the area of cross-section to try and reduce the area of the beam element truss R: Mass share between engine and truss.

stabilization of the compliance of the system is seen. The final areas for A_h and A_v is 0.40603 and 1.804 respectively which seems to conform with the idea that the code chose to reduced axial force bearing truss sections before reducing the bending force bearing truss sections. The mass share at the end of the optimization process stood at 66% for the pylon and 33% for the representative engine. Not only are the mass target criteria met but also the result is obtained with a speed-up (from the initial case) of **24.430**.

6 Conclusion

The final goal of the project was to analyze and compare the three approaches to the optimization problem and to see if significant speed-up was achievable. Tab. 5 clearly summarizes the speed-up in the various techniques used.

Technique	**Time Complex**	**Iterations**	**Speed Up**	$[\overline{K}]$ **Size**
Initial Code	71124.997s	1500	1	4737 x 4737
Uniform Area	1570.511s	582	45.30	277 x 277
Differential Area	2911.350s	582	24.430	277 x 277

Table 5: Final Comparative Table for Optimization of Condensed Beam Truss with speed-up values as seen on MATLAB code.

The following are the primary conclusions of the project:

1. In order to represent finite element model of an engine,a simplified beam element truss was created with a design zone used to simulate the presence of a engine pylon.
2. Using the top88 optimization scheme, the stiffness added by the truss was 'projected' using a projection matrix to the design zone.
3. Using Guyan Static Condensation Techniques, the stiffness matrix that defines the structure of the representative engine is reduced in size by choosing the primary nodes as the interface nodes and the secondary nodes as the remaining nodes.
4. Initially, the area of the beam truss is integrated into the optimization problem and the resulting plausible solution and speed up of **45.30** is achieved.
5. To progressively complex the problem, the areas of the vertical parts and the horizontal parts are optimized separately and the resulting plausible design zone and speed up of **24.43** is achieved.

7 REFERENCES

[1] Svanberg K. (1987), The Method of Moving Asymptotes – A New Method for Structural Optimization, International Journal for Numerical Methods in Engineering, Vol. 24 (No. 2), pp. 359 -373, https://doi.org/10.1002/nme.1620240207

[2] L.D. Flippen Jr, (1994), A Theory of Condensation Model Reduction, Computers Math. Application, Vol. 27 (No. 2), pp. 9 - 40.

[3] Paz M., Leigh W. (2001); Static Condensation and Substructuring. In: Integrated Matrix Analysis of Structures. Springer, Boston, MA

[4] Zu-Qing Qu, (2004), Model Reduction Techniques, London: Springer-Verlag.

[5] Zu-Qing Qu, Zhi-Fangfu, (2000), An Iterative Method For Dynamic Condensation Of Structural Matrices, Mechanical Systems and Signal Processing, Vol. 14 (No. 4), pp. 667 - 678.

[6] Andreassen, E., Clausen, A., Schevenels, M. et al. Struct Multidisc Optim (2011) 43: 1. https://doi.org/10.1007/s00158-010-0594-7

[7] Zhu, J.H., Zhang, WH. Xia, L. Arch Computat Methods Eng (2016) 23: 595. https://doi.org/10.1007/s11831-015-9151-2

[8] Deaton, J.D. Grandhi, R.V. Struct Multidisc Optim (2014) 49: 1. https://doi.org/10.1007/s00158-013-0956-z

[9] L. Xia, J. Zhu, Q. Zhang, (October, 2011), ''Some Recent Advances in Integrated Layout Design of Multicomponent Systems", Journal of Mechanical Design Vol.133 / 104503 -1 – 104503 – 15.

[10] J. H. Zhu, W. H. Zhang, (October, 2009), ''Integrated Layout Design of Supports and Structures", Computer Methods in Applied Mechanics and Engineering, 199 (2010) 557 - 569.

8 APPENDIX

8.1 Initial Code

Contents

```
function x=top88_plustruss_mma_nocond_internship(nelx,nely,volfrac,penal,rmin,ft)

close all
filename = 'Topology_Optimization_0.gif';
```

