A MATLAB® IMPLEMENTATION OF HYBRID-MIXED STRESS FORMULATION ON TOPOLOGY OPTIMIZATION

Luis Armando Piccino Navarro

Wesley Góis

luis.piccino@gmail.com

wesley.gois@ufabc.edu.br

Mechanical Engineering Graduate Program at Federal University of ABC

Alameda da Universidade, 09606-070, Sao Bernardo, Sao Paulo, Brazil

Abstract. *The present work aims to describe an implementation of the Hybrid-Mixed Stress Formulation on Topology Optimization written in MATLAB® and to introduce the programming for this non-conventional formulation. The code includes a detailed algorithm to implement a Hybrid-Mixed Stress Formulation analysis, sensitivity analysis, a sensitive and density filter, optimality criterion optimizer and the display of the optimal structure, with all the variables written in terms of the stress field, which is the main advantage of the implemented formulation against a classical Finite Element analysis. The proposed formulation is able to approximate directly and independently both displacements and stress fields, providing a more accurate approximation for the last one which is usually the variable taken for most engineering applications. To illustrate the capabilities, efficiency and use of the code the rectangular cantilever middle loaded problem is analyzed and results shows that the formulation can achieve designs with structural members essentially solid when applying sensitivity filter, instead of the fading effect on the edges presented in classical formulations.*

Keywords: *MATLAB®, Hybrid-Mixed Stress Formulation, Topology Optimization*

1 INTRODUCTION

Over the past decades the Topology Optimization Method has become one of the most popular methods in industry and academia as a mechanical design tool for conceptual projects. Most of its popularity is due the constant development and publication of free and easy-to-use educational tools addressing the methodology, with Sigmund (2001) as precursor with its "99 lines Topology Optimization code written in MATLAB®", followed by many others, highlighting Andreassen et al. (2011) with an 88 lines code based on the first one focusing on achieving efficiency and Talischi et al. (2012), presenting *PolyTop* an efficient MATLAB® modular implementation using unstructured polygonal finite element meshes.

Although most of them implements a classical formulation of the Finite Element Method (FEM) to address the stress analysis necessary for the optimization, the search for alternative FE formulations is becoming more popular, either for deal with checkerboard problem or for a better and directly approximate of the stress field to address stress constraints. On this field, reference is made to Bruggi (2016) which addressed mixed-FEM to solve the checkerboard problem, apply stress constraints and apply the Topology Optimization to incompressible media and to Feitsma (2016) which analyzed the feasibility of alternative formulations of FEM to solve the checkerboard.

In this work, an efficient MATLAB® tool for structural Topology Optimization with the stress analyzes made by the Hybrid-Mixed Stress Formulation (HMSF) is presented, which can be viewed as an extension of the code developed in Andreassen et al. (2011). The HMSF approximates both displacement and stress field, giving the mixed scheme to the formulation, and the displacement on the boundary and on the domain, characterizing the hybrid scheme. The present work is divided in four main chapter: The first one bring a theorical background of the formulation involved and how the Topology Optimization problem is constructed; The aspects of the MATLAB® implementation comes in sequence whit sub-sections dedicated to the Linear System Solver and Numerical Issues related to the formulation and finally it's presented the numerical results regarding the tool and a conclusion is drawn with a discussion of some potential extensions.

2 THEORICAL CONSIDERATIONS

The section describes the theorical aspects related to the HMSF and the also how the Minimum Compliance problem is written in terms of the stress, the main variable of the non-conventional formulation.

2.1 Hybrid-Mixed Stress Formulation

The HMSF can be defined as a numerical method approximates two unrelated fields, stress s_Ω and displacement q_Ω on domain, which defines the mixed scheme, and the displacement field on the static boundary q_Γ. The independency of the domain and boundary field defines the hybrid scheme. The main advantage over a classical Finite Element Method is the direct approximation of the stress field, making the post-processing to define the stress field unnecessary. On the other hand, as it approximates three independent fields, the formulation carry an elevated number of degrees os freedom, requiring specific algorithms to handle the large sparce coefficient matrix involved on the scheme.

