

IIT Hyderabad

Assignment 3

Submitted by:

ME21BTECH11001 Abhishek Ghosh

ME5053: Soft Robotics

Mechanical Engineering

27.04.2025

Submitted to:

Dr. Prabhat Kumar

1 Question 1

MATLAB Code for HoneyTop90 with Projection Filter

```
1 function honey_top90_projection_q1(HNex, HNey, volfrac, penal, rfill,
   ft)
2 %% ----- Material Properties -----
3 E0 = 1;
4 Emin = E0 * 1e-9;
5
6 %% ----- Mesh Generation and DOF Assignment -----
7 NstartVs = reshape(1:(1+2*HNex)*(1+HNey), 1+2*HNex, 1+HNey);
8 DOFstartVs = reshape(2*NstartVs(1:end-1,1:end-1)-1, 2*HNex*HNey, 1);
9 NodeDOFs = repmat(DOFstartVs, 1, 8) + repmat([2*(2*HNex+1)+[2 3 0 1] 0
   1 2 3], 2*HNex*HNey, 1);
10
11 % Remove DOFs for edges based on honeycomb pattern
12 skip = (2*HNex:2*HNex:2*HNex*HNey)' + mod(1:HNey,2)';
13 ActualDOFs = NodeDOFs(setdiff(1:2*HNex*HNey, skip), :);
14 HoneyDOFs = [ActualDOFs(2:2:end,1:2), ActualDOFs(1:2:end,:), ActualDOFs
   (2:2:end,7:8)];
15
16 % Nodal coordinates
17 Ncyi = repmat(reshape(repmat([-0.25 0.25]', HNey+1, 1), 2, HNey+1) +
   ...
18               reshape(1.5*sort(repmat((0:HNey)',2,1)), 2, HNey+1), HNex
   +1, 1);
19 Ncyi(:,1:2:end) = flip(Ncyi(:,1:2:end));
20 Ncyf = Ncyi(1:end-1,:);
21
22 HoneyNCO = [repmat((0:cos(pi/6):2*HNex*cos(pi/6)),1,HNey+1)', Ncyf(:)];
23
24 if mod(HNey,2) == 0
25     HoneyDOFs(end:-1:end-HNex+2,1:6) = HoneyDOFs(end:-1:end-HNex+2,1:6)
   - 2;
26     idx = (2*HNex+1)*(HNey+1);
27     HoneyNCO([idx-1, idx], :) = [];
28 end
29
30 [Nelem, Nnode] = deal(size(HoneyDOFs,1), size(HoneyNCO,1));
31
32 %% ----- Force and Boundary Conditions -----
33 L = max(HoneyNCO(:,1));
34 F = sparse(2*Nnode,1);
35
36 target_x = [L/4, L/2, 3*L/4];
37 Fvalue = 1;
38 force_nodes = [];
39
40 for i = 1:length(target_x)
41     [~, node_idx] = min(abs(HoneyNCO(:,1) - target_x(i)));
42     force_nodes = [force_nodes; node_idx];
43 end
44
45 F(2*force_nodes) = -Fvalue;
46
47 % Vertical supports at x=0 and x=L
```

```

48 fixeddofs = 2*find(abs(HoneyNCO(:,1)) < 1e-6 | abs(HoneyNCO(:,1) - L) <
    1e-6);
49 alldofs = 1:2*Nnode;
50 freedofs = setdiff(alldofs, fixeddofs);
51
52 %% ----- Assembly Preparation -----
53 iK = reshape(kron(HoneyDOFs, ones(12,1))', 144*Nelem, 1);
54 jK = reshape(kron(HoneyDOFs, ones(1,12))', 144*Nelem, 1);
55
56 KE = (E0/1000) * [
57     616.43012  92.77147  -168.07333  65.54377  -232.28511  -0.00032
58     -120.65312 -83.28564 -71.60020 -92.77115  -23.81836  17.74187;
59     92.77147  509.30685  101.02751  -71.90335  0.00032  -18.03857
60     -83.28564 -24.48314 -92.77179 -178.72347 -17.74187 -216.15832;
61     -168.07333 101.02751  455.74522  0.00000 -168.07333 -101.02751
62     -71.60020 -92.77179  23.60185  -0.00000 -71.60020  92.77179;
63     65.54377  -71.90335  0.00000  669.99176 -65.54377  -71.90335
64     -92.77115 -178.72347 -0.00000 -168.73811  92.77115 -178.72347;
65     -232.28511 0.00032  -168.07333 -65.54377  616.43012 -92.77147
66     -23.81836 -17.74187 -71.60020  92.77115 -120.65312  83.28564;
67     -0.00032 -18.03857 -101.02751 -71.90335 -92.77147  509.30685
68     17.74187 -216.15832  92.77179 -178.72347  83.28564 -24.48314
69     -120.65312 -83.28564 -71.60020 -92.77115 -23.81836  17.74187
70     616.43012  92.77147 -168.07333  65.54377 -232.28511 -0.00032;
71     -83.28564 -24.48314 -92.77179 -178.72347 -17.74187 -216.15832
72     92.77147  509.30685 101.02751 -71.90335  0.00032  -18.03857;
73     -71.60020 -92.77179  23.60185  -0.00000 -71.60020  92.77179
74     -168.07333 101.02751  455.74522  0.00000 -168.07333 -101.02751;
75     -92.77115 -178.72347 -0.00000 -168.73811  92.77115 -178.72347
76     65.54377 -71.90335  0.00000  669.99176 -65.54377 -71.90335;
77     -23.81836 -17.74187 -71.60020  92.77115 -120.65312  83.28564
78     -232.28511 0.00032 -168.07333 -65.54377  616.43012 -92.77147;
79     17.74187 -216.15832  92.77179 -178.72347  83.28564 -24.48314
80     -0.00032 -18.03857 -101.02751 -71.90335 -92.77147  509.30685
81 ];
82
83 %% ----- Filter Preparation -----
84 Cxx = repmat([sqrt(3)/2*(1:2:2*HNex-1), sqrt(3)*(1:HNex-1)], 1, ceil(
    HNey/2));
85 Cyy = repmat(3/4 + 3/2*(0:HNey-1), HNex, 1);
86 Cyy(HNex+1:2*HNex:end) = [];
87 ct = [Cxx(1:length(Cyy))', Cyy(:)];
88
89 DD = cell(Nelem,1);
90 for j = 1:Nelem
91     dist = sqrt((ct(j,1) - ct(:,1)).^2 + (ct(j,2) - ct(:,2)).^2);
92     idx = find(dist <= rfill);
93     DD{j} = [idx, j*ones(size(idx)), dist(idx)];
94 end
95 DD = vertcat(DD{:});
96 HHs = sparse(DD(:,2), DD(:,1), 1 - DD(:,3)/rfill);
97 HHs = spdiags(1./sum(HHs,2), 0, Nelem, Nelem) * HHs;
98
99 %% ----- Heaviside Projection Filter Parameters -----
100 % Adding Heaviside projection filter parameters
101 beta = 1; % Initial beta value
102 betamax = 128; % Maximum beta value
103 eta = 0.5; % Threshold parameter