MATERIAL PROPERTIES

```
E0 = 1;
Emin = 1e-9;
nu = 0.3;
P=4;
Sl=1;
```

PREPARE FINITE ELEMENT ANALYSIS

```
A11 = [12  3 -6 -3;  3 12  3  0; -6  3 12 -3; -3  0 -3 12];
A12 = [-6 -3  0  3; -3 -6 -3 -6;  0 -3 -6  3;  3 -6  3 -6];
B11 = [-4  3 -2  9;  3 -4 -9  4; -2 -9 -4 -3;  9  4 -3 -4];
B12 = [ 2 -3  4 -9; -3  2  9 -2;  4  9  2  3; -9 -2  3  2];
% D=E0/(1-nu^2)*[1 nu 0;nu 1 0; 0 0 (1-nu)/2];
% B=1/2*[-1 0 1 0 1 0 -1 0;0 -1 0 -1 0 1 0 1;-1 -1 -1 1 1 1 1 -1];
% DB=D*B;
% B=1/2*[-1 0 1 0 1 0 -1 0;0 -1 0 -1 0 1 0 1;-1 -1 -1 1 1 1 1 -1];
% DB=D*B;
% Cvm=[1 -0.5 0;-0.5 1 0;0 0 3];
% Sel=DB'*Cvm*DB;
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
% fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
fixeddofs = [2*(fix(nelx/2)*(nely+1)+1:(nely+1):((nelx)*(nely+1)+1))-1;2*(fix(nelx/2)*(nely+1)+1:(nely+1):((nelx)*(nely+1)+1))];
fixeddofs=fixeddofs(:);
```

PREPARE FILTER

```
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
```

INITIALIZE ITERATION

x = repmat(1,nely,nelx); xPhys = x; loop = 0; change = 1;

INITIALIZE ITERATION

```
x = repmat(1,nely,nelx);
```

```
xPhys = x;
loop = 0;
change = 1;
m = 1;
n = length(xPhys(:));
epsimin = 0.0000001;
eeen    = ones(n,1);
eeem    = ones(m,1);
zeron   = zeros(n,1);
zerom   = zeros(m,1);
xval    = xPhys(:);
xold1   = xval;
xold2   = xval;
xmin    = zeron;
xmax    = eeen;
low     = xmin;
upp     = xmax;
C       = 1000*eeem;
d       = 0*eeem;
a0      = 1;
a       = zerom;
outeriter = 0;
maxoutit  = 1500;
kkttol  =0.001;
%
%%%% The iterations start:
kktnorm = kkttol+10;
% kktnorm = kkttol;
outit = 0;
change=1;
```

START ITERATION

```
while kktnorm > kkttol && outit < maxoutit && change>0.001
```

Generate the truss stiffness matrix

```
    A=0.3;
    I=0.06;
    [Kcc,Kce,Kee,Fe,Fc,Recovery_matrixc,Recovery_matrixe]=truss_stiffness_no_condensation(nelx,nely,E0,A,I);
```

Generate the Projection matrix, load vector and stiffness matrix ready for assembly

```
    coupling_nodes=[nely+1:nely+1:(nely+1)*(nelx+1)];
    coupling_dofs=[coupling_nodes*2-1;coupling_nodes*2];
    coupling_dofs=coupling_dofs(:);
    Pr = sparse(coupling_dofs,1:2*(nelx+1),ones(1,2*(nelx+1)),2*(nelx+1)*(nely+1),2*(nelx+1));
    Kt = [Pr*Kcc*Pr' Pr*Kce;Kce'*Pr' Kee];
    Kt=(Kt+Kt')/2;
    Ft=[Pr*Fc;Fe];
    Rt=[Recovery_matrixc*Pr' zeros(size(Recovery_matrixc,1),size(Recovery_matrixe,2))
        zeros(size(Recovery_matrixe,1),size(Pr',2)) Recovery_matrixe];
    U = zeros(2*(nely+1)*(nelx+1)+length(Fe),1);
    F=U;
    Lambda=[U,U];
    F=F+Ft;
    alldofs = [1:length(F)];
    freedofs = setdiff(alldofs,fixeddofs);

    % original_force=-1;
    % Gamma=sparse(reshape( repmat(1:(nelx+1),2,1),[],1),reshape([1:(nelx+1);(1:(nelx+1))+(nelx+1)],[],1),reshape([ones(1,nelx+1);-ones(1,nelx+1)],[],1))
    % Rt=Gamma*Rt;
    outit   = outit+1;
    outeriter = outeriter+1;
    if ft == 1
        xPhys(:)=xval;
    elseif ft == 2
        xPhys(:) = (H*xval(:))./Hs;
    end
```

FE-ANALYSIS

```
    sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
    K0 = sparse(iK,jK,sK,size(Kt,1),size(Kt,2));

    K=K0+Kt;
    K = (K+K')/2;
    U(freedofs) = K(freedofs,freedofs)\F(freedofs);
%      cndK=log(condest(K(freedofs,freedofs)))/log(10);
```

OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS

```
%compliance of DZ only
   dv = ones(nely,nelx);
%total compliance
c=F'*U;
ce=reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
```

FILTERING/MODIFICATION OF SENSITIVITIES

```
if ft == 1
    dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
elseif ft == 2
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
end
```

Gather info for MMA

```
f0val=c;
fval=(mean(xPhys(:))-volfrac)/volfrac;%;1*(-0.38-FAN_Ax_disp)/0.38
df0dx=dc(:);
dfdx=dv(:)'/length(dv(:))/volfrac;%;-1*dfandisp(:)'/0.38
innerit=0;
outvector1 = [outeriter innerit f0val fval'];
outvector2 = xval;
```

MMA code optimization

```
[x(:),ymma,zmma,lam,xsi,eta,mu,zet,S,low,upp] = ...
    mmasub(m,n,outeriter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,fval,dfdx,low,upp,a0,a,C,d);
xold2 = xold1;
xold1 = xval;
xval  = x(:);
change=norm(xval-xold1);
%     xval(x1>=2500&x4<=4500&z1==min(z1))=0;
```

PRINT RESULTS

```
    fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f kktnorm.:%7.3f Area.:%7.3f\n',outit,c, ...
        mean(xPhys(:)),kktnorm,A);
    figure(2)
    hold on
    plot(outit,c,'bo','MarkerFaceColor','b')
    plot(outit,mean(xPhys(:))*100,'ro','MarkerFaceColor','r')
%     plot(outit,G_ks*100,'mo','MarkerFaceColor','m')

%     plot(outit,cndK,'go','MarkerFaceColor','g')
    % plot(outeriter,(1+GKSl)*VMl,'ko','MarkerFaceColor','k')
    title(['Convergence V = ',num2str(mean(xPhys(:))*100),',compliance =',num2str(c),', iter = ', num2str(A)])
    grid on
    legend('compliance','V %')
    xlabel('iter')
```

%% The residual vector of the KKT conditions is calculated:

```
    [residu,kktnorm,residumax] = ...
        kktcheck(m,n,x(:),ymma,zmma,lam,xsi,eta,mu,zet,S, ...
        xmin,xmax,df0dx,fval,dfdx,a0,a,C,d);
    outvector1 = [outeriter innerit f0val fval(:)'];
    outvector2 = xval;
```

PLOT DENSITIES

```
figure(1)
colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off; drawnow;

    h = figure(1); set(h,'Color',[1 1 1]);
    [Yy,Xx]=find(nodenrs);
    Yy=nely+1-Yy;
    Xx=Xx-1;
    hold on; patchplot2 = patch('Vertices',[Xx,Yy],'Faces',edofMat(:,[2,4,6,8])/2,'FaceVertexCData',(1-xPhys(:))*[1 1 1],'FaceColor','flat','EdgeColor','
    axis equal; axis off; drawnow;hold off
    title(['Design zone at iteration ',num2str(outit)])
    colormap(gray);
    drawnow
    % Capture the plot as an image
      frame = getframe(h);
      im = frame2im(frame);
      [imind,cm] = rgb2ind(im,256);
    % Write to the GIF File
      if outit == 1
          imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
      else
          imwrite(imind,cm,filename,'gif','WriteMode','append');
      end

end
%
```