To derive the weak form of the formulation, consider a linear-elastic body in domain Ω with boundary Γ composed by a kinematic, Γ_u, and a static Γ_t boundary. Thus the Galerkin weighting relations based on the equilibrium, compatibility and Neumann boundary condition equations is given by:

$$\int_{\Omega} \delta u^T (L\sigma + b) d\Omega = 0 \tag{1}$$

$$\int_{\Gamma_t} \delta u_{\Gamma}^T (N\sigma - \bar{t}) d\Gamma = 0 \tag{2}$$

$$\int_{\Omega} \delta \sigma^T (\varepsilon - L^T u) d\Omega = 0 \tag{3}$$

In the above expressions, σ holds the stress, ε the strain and u the displacement fields defined in the domain Ω. The matrix L represent the differential divergence operator; b the body forces vector; N the matrix holding the components of the unit normal vector to the boundary and $\bar{t}$ the vector of surface forces on Γ_t.

By recalling the constitutive relation expressed in terms of the flexibility matrix f for isotropic material (considering plane elasticity) and applying the Green theorem, Equation 3 is written as:

$$\int_{\Omega} \delta\sigma^T f \sigma d\Omega + \int_{\Omega} (L\delta\sigma)^T u d\Omega - \int_{\Gamma_t} (N\delta\sigma)^T u_{\Gamma} d\Gamma + \int_{\Gamma_u} \overline{u}^T (N\sigma) d\Gamma = 0 \tag{4}$$

Where $\overline{u}$ is the vector of prescribed displacement of Γ_u. The approximation of the displacement and stress fields are done by using shape functions defined out of the Partition of Unity. In this work, bi linear polynomial shape functions is applied to approximate the fields. In a isoparametric coordinate system (ξ, η), the shape functions are written as:

$$\begin{aligned} \phi_1 &= \frac{1}{4}(\xi - 1)(\eta - 1) \quad \phi_2 = -\frac{1}{4}(\xi + 1)(\eta - 1) \\ \phi_3 &= \frac{1}{4}(\xi + 1)(\eta + 1) \quad \phi_4 = -\frac{1}{4}(\xi - 1)(\eta + 1) \end{aligned} \tag{5}$$

Thus the approximated fields can be written in terms of the nodal variables and the shape functions as:

$$\sigma = \phi_i s_{\Omega_i}, \quad u = \phi_i q_{\Omega_i}, \quad u_{\Gamma} = \phi_i s_{\Gamma_i} \tag{6}$$

Finally, taking in consideration the approximated fields in Eq. 6 and Eqs. 1, 2 and 4, the following linear system of equation of the Hybrid-Mixed Stress Formulation can be generate:

$$\begin{bmatrix} \int_{\Omega} \phi_j^T f \phi_i d\Omega & \int_{\Omega} \phi_j'^T \phi_i d\Omega & -\int_{\Gamma_t} (N\phi_j)^T \phi_i d\Gamma \\ \int_{\Omega} \phi_i'^T \phi_j d\Omega & 0 & 0 \\ -\int_{\Gamma_t} (N\phi_i)^T \phi_j d\Gamma & 0 & 0 \end{bmatrix} \begin{Bmatrix} s_{\Omega} \\ q_{\Omega} \\ q_{\Gamma} \end{Bmatrix} = \begin{Bmatrix} 0 \\ -\int_{\Omega} \phi_j^T b d\Omega \\ -\int_{\Gamma_t} \phi_j^T \bar{t} d\Gamma \end{Bmatrix} \tag{7}$$

Each integral on the above system define a sub matrix. The one related to the stress field is defined on the code as *FEL*, the one related to the displacement on the domain as *AEL*, the displacement on the boundary as *AGEL* and the load vector as *F*.

For a more in-depth analysis on the HMSF the reader is referred to the outstanding paper of De Freitas, De Almeida, and Pereira (1996) and to Góis and Proença (2012) work.

2.2 Problem Formulation

Introducing a density function $\rho(x)$ following the SIMP (Solid Isotropic Material with Penalization) methodology (Bendsøe, 1989), an inverse penalization is done on flexibility matrix, where f_0 is the flexibility matrix of virgin material.

$$f(\rho(x)) = \rho(x)^{-p} f_0 \tag{8}$$

In order to write a minimum compliance problem with Hybrid-Mixed Stress elements is necessary to write the compliance in terms of the primal variable in the formulation, stress. To do so, it is recalled that for linear elastic structures the complementary strain energy and the strain energy (*a.k.a.*, compliance) are the same; Thus, the minimum compliance problem can be written as follows:

$$\begin{aligned} & min\ C = \sum_{e=1}^{N} \rho_e^{-p} \sigma_e^T f_e^0 \sigma_e, \\ & s.t. \begin{cases} KU = F \\ V = \sum_{i=1}^{N} \rho_i \mu_i \leq volfrac \\ \rho_{min} \leq \rho_e \leq 1 \end{cases} \end{aligned} \tag{9}$$

