

```

function mmtopopt(problem,nely,ft,rmin,matmod,m,massfrac,costfrac,x0,D,E,P)
% Multi-material topology optimization_Modified ordered SIMP, 2021
%
% Arguments:
% Problem: Bridge - problem = 1, Cantilever beam - problem = 2
% Number of elements in the vertical direction: nely
% Filter scheme: Sensitivities filter ft = 1, Density filter ft = 2, Threshold projection ft = 3
% Filter radius: rmin
% Material model: Zuo & Saitou - matmod = 1, Silveira & Palma - matmod = 2
% Constraints: Mass - m = 1, Cost - m = 2, Mass and cost - m = 3
% Mass fraction constraint: massfrac
% Cost fraction constraint: costfrac
% Design variables initial value: x0
% Normalized densities vector: e.g. D = [0, 0.4, 0.6, 1.0]
% Normalizes elastic modulus vector: e.g. E = [1e-9, 0.5, 0.7, 1.0]
% Normalized cost vector: e.g. P = [0, 1.6, 1.2, 1.0]
%
% Some important papers to understand the code:
% Silveira, O.A.A, & Palma, L.F., Some considerations on multi-material topology optimization using
ordered SIMP
% Zuo, W., Saitou, K. Multi-material topology optimization using ordered SIMP interpolation. Struct
Multidisc Optim 55, 477-491 (2017). https://doi.org/10.1007/s00158-016-1513-3
% Andreassen, E., Clausen, A., Schevenels, M. et al. Efficient topology optimization in MATLAB using
88 lines of code. Struct Multidisc Optim 43, 1-16 (2011). https://doi.org/10.1007/s00158-010-0594-7
% Wang, F., Lazarov, B.S. & Sigmund, O. On projection methods, convergence and robust formulations
in topology optimization. Struct Multidisc Optim 43, 767-784 (2011). https://doi.org/10.1007/s00158-010-0602-y
% Filters:
% Sigmund, O. Morphology-based black and white filters for topology optimization. Struct Multidisc
Optim 33, 401-424 (2007). https://doi.org/10.1007/s00158-006-0087-x

close all; clc;
%% PROBLEM BOUNDARY CONDITIONS
switch problem
case 1
    % Bridge (using symmetry, 1x1 square and even domain)
    sym = 1;
    nelx = nely;
    F = sparse(2*(nely+1)*(nelx+1),1);
    F(2*(nely+1),1) = -1;
    F(2*(nely+1)*(2*nelx/4+1),1) = -1;
    fixeddofs = union(1:2:2*(nely+1),2*(nelx+1)*(nely+1));
case 2
    % Cantilever beam (2x1 rectangle domain)
    sym = 0;
    nelx = 2*nely;
    F = sparse(2*(nely+1)*(nelx+1),1,-1,2*(nely+1)*(nelx+1),1);
    fixeddofs = 1:2*(nely+1);
otherwise
    error('This problem does not exist.')
end
% Displacement vector
U = zeros(2*(nely+1)*(nelx+1),1);
% Free Degrees of Freedom
alldofs = 1:2*(nely+1)*(nelx+1);
freedofs = setdiff(alldofs,fixeddofs);

%% PREPARE FINITE ELEMENT ANALYSIS
nu = 0.3;
A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);

%% PREPARE FILTER
[H,Hs] = prepare_filter(nelx,nely,rmin);

% INITIALIZE ITERATION
x(1:nely,1:nelx) = x0;
%Filter:
if ft == 1 % Sensitivities filter ft = 1,
    xPhys = x;
elseif ft == 2 % Density filter ft = 2
    xPhys = zeros(nely,nelx);

```

```

    xPhys(:) = (H*x(:))./Hs;
elseif ft == 3 % Threshold projection ft = 3
    eta = 0.5;
    beta = 0.1; %Initial beta
    xTilde(:) = (H*x(:))./Hs;
    nmat = length(E); % Number of phases (solids + void)
    for i=1:nmat-1
        indices = xTilde > D(i) & xTilde <= D(i+1);
        scale_y = D(i+1) - D(i);
        scale_x = 1/(D(i+1)-D(i));
        shift_x = D(i);
        shift_y = D(i);
        we = scale_x*(xTilde(indices)-shift_x);
        xPhys(indices) = scale_y*((tanh(beta*eta)+tanh(beta*(we-eta)))/(tanh(beta*eta)+tanh(beta*(1-
eta))))+shift_y;
    end
    xPhys = reshape(xPhys,nely,nelx);
end

% INITIALIZE MMA OPTIMIZER
if m == 1 || m == 2; mm = 1; elseif m == 3; mm = 2; end % The number of general constraints.