8.2 Uniform Area Condensation Code

```
function [Kcc,Kce,Kee,Fe,Fc,K_cond,F_cond, compliance_truss, u_cond,B]=
truss_stiffness_condensation(nelx,nely,E,A,I,L)
Kt=E*A/L;% axial stiffness of all the beam
Kf=E*I/L^3;% flexural behavior
Keh=[Kt 0 0 -Kt 0 0
    0 12*Kf 6*L*Kf 0 -12*Kf 6*L*Kf
    0 6*L*Kf 4*L^2*Kf 0 -6*L*Kf 2*L^2*Kf
    -Kt 0 0 Kt 0 0
    0 -12*Kf -6*L*Kf 0 12*Kf -6*L*Kf
    0 6*L*Kf 2*L^2*Kf 0 -6*L*Kf 4*L^2*Kf]; % Element stiffness matrix for Horizontal beam
Pov=[0 1 0 0 0 0
     1 0 0 0 0 0
     0 0 1 0 0 0
     0 0 0 0 1 0
     0 0 0 1 0 0
     0 0 0 0 0 1];% Rotation Matrix
Kev=Pov'*Keh*Pov; % stiffness matrix vertical element
node_list = 1:(2*(nelx+1)+3*(nely-1));%
node_dofs_map=reshape((1:3*(2*(nelx+1)+3*(nely-1)))',3,[])';
ELEMENT(1:nelx,:)=[(1:nelx)',(1:nelx)'+1];%Element connectivity matrix
ELEMENT(nelx + (1:nelx),:) = [nelx+1+(1:nelx)',(1:nelx)'+nelx+2];
ELEMENT(2*nelx+(1:(nely)),:)=[1,2*nelx+2+1
                              ((2*nelx+2+1):(2*nelx+2+nely-2))',((2*nelx+2+1):(2*nelx+2+nely-2))'+1
                                                    (2*nelx+2+nely-1), nelx+2];
ELEMENT(2*nelx+nely+(1:(nely)),:)=[fix(nelx/2),2*nelx+2+nely
                                                    ((2*nelx+2+nely):(2*nelx+2+2*nely-3))',((2*nelx+2+nely):(2*nelx+2+2*nely-3))'+1
                                                    (2*nelx+2+2*nely-2), nelx+1+fix(nelx/2)];
ELEMENT(2*nelx+2*nely+(1:(nely)),:)=[nelx+1,2*nelx+2+2*nely-2+1
                                                    ((2*nelx+2+2*nely-2+1):(2*nelx+2+3*(nely-1)-1))',((2*nelx+2+2*nely-2+1):(2*nelx+2+3*(nely-1)-1))'+1
                                                    (2*nelx+2+3*(nely-1)), 2*nelx+2];
ELEMENT_DOF=[node_dofs_map(ELEMENT(:,1),:),node_dofs_map(ELEMENT(:,2),:)]; % Local to global DOFs
```

```
%Local indexes and Non-zero terms in the stiffness matrix of horizontal and vertical elements
[io,jo,ko]=find(Keh);
[iv,jv,kv]=find(Kev);

%Global indexes and Non-zero terms in the stiffness matrix of hor. elements
Io=reshape(ELEMENT_DOF(1:2*nelx,io)',[],1);
Jo=reshape(ELEMENT_DOF(1:2*nelx,jo)',[],1);
Ko=[repmat(ko,nelx,1);repmat(ko,nelx,1)];

%Global indexes and Non-zero terms in the stiffness matrix of ver. elements
Iv=reshape(ELEMENT_DOF(2*nelx+(1:3*nely),iv)',[],1);
Jv=reshape(ELEMENT_DOF(2*nelx+(1:3*nely),jv)',[],1);
Kv=repmat(kv,3*nely,1);

% Global Stiffness Matrix
K = sparse([Io;Iv],[Jo;Jv],[Ko;Kv], max(node_dofs_map(:)), max(node_dofs_map(:))); %stiffness matrix assembly
K=(K+K')/2; % make it exactly symmetric

F = sparse(3*((nelx+1))+1,1,-1,max(node_dofs_map(:)),1);% Load Vector Assembly
coupling_dofs=[(1:3:(3*(nelx+1)-2));2:3:(3*(nelx+1)-1)]; %u and v DOFs of the 1st hor. beam
coupling_dofs = coupling_dofs(:);

reduced_dofs = setdiff(node_dofs_map(:),coupling_dofs); %Slave Nodes

ID_Load = find(reduced_dofs == (3*((nelx+1))+1)); %Force Node

%coupling_dofs = [coupling_dofs; reduced_dofs(ID_Load)];

%coupling DOFs = master nodes; Eliminated DOFs = slave nodes
u0=zeros(size(K,1),1);
u0(reduced_dofs) = K(reduced_dofs,reduced_dofs)\F(reduced_dofs);
Kcc = K(coupling_dofs,coupling_dofs);
Kce = K(coupling_dofs,reduced_dofs);
Kee = K(reduced_dofs,reduced_dofs);
Fc = F(coupling_dofs);
Fe = F(reduced_dofs);
B = (Kce/Kee);
%For the Condensation
K_cond = Kcc - B*Kce';
F_cond = Fc - B*Fe;
u_cond = K_cond\F_cond;
B_t = B';
compliance_truss = 1*(u0(ID_Load) + B_t(ID_Load,:)*u_cond)/10e12;
```

```
% Plotting
Xo=[(0:nelx)';(0:nelx)'];
Yo=[zeros(nelx+1,1);repmat(-nely,nelx+1,1)];
Xv=[zeros(nely-1,1);repmat(fix(nelx/2)-1,nely-1,1);repmat(nelx,nely-1,1)];
Yv=repmat((-1:-1:-nely+1)',3,1);
X=[Xo;Xv];
Y=[Yo;Yv];
plot(X(ELEMENT)',Y(ELEMENT)','k-o','MarkerFaceColor','k')
axis equal
U=reshape(u0,3,[])';
U=U(:,1:2);
DX=U(:,1);
DY=U(:,2);
hold on
```