Where the system *KU=F* represents the matricial system in Eq. 7. Lastly, the sensitivities of the objective function is given by:

$$\frac{\partial c}{\partial x_e} = -p \rho_e^{-1-p} \sigma_e^T f_e^0 \sigma_e \tag{10}$$

For further details on the Optimality Criteria and Filtering, the reader is refered to Andreassen et al. (2011) and Sigmund (2001).

3 MATLAB IMPLEMENTATION

In this section the MATLAB® code (see Appendix) is explained. The code is built up based on the codes written by Andreassen et al. (2011) and Sigmund (2001). The main program is called from the MATLAB® prompt by:

```
TOHMSF(nelx,nely,volfrac,penal,rmin,ft)
```

Where *nelx* and *nely* are the number of elements in the horizontal and vertical directions, *volfrac* the volume fraction desired, *penal* the penalization factor, *rmin* the filter sizer and lastly *ft*, a variable implemented in Andreassen et al. (2011) to define the filter used in the analysis:

sensitivity filter (*ft = 1*) or density filter (*ft = 2*). Variables such as boundary conditions and applied load are directly modified inside the code. The default problem corresponds to the rectangular cantilever middle loaded.

The main differences between the 88 line topology optimization MATLAB® code and the present one are the elastic linear analysis, which is now made via a Hybrid-Mixed Stress Analysis and the construction of the objective function and its derivative, which is now completly written in terms of the stress field, directly approximated via the chosen formulation. For more details on the implementation of the filter and optimality criteria based optimizer, the reader is referred to Andreassen et al. (2011).

Some details on the implementation of the non-conventional formulation, the linear system solver and numerical issues are discussed in the following subsections.

3.1 Hybrid-Mixed Stress Analysis

The Hybrid-Mixed Stress Analysis is splitted into two sections on the code. The first is a function named *EL* (lines 44 - 59) in which the three sub-matrices that composes the coefficient matrix are defined and also the address of the non-zero elements and its value in each matrix is taken in order to assemble via the *sparse* in-built function. The computation of the element matrices were made defining a Young Modulus $E = 1000$ N/m^2 and a Poisson's ratio $\nu = 0.3$.

The design domain is assumed to be rectangular with unitary square elements. Although the code is based on Andreassen et al. (2011) and Sigmund (2001) the numbering of nodes and elements are construct in a different way; both are numbered line-wise from left to right, starting at the lower left corner and the DOFs *3n - 2*, *3n - 1* and *3n* correspond to the plane stress state of node *n*, the DOFs *2n -1 + 3nodes* and *2n + 3nodes* corresponding to the horizontal and vertical domain displacement, where *nodes* holds the total number os nodes, and finally *2n -1 + 5nodes* and *2n + 5nodes* corresponding the horizontal and vertical boundary displacements.

The second section is defined on *HMSF*, where the global matrix is assembled (lines 61 - 115). To process it, a loop over all elements is made defining the variables *n1* and *n2*, the lower and upper left element global node number used to define the position of the element sub matrices on the global. In sequence, loops three loops are performed over the non-zeros elements of the sub matrices to establish the triplet *I*, *J* and *S* that are later used to construct the global coefficient matrix using the *sparse* function:

```
93 K = sparse(I,J,S,ndof,ndof);
```

For this formulation, because the number of degrees of freedom is 3.5 times higher than a classical Finite Element Method, its essential the construction of the coefficient matrix via *sparse* function as the memory required to store full matrix become very large as the element number increase, as it can be seen in Table 1.

The applied load is also constructed via a triplet *FI*, *FJ* and *FS*, where the first variable holds the DOF of the applied load and the last one the load value. The boundary conditions are defined via the variable *nfixed* holding the nodes where the conditions are applied and the variable *doffixed*, where DOFs with prescribed displacement are stored. It should be noted that the first portion of the variable, *2*nodes* defines the displacement in *y* direction and the second, *2*nodes - 1* in *x* direction.