n = nelx*nely; % The number of design variables x_j.
xmin = zeros(n,1); % Column vector with the lower bounds for the variables x_j.
xmax = ones(n,1); % Column vector with the upper bounds for the variables x_j.
xold1 = x(:); % xval, one iteration ago (provided that iter>1).
xold2 = x(:); % xval, two iterations ago (provided that iter>2).
low = ones(n,1); % Column vector with the lower asymptotes from the previous iteration
(provided that iter>1).
upp = ones(n,1); % Column vector with the upper asymptotes from the previous iteration
(provided that iter>1).
a0 = 1; % The constants a_0 in the term a_0*z.
a = zeros(mm,1); % Column vector with the constants a_i in the terms a_i*z.
c_MMA = 10000*ones(mm,1); % Column vector with the constants c_i in the terms c_i*y_i., See
Svanberg ...
d = zeros(mm,1); % Column vector with the constants d_i in the terms 0.5*d_i*(y_i)^2.

%% START ITERATION
loop = 0;
loopinmax = 25;
pi = 1;
pf = 5;
ps = 0.25;
if ft == 3
    eta = 0.5;
    beta = 0.1; %Initial beta
    betaf = 25; %Final beta
    rounds = (pf-pi)/ps + 1; %External rounds
    q = (betaf/beta)^(1/(rounds*loopinmax-1)); %Geometric sequence common ratio
end

% External loop
for penal = pi:ps:pf % Continuation method

    if ft == 3; disp([' Continuation: beta = ' sprintf('%.2f',beta)]); end
    disp([' Continuation: penal = ' sprintf('%.2f',penal)]);

    %Internal loop
    for loopin=1:loopinmax

        % Update iterations
        loop = loop+1;

        %% MATERIAL MODEL
        [E_,dE_,P_,dP_] = MaterialModel(matmod,nelx,nely,xPhys,penal,D,E,P);

        %% FE-ANALYSIS
        sK = reshape(KE(:)*E_(:)',64*nelx*nely,1);
        K = sparse(iK,jK,sK);
        K = (K+K')/2;
        U(freedofs) = K(freedofs,freedofs)\F(freedofs);

        %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
        % Compliance
        ce = reshape(sum((U(edofMat)*KE).*U(edofMat)),2),nely,nelx);
        c = sum(sum(E_.*ce));
        dc = -dE_.*ce;
        ve = ones(nely,nelx); % Elemental volume (Element 1x1)
        % Mass
        mass = sum(sum(xPhys))/(massfrac*n) - 1; %simplified
        dmass = ve / (massfrac*n);
    end
end

```

```

% Cost
cost = sum(ve(:).*xPhys(:).*P_(:))/(costfrac*n) - 1; %simplified
dcost = (ve.*P_ + ve.*xPhys.*dP_)/(costfrac*n);

%% FILTERING/MODIFICATION OF SENSITIVITIES
if ft == 1
    gamma = 1e-3; %See Andreassen et al. 2011
    dc(:) = H*(x(:).*dc(:))./Hs./max(gamma,x(:));
elseif ft == 2
    dc(:) = H*(dc(:))./Hs;
    dmass(:) = H*(dmass(:))./Hs;
    dcost(:) = H*(dcost(:))./Hs;
elseif ft == 3
    for i=1:nmat-1
        indices = xTilde >= D(i) & xTilde < D(i+1);
        scale_y = D(i+1) - D(i);
        scale_x = 1/(D(i+1)-D(i));
        shift_x = D(i);
        we = scale_x*(xTilde(indices)-shift_x);
        dx(indices) = scale_y*(1-tanh(beta*(we-eta)).^2)*beta*scale_x/(tanh(beta*eta)
+tanh(beta*(1-eta)));
    end
    dc(:) = H*(dc(:).*dx(:))./Hs;
    dmass(:) = H*(dmass(:).*dx(:))./Hs;
    dcost(:) = H*(dcost(:).*dx(:))./Hs;
end

%% DESIGN UPDATE BY METHOD OF MOVING ASYMPTOTES
xval = x(:);
f0val = c;
df0dx = dc(:);
%Number of constraints
if m == 1 %Mass constraint
    fval = mass;
    dfdx = dmass(:)';
elseif m == 2 %Cost constraints
    fval = cost;
    dfdx = dcost(:)';
elseif m == 3 %Mass and cost constraints
    fval = [mass; cost];
    dfdx = [dmass(:)'; dcost(:)'];
end
[xmma, ~, ~, ~, ~, ~, ~, ~, ~, low, upp] = ...