8.3 Differential Area Code

Contents

```
function x=top88_plustruss_mma_cond_internship_by_set(nelx,nely,L,volfrac,tarmass_A1,tarmass_A2,tarmass_A3,tarmass_A4,tarmass_A5,penal,rmin,ft)

% nelx=50;
% nely=40;
% L=1;
% volfrac=0.3;
% tarmass_A1=3;
% tarmass_A2=3;
% tarmass_A3=1.5;
% tarmass_A4=1.5;
% tarmass_A5=1.5;
% penal=3;
% rmin=3.0;
% ft=2;

filename = 'Topology_Optimization_V0.3.gif';
% NEED TO DO:
% 1. OPTIMIZATION BY GROUPING AREAS OF BEAMS
% 2. GUASS-JORDAN ELIMINATION (NEW CONDENSATION TECHNIQUE)
% 3. COMPARISON OF TECHNIQUES
close all
```

MATERIAL PROPERTIES

```
E0 = 1;
Emin = 1e-9;
nu = 0.3;
P=4;
Sl=1;
rho = 1;
```

PREPARE FINITE ELEMENT ANALccYSIS

```
A11 = [12  3 -6 -3;  3 12  3  0; -6  3 12 -3; -3  0 -3 12];
A12 = [-6 -3  0  3; -3 -6 -3 -6;  0 -3 -6  3;  3 -6  3 -6];
B11 = [-4  3 -2  9;  3 -4 -9  4; -2 -9 -4 -3;  9  4 -3 -4];
B12 = [ 2 -3  4 -9; -3  2  9 -2;  4  9  2  3; -9 -2  3  2];
% D=E0/(1-nu^2)*[1 nu 0;nu 1 0; 0 0 (1-nu)/2];
% B=1/2*[-1 0 1 0 1 0 -1 0;0 -1 0 -1 0 1 0 1;-1 -1 -1 1 1 1 1 -1];
% DB=D*B;
% B=1/2*[-1 0 1 0 1 0 -1 0;0 -1 0 -1 0 1 0 1;-1 -1 -1 1 1 1 1 -1];
% DB=D*B;
% Cvm=[1 -0.5 0;-0.5 1 0;0 0 3];
% Sel=DB'*Cvm*DB;
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
%fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
```

```
fixeddofs = [2*(fix(nelx/2)*(nely+1)+1:(nely+1):((nelx)*(nely+1)+1))-1;
2*(fix(nelx/2)*(nely+1)+1:(nely+1):((nelx)*(nely+1)+1))];
fixeddofs = fixeddofs(:);
```

PREPARE FILTER

```
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
    for j1 = 1:nely
        e1 = (i1-1)*nely+j1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
                e2 = (i2-1)*nely+j2;
                k = k+1;
                iH(k) = e1;
                jH(k) = e2;
                sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
            end
        end
    end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
```

INITIALIZE ITERATION

x = repmat(1,nely,nelx); xPhys = x; loop = 0; change = 1;

INITIALIZE ITERATION

```
x = ones(nely,nelx);
A_max = 2;
xPhys = x;
loop = 0;
change = 1;
m = 6;
n = length(xPhys(:))+5;
epsimin = 0.0000001;
eeen    = ones(n-5,1);
eeem    = ones(m,1);
```

```
zeron   = zeros(n,1);
zerom   = zeros(m,1);
xval    = [xPhys(:); 6.5; 6.5; 3.5; 3.5; 3.5];
xold1   = xval;
xold2   = xval;
xmin    = zeron;
xmin(end-4:end) = 0.3*ones(5,1);
xmax    = cat(1,eeen, A_max*ones(5,1));
low     = xmin;
upp     = xmax;
C       = 1000*eeem;
d       = 0*eeem;
a0      = 1;
a       = zerom;
outeriter = 0;
maxoutit  = 1500;
kkttol  =0.001;

%%%% The iterations start:
kktnorm = kkttol+10;
% kktnorm = kkttol;
outit = 0;
change=1;
```