Table 1: Memory Required to Allocate Coefficient Matrix

Mesh Size	Sparse Storage	Full Storage
60 x 40	3.8 Mb	2.5 Gb
100 x 100	15.7 Mb	> 38.0 Gb

3.2 Linear System Solver

The resulting linear system is linear dependent with its coefficient matrix symmetric, ill-conditioned and non positive-definite, and by these characteristics a numerical procedure is applied to solve it. In this work, the Babuška Method is applied. The method consists basically in construct a diagonal matrix B to ponderate the coefficient matrix and perturb its diagonal by a small number, in order to achieve a positive-definite matrix (lines 99 -107). In sequence, the system can be solved by a direct or iterative method.

Due to the sparse characteristics of the matrix and the efficiency of the routine compared to the MATLAB® in-built function *backslash* the Mutifrontal Method (lines 112 - 113) is used to solve the ponderated system (HSL, 2017). The subroutines *ma57_factor* and *ma57_solve* are part of the Harwell Subroutine Library, a collection of Fortran codes for large scientific computation, and are available for academinc purpouses under request at `http://www.hsl.rl.ac.uk/`.

An alternative to the Multifrontal Method is to solve the system by an iterative method, as described in Góis and Proença (2012). The method is following presented in terms of a MATLAB® function, in which the variable K referes to the coefficient matrix, F the independent vector and x the unknowns vector, initially set as zero. Further comparison on the efficiency of the methods will be addressed on next section.

```
function [x] = BABUSKA(K,F,x)
        j = 0;
        r = F; tol = 10^-3; err = 1.0;
        den = transpose(r)*r;
        while err > tol
            a = K\r;
            x = x + a;
            r = r - K*a; num = transpose(r)*r;
            j = j + 1;
            err = sqrt(num/den);
        end
end
```

3.3 Numerical Issues

As mentioned earlier the coefficient matrix of the formulation is ill-conditioned and throughout the optimization process this condition may worsen, affecting the solvability of the system when applying a iterative method, as the one described above. For this reason, in iterative methods its necessary to reduce the tolerance of convergence to a level high enough to not cause a divergence of the problem and low enough to ensure a solution. By testing different tolerances,

it has been found that $tol = 10^{-3}$ can either ensure solution and convergence. For linear analysis without optimization a lower tolerance can be used.

The formulation also encounters an issue related to shear locking. The application of a shear type force leads to a locked solution, similar to the occurrence of the locking phenomena when applying classical Finite Element with Timoshenko beam elements to analyze thin beams (Euler-Bernoulli beams). For this reason, a reduced integration technique was applied to two element sub matrices related to domain variables, *FEL* and *AEL*.

4 NUMERICAL RESULTS

To check the capabilities and performance of the code, the rectangular cantilever middle loaded optimization problem is performed. Three meshes consisting of *60 x 40*, *120 x 80* and *210 x 140* elements were considered and solved by meanings of the two linear solver, the in-built *backslash* function and the Mutifrontal Method subroutine. The volume constraint is set to 0.4 and the penalization parameter to 3.0. To check the difference in the filter application, both sensitivity and density filter was considered, with a r_{min} set to 0.03 times the width of the design domain.

Figure 1 shows the optimal design achieved. It can be checked that both filters lead to the same optimal design, checkerboard free and and mesh independent designs. The main difference lies on the shading effect of the edges presented on the ones with density filter. For the designs with sensitivity filter an almost black-and-white design is achieved.