mmasub(mm, n, loop, xval, xmin, xmax, xold1, xold2, ...
f0val, df0dx, fval, dfdx, low, upp, a0, a, c_MMA, d);

% Update MMA Variables
xnew = reshape(xmma, nely, nelx);
xold2 = xold1(:);
xold1 = x(:);
x = xnew;

% Update Physical densities
if ft == 1
    xPhys = xnew;
elseif ft == 2
    xPhys(:) = (H*xnew(:))./Hs;
elseif ft == 3
    beta = q*beta; % Update beta
    xTilde(:) = (H*xnew(:))./Hs;
    for i=1:nmat-1
        indices = xTilde > D(i) & xTilde <= D(i+1);
        scale_y = D(i+1) - D(i);
        scale_x = 1/(D(i+1)-D(i));
        shift_x = D(i);
        shift_y = D(i);
        xPhys(indices) = scale_y*((tanh(beta*eta)+tanh(beta*(scale_x*(xTilde(indices)-
shift_x)-eta)))/(tanh(beta*eta)+tanh(beta*(1-eta))))+shift_y;
    end
    xPhys = max(reshape(xPhys, nely, nelx), 1e-5);
end
if max(max(xPhys))==1; error('Maybe we have a problem!'); end

%% POST PROCESSOR
% MEASURE OF NON-DISCRETENESS
[nondisc, nelem_int] = mnd(nelx, nely, xPhys, D);

% PRINT RESULTS
disp([' It.: ' sprintf('%i', loop) ' Obj.: ' sprintf('%.2f', c) ]);

```

```

disp([' Mass Fraction: ' sprintf('%.3f',sum(sum(xPhys))/n)];
[~,~,P_~] = MaterialModel(matmod,nelx,nely,xPhys,penal,D,E,P);
disp([' Cost Fraction: ' sprintf('%.3f',sum(xPhys(:).*P_(:))/(n))]);

% RENDER MULTI-MATERIAL TOPOLOGY
if mod(loop,5)==0
    Render(nelx,nely,xPhys,D,sym,problem);
end
end

% MEASURE OF NON-DISCRETENESS
global nondisc = sum(nelem_int .* nondisc)/sum(nelem_int);
disp(['Global NonDisc: ' sprintf('%.2f%%',global_nondisc)...
    ' Number of elems: ' sprintf('%i %i %i',nelem_int) ...
    ' NonDisc: ' sprintf('%.2f%% %.2f%% %.2f%%',nondisc)]);

end

%% %%%%% PREPARE FILTER %%%%%
function [H,Hs] = prepare_filter(nelx,nely,rmin)
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);
end

%% %%%%% RENDER MULTI-MATERIAL TOPOLOGY %%%%%
function Render(nelx,nely,x,D,sym,problem)
if problem == 1 % Void, MAT1, MAT2, and MAT3 - Section 5.1
    switch length(D)
        case 3
            if D(2) == 0.4/0.6
                color = [1 1 1; 0 1 0; 0 0 1]; % Green and blue
            elseif D(2) == 0.4
                color = [1 1 1; 0 1 0; 1 0 0]; % Green and red
            elseif D(2) == 0.6
                color = [1 1 1; 0 0 1; 1 0 0]; %Blue and red
            end
        case 4
            color = [1 1 1; 0 1 0; 0 0 1; 1 0 0]; % Green, blue and red
    end
elseif problem == 2 % Steel and aluminum - Section 5.2
    color = [1 1 1; 0.5 0.5 0.5; 0 0 0];
end
% Render
x = reshape(x,nely*nelx,1);
for i=1:length(D)-1
    indices = x(:,1) > D(i) & x(:,1) <= D(i+1);
    x(indices,2) = i; %Interval
    we = (x(indices,1)-D(i))/(D(i+1)-D(i));
    x(indices,3) = 1-we; %Left side
    x(indices,4) = we; %Right side
end
I = x(:,3).*color(x(:,2),:) + x(:,4).*color(x(:,2)+1,:);
I = imresize(reshape(I,nely,nelx,3),1,'bilinear');
I = flip(I,1);
%Symmetry
hold on
if sym == 1 % Using symmetry
    image([flip(I,2) I]); axis image off; drawnow;