START ITERATION

```
while kktnorm > kkttol && outit < maxoutit && change > 0.01

%    if outit == 0
%        A_old = A_max;
%    else
%        A_old = A;
%    end
   I1 = 100*(sqrt(xval(end-4))^4)/12;
   I2 = 100*(sqrt(xval(end-3))^4)/12;
   I3 = 100*(sqrt(xval(end-2))^4)/12;
   I4 = 100*(sqrt(xval(end-1))^4)/12;
   I5 = 100*(sqrt(xval(end))^4)/12;

   %Condensation of Structure
   %if A_old - A > 0
        [Kcc,Kce, Kce_pr,Kee, Kee_pr, Fe, Fe_pr,Fc,K_cnd,F_cnd,compliance_truss, u_c, coupling_dofs_truss]
        = truss_stiffness_condensation_by_set(nelx,nely,E0,xval(end-4),...
            xval(end-3),xval(end-2),xval(end-1),xval(end), I1, I2, I3, I4, I5, L);
```

Generate the Projection matrix, load vector and stiffness matrix ready for assembly

```
        coupling_nodes = [nely+1 (nely+1)*fix((nelx+1)/2) (nely+1)*(nelx+1)];
        coupling_dofs = [coupling_nodes*2-1; coupling_nodes*2];
        coupling_dofs = coupling_dofs(:);

        coupling_nodes_pr = [nely+1:nely+1:(nely+1)*(nelx+1)];
        coupling_dofs_pr = [coupling_nodes_pr*2-1;coupling_nodes_pr*2];
        coupling_dofs_pr = coupling_dofs_pr(:);

        coupling_dofs_truss = sort(coupling_dofs_truss(:));

        coupling_dofs_sp1 = repmat(coupling_dofs_truss(:),6);
        coupling_dofs_sp1 = coupling_dofs_sp1(:,1);

        coupling_dofs_sp2 = sort(coupling_dofs_sp1);

        K_cnd_vec = K_cnd(:);
        K_cnd = sparse(coupling_dofs_sp1',coupling_dofs_sp2',K_cnd_vec,2*(nelx+1),2*(nelx+1));

        Pr = sparse(coupling_dofs,[1 2 11 12 21 22],ones(1,6),2*(nelx+1)*(nely+1),2*(nelx+1));
        Pr_pr = sparse(coupling_dofs_pr,1:2*(nelx+1),ones(1,2*(nelx+1)),2*(nelx+1)*(nely+1),2*(nelx+1));

        Kt = [Pr*K_cnd*Pr' Pr_pr*zeros(size(Kce_pr)); zeros(size(Kce_pr))'*Pr_pr' zeros(size(Kee_pr))];

        % Truss Displacement Recovery
%         U_c = (K_cnd^-1)*F_cnd;
%         R_g = -(Kee^-1)*Kce';
%         U_e = R_g*U_c;
%         U_t = [U_c; U_e];

        F_cnd_s = sparse([1 2 11 12 21 22]',ones(1,6)',F_cnd,2*(nelx+1),1);

        Kt = (Kt+Kt')/2;
        Ft = [Pr*F_cnd_s; zeros(size(Fe_pr))];
        U = zeros(2*(nely+1)*(nelx+1)+length(Fe_pr),1);
        F = U; %
        F = F + Ft;
        alldofs = [1:length(F)];
        freedofs = setdiff(alldofs,fixeddofs); %For Design Zone
  % end
   % New Condensed Stiffness Matrix (For new Area)
```

```
    outit   = outit+1;
    outeriter = outeriter+1;
    if ft == 1
    xPhys(:)=xval(end-5);
    elseif ft == 2
    xPhys(:) = (H*xval(1:end-5))./Hs;
    end
```

FE-ANALYSIS

```
    sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
    K0 = sparse(iK,jK,sK,size(Kt,1),size(Kt,2));
    K = K0 + Kt; % Question: For this step, we need to keep Kt the same size as K0,
                 % so I have replaced the Kt slaves with zeros. Will this
                 % cancel the speed-up gain we get from condensation.
    K = (K+K')/2;
    U(freedofs) = K(freedofs,freedofs)\F(freedofs); % We get a near singular matrix because of the zeros in the matrix
%      cndK=log(condest(K(freedofs,freedofs)))/log(10);
```

OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS

```
    %Compliance of DZ only
    dv = ones(nely,nelx);
    %Total Compliance (DZ + Truss)
    c = abs(F'*U);% + compliance_truss); %Recovery of Truss Compliance
    ce=reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
    dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
```