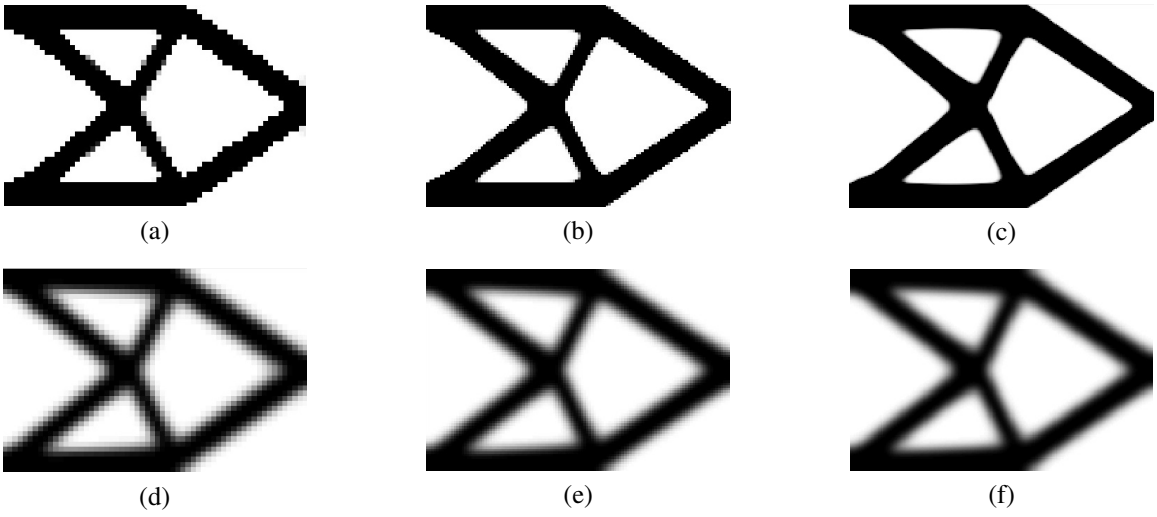

Figure 1: Optimal design for the rectangular cantilever middle loaded using sensitivity filter (top) and density filter (bottom). The meshes used are (a)/(d) 60 x 40, (b)/(e) 120 x 80, and (c)/(f) 210 x 140 elements.

Concerning the efficiency of the chosen linear solver, Table 2 shows the computational time in seconds per iteration for optimization of the rectangular cantilever middle-loaded using sensitivity filter. The computation was determined as the average of the first ten iterations of the optimization loop, the same procedure done by Andreassen et al. (2011).

One can check that the Multifrontal Method is far more efficient than the in-builted MATLAB® function; For a regular mesh size, say 120 x80 a factor of almost 100 speed improvement is accomplished. As the mesh size become larger the computational time using *backslah* becomes

Table 2: CPU time in seconds per iteration for different linear solvers for the optimization of the rectangular cantilever middle loaded.

Mesh Size	60 x 40	120 x 80	210 x 140
Multifrontal Method	1.77	8.65	59.63
Backslash	15.07	757.13	-

excessively large; Moreover, the in-builted function consumes more memory when solving the system, turning impractable the optimization of domains with large mesh sizes.

5 CONCLUSION

This work presents a MATLAB code for topology optimization applying a Hybrid-Mixed Stress element. The code was based mainly on the 88 line code presented by Sigmund (2001) and includes optimizer, mesh dependency filters on density and sensitivities and a HMSF analysis code. Due to its similarity with the referenced code, it can be extended to include multi load problems, passive areas and other boundary conditions. Although the code was created to be efficient by taking advantage of the sparsity of the coefficient matrix, its necessary the application of external linear solvers to acheive best results in terms of efficiency, as in-builted MATLAB® functions can make the process slower.

The code is provided in Appendix and can be requested by e-mailing the first author to encourage research in the field of application of non-conventional formulations of the Finite Element Method on Topology Optimization problems. Ongoing research aims to take advantage of the directly approximated stress field to apply stress constraints as an extension of the code.

ACKNOWLEDGEMENTS

This work was supported by the Coordination for the Improvement of Higher Education Personnel, CAPES, trough graduate scholarship for the first author. The authors also thank the financial support of CNPq (National Council for Research and Development) under grant number 471291/2013-7.

REFERENCES

Andreassen, Erik et al., 2011. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, vol. 43, n. 1, pp. 1–16.

Bendsøe, M P, 1989. Optimal shape design as a material distribution problem. *Structural optimization*, vol. 1, n. 4, pp. 193–202.

Bruggi, Matteo, 2016. Topology optimization with mixed finite elements on regular grids. *Computer Methods in Applied Mechanics and Engineering*, vol. 305, pp. 133–153.

De Freitas, J. A Teixeira, J. P Moitinho De Almeida, and E. M B Ribeiro Pereira, 1996. Non-conventional formulations for the finite element method. *Structural Engineering and Mechanics*, vol. 4, n. 6, pp. 655–678.

Feitsma, J. M., 2016. *Feasibility of Alternative Finite Element Formulations within Topology Optimization*. Master's Thesis. Delft University of Technology.

Góis, W. and S. P. B. Proença, 2012. Generalized Finite Element Method on Nonconventional Hybrid-Mixed Formulation. *International Journal of Computational Methods*, vol. 09, n. 03, p. 1250038.

HSL, 2017. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.

Sigmund, O., 2001. A 99 line topology optimization code written in Matlab. *Structural and Multidisciplinary Optimization*, vol. 21, n. 2, pp. 120–127.