elseif sym == 0 % No symmetry
    image(I); axis image off; drawnow;
end
end

```

```

axis equal; axis tight; axis off;
end

%% %%% MEASURE OF NON-DISCRETENESS %%%
function [nondisc,nelem_int] = mnd(nelx,nely,x,D)
x = reshape(x,nely*nelx,1);
for i=1:length(D)-1
    indices = x(:,1) > D(i) & x(:,1) <= D(i+1);
    x(indices,2) = i; %Interval
    we = (x(indices,1)-D(i))/(D(i+1)-D(i));
    x(indices,3) = 1-we; %Left side
    x(indices,4) = we; %Right side
    %Number of elements and non-discreteness measure per interval
    nelem_int(i) = sum(indices);
    nondisc(i) = (sum(4*(x(indices,4)).*(1 - x(indices,4))))/(nelem_int(i))*100;
end
if sum(nelem_int) ~= nely*nelx; error('We have a problem!'); end
end

%% %%% MATERIAL MODEL %%%
function [E_,dE_,P_,dP_] = MaterialModel(matmod,nelx,nely,x,penal,D,E,P)
x = reshape(x,nely*nelx,1);
if matmod == 1 % Zuo & Saitou interpolation
    for i=1:length(D)-1
        indices = x(:,1) >= D(i) & x(:,1) <= D(i+1);
        % Elastic Modulus
        A = (E(i)-E(i+1))/(D(i)^(1*penal)-D(i+1)^(1*penal));
        B = E(i)-A*(D(i)^(1*penal));
        x(indices,2) = A*(x(indices,1).^(penal))+B;
        x(indices,3) = A*penal*(x(indices,1).^((penal)-1));
        % Cost
        penal = 1/penal;
        A = (P(i)-P(i+1))/(D(i)^(1*penal)-D(i+1)^(1*penal));
        B = P(i)-A*(D(i)^(1*penal));
        x(indices,4) = A*(x(indices,1).^(penal))+B;
        x(indices,5) = A*penal*(x(indices,1).^((penal)-1));
    end
elseif matmod == 2 % Silveira & Palma interpolation (modified interpolation)
    for i=1:length(D)-1
        indices = x(:,1) >= D(i) & ( x(:,1)<D(i+1) | x(:,1)==1 );
        % Elastic Modulus
        if E(i+1) >= E(i)
            a = E(i+1) - E(i);
            b = 1/(D(i+1) - D(i));
            c = D(i);
            d = E(i);
            x(indices,2) = a*(b*(x(indices,1)-c)).^(penal)+d;
            x(indices,3) = a*penal*(b*(x(indices,1)-c)).^(penal-1)*b;
        else
            a = E(i) - E(i+1);
            b = 1/(D(i+1) - D(i));
            c = D(i+1);
            d = E(i+1);
            x(indices,2) = a*(b*(c-x(indices,1))).^(penal)+d;
            x(indices,3) = a*penal*(b*(c-x(indices,1))).^(penal-1)*-b;
        end
        % Cost
        if P(i+1) >= P(i)
            a = P(i) - P(i+1);
            b = 1/(D(i+1) - D(i));
            c = D(i+1);
            d = P(i+1);
            x(indices,4) = a*(b*(c-x(indices,1))).^(penal)+d;
            x(indices,5) = a*penal*(b*(c-x(indices,1))).^(penal-1)*-b;
        else
            a = P(i+1) - P(i);
            b = 1/(D(i+1) - D(i));
            c = D(i);
            d = P(i);
            x(indices,4) = a*(b*(x(indices,1)-c)).^(penal)+d;
            x(indices,5) = a*penal*(b*(x(indices,1)-c)).^(penal-1)*b;
        end
    end
end
end
E_ = reshape(x(:,2),nely,nelx);
dE_ = reshape(x(:,3),nely,nelx);
P_ = reshape(x(:,4),nely,nelx);
dP_ = reshape(x(:,5),nely,nelx);
end

```