FILTERING/MODIFICATION OF SENSITIVITIES

```
    if ft == 1
        dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
    elseif ft == 2
        dc(:) = H*(dc(:)./Hs);
        dv(:) = H*(dv(:)./Hs);
    end
```

Gather info for MMA

```
    f0val = c; %Full Truss + DZ Compliance
```

```
 % Constraint Function Definition
 vol_frac = (mean(xPhys(:))-volfrac)/volfrac; %Volume Fraction
% mass_constraints = (xval(end-4)*L*nelx*rho +  xval(end-3)*nelx*L*rho +...
% xval(end-2)*nely*L*rho + xval(end-1)*nely*L*rho + xval(end)*nely*L*rho + sum(rho*xval(1:nelx*nely))- tarmass)/tarmass; %Check Mass Constraint

 mass_constraints_A1 = (xval(end-4)*L*nelx*rho - tarmass_A1)/tarmass_A1;
 mass_constraints_A2 = (xval(end-4)*L*nelx*rho - tarmass_A2)/tarmass_A2;
 mass_constraints_A3 = (xval(end-4)*L*nelx*rho - tarmass_A3)/tarmass_A3;
 mass_constraints_A4 = (xval(end-4)*L*nelx*rho - tarmass_A4)/tarmass_A4;
 mass_constraints_A5 = (xval(end-4)*L*nelx*rho - tarmass_A5)/tarmass_A5;


 mass =xval(end-4)*L*nelx*rho +  xval(end-3)*nelx*L*rho +...
 xval(end-2)*nely*L*rho + xval(end-1)*nely*L*rho + xval(end)*nely*L*rho + sum(rho*xval(1:nelx*nely));
 mass_plyon = sum(rho*xval(1:nelx*nely));
 mass_engine = xval(end-4)*L*nelx*rho +  xval(end-3)*nelx*L*rho +...
 xval(end-2)*nely*L*rho + xval(end-1)*nely*L*rho + xval(end)*nely*L*rho;
 M_Ratio_Pylon = mass_plyon/mass;
 M_Ratio_Engine = mass_engine/mass;
 fval = [vol_frac; mass_constraints_A1; mass_constraints_A2; mass_constraints_A3; mass_constraints_A4; mass_constraints_A5];

 % Construction of Sensitivity of Area of the Truss for Optimization (Truss)
 [dK_cnd_dA, Kee_a, B_a] = area_cond_by_set(nelx,nely,E0,L,Kce,Kee);
 dK_cnd_vec_dA = dK_cnd_dA(:);

 dK_cnd_dA = sparse(coupling_dofs_sp1',coupling_dofs_sp2',dK_cnd_vec_dA,2*(nelx+1),2*(nelx+1));
 K_area = [Pr*dK_cnd_dA*Pr' Pr_pr*zeros(size(Kce_pr)); zeros(size(Kce_pr))'*Pr_pr' zeros(size(Kee_pr))];
 K_area = (K_area+K_area')/2;

 [Kee_a1, B_a1] = area_cond_by_set_dA1(nelx,nely,E0,L,Kce,Kee);
 [Kee_a2, B_a2] = area_cond_by_set_dA2(nelx,nely,E0,L,Kce,Kee);
 [Kee_a3, B_a3] = area_cond_by_set_dA3(nelx,nely,E0,L,Kce,Kee);
 [Kee_a4, B_a4] = area_cond_by_set_dA4(nelx,nely,E0,L,Kce,Kee);
 [Kee_a5, B_a5] = area_cond_by_set_dA5(nelx,nely,E0,L,Kce,Kee);
 [dtruss_dA1, dtruss_dA2, dtruss_dA3, dtruss_dA4, dtruss_dA5] = truss_compliance_sensitivity_by_area_by_set(Kee_a1,Kee_a2,Kee_a3,
 Kee_a4,Kee_a5, Fe, B_a1,B_a2,B_a3,B_a4,B_a5, u_c);

 dc_dA1 = U'*K_area*U + dtruss_dA1;
 dc_dA2 = U'*K_area*U + dtruss_dA2;
 dc_dA3 = U'*K_area*U + dtruss_dA3;
 dc_dA4 = U'*K_area*U + dtruss_dA4;
 dc_dA5 = U'*K_area*U + dtruss_dA5;
```

```
%Construction of Sensitivity of Objective Functions
df0dc_dx = dc(:);
df0dx = [df0dc_dx; dc_dA1; dc_dA2; dc_dA3; dc_dA4; dc_dA5];

dfdx_dv = dv(:)'/length(dv(:))/volfrac;%;-1*dfandisp(:)'/0.38

dfdx_A1 = L*nelx*rho;
dfdx_A2 = L*nelx*rho;
dfdx_A3 = nely*L*rho;
dfdx_A4 = nely*L*rho;
dfdx_A5 = nely*L*rho;

dfdx = [dfdx_dv 0 0 0 0 0; zeros(1,length(dfdx_dv)) dfdx_A1 0 0 0 0; zeros(1,length(dfdx_dv)) 0 dfdx_A2 0 0 0;...
    zeros(1,length(dfdx_dv)) 0 0 dfdx_A3 0 0;...
    zeros(1,length(dfdx_dv)) 0 0 0 dfdx_A4 0; zeros(1,length(dfdx_dv)) 0 0 0 0 dfdx_A5];

innerit=0;
outvector1 = [outeriter innerit f0val fval'];
outvector2 = xval;
```