Talischi, Cameron et al., 2012. PolyTop: a Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, vol. 45, n. 3, pp. 329–357.

APPENDIX

```
%% TOPOLOGY OPTIMIZATION WITH HYBRID–MIXED STRESS ELEMENT %%
%  BY LUIS ARMANDO, AUGUST 2017 %
function TOHMSF (nelx,nely,volfrac,penal,rmin,ft)
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
change = 1.; c= 0.; loop = 0; dv = ones(nely,nelx);
[H,Hs]    = check(nelx,nely,rmin);
while change > 0.01
xold = x; loop = loop + 1;
% HMSF–ANALYSIS
[STRESS] = HMSF(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [~,~,~,~,~,~,~,~,~,FEL] = EL;
  c = 0.; dc = zeros(nely,nelx);
  for ely = 1:nely
    for elx = 1:nelx
        n1 = elx + (nelx+1)*(ely-1);
        n2 = elx + (nelx+1)*ely;
        S = STRESS([n1*3-2; n1*3-1; n1*3; n1*3+1; n1*3+2; n1*3+3;
            n2*3+1; n2*3+2; n2*3+3; n2*3-2; n2*3-1; n2*3],1);
      c = c + x(ely,elx)^(-penal)*S'*FEL*S;
      dc(ely,elx) = penal*x(ely,elx)^(-penal-1)*S'*FEL*S;
    end
  end
% FILTERING
if ft == 1 % ON SENSITIVITIES
    dc(:) = H*(x(:).*dc(:))./Hs;
elseif ft == 2 % ON DENSITY
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
end
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
```

```
[x] = OC(nelx,nely,x,volfrac,dc,dv,ft,H,Hs);
change = max(max(abs(x-xold)));
% PRINT RESULTS
  disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
       ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
       ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-flip(x)); axis equal; axis tight; axis off;pause(1e-6);
end
end
% ELEMENT MATRICES
function [iF,jF,sF,iAe,jAe,sAe,iAg,jAg,sAg,FEL] = EL
F = 1e-8*[6250 -1875 0; -1875 6250 0; 0 0 16250];
FEL = [F F F F; F F F F; F F F F; F F F F]; [iF,jF,sF] = find(FEL);
AEL = [-1/8 0 -1/8 0 -1/8 0 -1/8 0; 0 -1/8 0 -1/8 0 -1/8 0 -1/8;
    -1/8 -1/8 -1/8 -1/8 -1/8 -1/8 -1/8 -1/8; 1/8 0 1/8 0 1/8 0 1/8 0;
    0 -1/8 0 -1/8 0 -1/8 0 -1/8; -1/8 1/8 -1/8 1/8 -1/8 1/8 -1/8 1/8;
    -1/8 0 -1/8 0 -1/8 0 -1/8 0; 0 1/8 0 1/8 0 1/8 0 1/8;
    1/8 -1/8 1/8 -1/8 1/8 -1/8 1/8 -1/8; 1/8 0 1/8 0 1/8 0 1/8 0;
    0 1/8 0 1/8 0 1/8 0 1/8; 1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]; [iAe,jAe,sAe] = find(AEL);
AGEL = [-1/3 0 0 0 -1/6 0 0 0; 0 -1/3 0 -1/6 0 0 0 0;
    -1/3 -1/3 -1/6 0 0 -1/6 0 0; 0 0 1/3 0 0 0 1/6 0;
    0 -1/6 0 -1/3 0 0 0 0; -1/6 0 -1/3 1/3 0 0 0 1/6;
    -1/6 0 0 0 -1/3 0 0 0; 0 0 0 0 0 1/3 0 1/6;
    0 -1/6 0 0 1/3 -1/3 1/6 0; 0 0 1/6 0 0 0 1/3 0;
    0 0 0 0 0 1/6 0 1/3; 0 0 0 1/6 1/6 0 1/3 1/3]; [iAg,jAg,sAg] = find(AGEL);
end
% HMSF ANALYSIS
function [STRESS] = HMSF(nelx,nely,x,penal)
[iF,jF,sF,iAe,jAe,sAe,iAg,jAg,sAg,~] = EL;
ntrip = 0; ndof = (nelx+1)*(nely+1)*7; nodes = (nelx+1)*(nely+1);
I = zeros((nelx)*(nely)*(size(iF,1)+size(iAg,1)+size(iAg,1)),1);
J = zeros((nelx)*(nely)*(size(iF,1)+size(iAg,1)+size(iAg,1)),1);
S = zeros((nelx)*(nely)*(size(iF,1)+size(iAg,1)+size(iAg,1)),1);
for ely = 1:nely