MMA code optimization

```
[xmma,ymma,zmma,lam,xsi,eta,mu,zet,S,low,upp] = ...
    mmasub(m,n,outeriter,xval,xmin,xmax,xold1,xold2, ...
        f0val,df0dx,fval,dfdx,low,upp,a0,a,C,d);
xold2 = xold1;
xold1 = xval;
xval  = xmma;
change=norm(xval-xold1);
```

PRINT RESULTS

```
fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f kktnorm.:%7.3f Area.:%7.3f\n Mass: %7.3f\n',outit,c, ...
    mean(xPhys(:)),kktnorm, xval(end-4), mass);
figure(2)
hold on
plot(outit,c,'bo','MarkerFaceColor','b')
plot(outit,mean(xPhys(:))*100,'ro','MarkerFaceColor','r')
%plot(outit,cndK,'go','MarkerFaceColor','g')
title(['Convergence V = ',num2str(mean(xPhys(:))*100),',compliance =',num2str(c),', iter = ', num2str(outit)])
grid on
legend('compliance','V %')
```

```
xlabel('iter')
```

PLOT Areas

```
figure(3)
hold on
plot(outit,xval(end-4),'ro','MarkerFaceColor','r')
plot(outit,xval(end-3),'go','MarkerFaceColor','g')
plot(outit,xval(end-2),'yo','MarkerFaceColor','y')
plot(outit,xval(end-1),'co','MarkerFaceColor','c')
plot(outit, xval(end),'mo','MarkerFaceColor','m')
title(['iter = ', num2str(outit), ' A1 =' , num2str(xval(end-4)), ' A2 =' , num2str(xval(end-2))])
grid on
legend('A1','A2')
xlabel('iter')
```

MASS Graph

```
figure(4)
hold on
plot(outit,M_Ratio_Engine,'ro','MarkerFaceColor','r')
plot(outit,M_Ratio_Pylon,'bo','MarkerFaceColor','b')
%title(['iter = ', num2str(outit), ' A1 =' , num2str(xval(end-4)), ' A2 =' , num2str(xval(end-2))])
grid on
legend('Mass of Engine','Mass of Pylon')
xlabel('iter')
```

%% The residual vector of the KKT conditions is calculated:

```
[residu,kktnorm,residumax] = ...
    kktcheck(m,n,xval,ymma,zmma,lam,xsi,eta,mu,zet,S, ...
    xmin,xmax,df0dx,fval,dfdx,a0,a,C,d);
outvector1 = [outeriter innerit f0val fval(:)'];
outvector2 = xval;
```

PLOT DENSITIES

```
h = figure(1);
set(h,'Color',[1 1 1]);
[Yy,Xx]=find(nodenrs);
Yy=nely+1-Yy;
Xx=Xx-1;
```

```
    hold on; patchplot2 = patch('Vertices',[Xx,Yy],'Faces',edofMat(:,[2,4,6,8])/2,'FaceVertexCData',(1-xval(1:end-5))*[1 1 1],'FaceColor',
    'flat','EdgeColor','none'); axis equal; axis off; drawnow;hold off
    title(['Design zone at iteration ',num2str(outit)])
    colormap(gray);
%     drawnow
%     % Capture the plot as an image
%       frame = getframe(h);
%       im = frame2im(frame);
%       [imind,cm] = rgb2ind(im,256);
%     % Write to the GIF File
%      if outit == 1
%           imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
%      else
%           imwrite(imind,cm,filename,'gif','WriteMode','append');
%      end

end
```