    for elx = 1:nelx
        n1 = elx + (nelx+1)*(ely-1);
        n2 = elx + (nelx+1)*ely;
        fdof = [n1*3-2 n1*3-1 n1*3 n1*3+1 n1*3+2 n1*3+3 ...
            n2*3-2 n2*3-1 n2*3 n2*3+1 n2*3+2 n2*3+3];
        dof = [n1*2-1 n1*2 n1*2+1 n1*2+2 n2*2-1 n2*2 n2*2+1 n2*2+2];
        aedof = dof + nodes*3; agdof = dof + nodes*5;
        for fi = 1:size(iF,1)
            ntrip = ntrip + 1;
```

```
77              I(ntrip) = fdof(iF(fi)); J(ntrip) = fdof(jF(fi)); S(ntrip
    ) = x(ely,elx)^(-penal)*sF(fi);
78          end
79          for ei = 1:size(iAe,1)
80              ntrip = ntrip + 1;
81              I(ntrip) = fdof(iAe(ei)); J(ntrip) = aedof(jAe(ei)); S(
    ntrip) = sAe(ei);
82              ntrip = ntrip + 1;
83              I(ntrip) = aedof(jAe(ei)); J(ntrip) = fdof(iAe(ei)); S(
    ntrip) = sAe(ei);
84          end
85          for gi = 1:size(iAg,1)
86              ntrip = ntrip + 1;
87              I(ntrip) = fdof(iAg(gi)); J(ntrip) = agdof(jAg(gi)); S(
    ntrip) = -sAg(gi);
88              ntrip = ntrip + 1;
89              I(ntrip) = agdof(jAg(gi)); J(ntrip) = fdof(iAg(gi)); S(
    ntrip) = -sAg(gi);
90          end
91      end
92  end
93  K = sparse(I,J,S,ndof,ndof);
94  % APPLIED LOAD
95  FI =  5*nodes+(nelx+1)*(nely/2+1)*2; FJ = 1; FS = -10; F = sparse(FI,
    FJ,-FS,ndof,1);
96  % BOUNDARY CONDITION
97  nfixed = 1:nelx+1:(nelx+1)*nely+1;
98  doffixed = sort([2*nfixed 2*nfixed-1]) + 5*nodes;
99  % MATRIX MODIFICATION FOR SOLVING SYSTEM
100 iB = zeros(3*nodes,1); jB = zeros(3*nodes,1); sB = zeros(3*nodes,1);
101 for bi = 1:3*nodes
102     iB(bi) = bi; jB(bi) = bi; sB(bi) = K(bi,bi)^ (-1/2);
103 end
104 B = sparse(iB,jB,sB,3*nodes,3*nodes);
105 K(:,1:3*nodes) = K(:,1:3*nodes)*B;
106 K(1:3*nodes,:) = B*K(1:3*nodes,:);
107 K = K + 10^-8*speye(7*nodes);
108
109 K(doffixed,:) = 0; K(doffixed,doffixed) = eye(max(size(doffixed)));
110 F(doffixed) = 0;
111 % MULTIFRONTAL METHOD FOR SOLVING LINEAR SYSTEM
112 struct = ma57_factor(K);
113 U = ma57_solve(K, full(F), struct);
114 STRESS = B*U(1:3*nodes);
115 end
116 % H/Hs MATRIX DEFINITION FOR FILTER APPLICATION
117 function [H,Hs]=check(nelx,nely,rmin)
118 iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
119 jH = ones(size(iH));
```

```
sH = zeros(size(iH));
k = 0;
for i1 = 1:nely
    for j1 = 1:nelx
        e1 = (j1-1)*nely+i1;
        for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nely)
            for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),
    nelx)
                e2 = (j2-1)*nely+i2;
                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);
end
% OPTIMALITY CRITERIA METHOD
function [xnew]=OC(nelx,nely,x,volfrac,dc,dv,ft,H,Hs)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(dc./dv/lmid))
   )));
  if ft == 2
      xnew(:) =  (H*xnew(:))./Hs;
  end
  if sum(sum(xnew)) - volfrac*nelx*nely > 0;
    l1 = lmid;
  else
    l2 = lmid;
  end
end
end
```