```

```

92
93 %% ----- Initialization -----
94 x = volfrac * ones(Nelem,1);
95 [loop, change, maxiter, dv, move] = deal(0, 1, 200, ones(Nelem,1), 0.2)
96 ;
97 %% ----- Initialize variables based on filter type -----
98 if ft == 0 || ft == 1 % No filter or sensitivity filter
99     xPhys = x;
100 elseif ft == 2 % Density filter
101     xPhys = HHs * x;
102 elseif ft == 3 % Heaviside projection filter
103     xTilde = HHs * x; % First apply density filter
104     % Then apply Heaviside projection
105     xPhys = (tanh(beta*eta) + tanh(beta*(xTilde-eta))) ./ (tanh(beta*
        eta) + tanh(beta*(1-eta)));
106 end
107
108 %% ----- Optimization Loop -----
109 while (change > 0.01 && loop < maxiter)
110     loop = loop + 1;
111
112     % Finite Element Analysis
113     sK = reshape(KE(:) * (Emin + xPhys'.^penal * (E0-Emin)), 144*Nelem,
        1);
114     K = sparse(iK, jK, sK);
115
116     U = zeros(2*Nnode,1);
117     U(freedofs) = decomposition(K(freedofs,freedofs),'chol','lower') \
        F(freedofs);
118
119     % Objective and Sensitivities
120     ce = sum((U(HoneyDOFs) * KE) .* U(HoneyDOFs), 2);
121     c = sum((Emin + xPhys.^penal * (E0-Emin)) .* ce);
122     dc = -penal * (E0-Emin) * xPhys.^(penal-1) .* ce;
123
124     % Apply appropriate filtering to sensitivities
125     if ft == 0 % No filtering
126         % Do nothing
127     elseif ft == 1 % Sensitivity filtering
128         dc = HHs' * (x.*dc) ./ max(1e-3, x);
129     elseif ft == 2 % Density filtering
130         dc = HHs' * dc;
131         dv = HHs' * dv;
132     elseif ft == 3 % Heaviside projection filter
133         % Chain rule for sensitivities
134         dH = beta * (1 - tanh(beta*(xTilde-eta)).^2) ./ (tanh(beta*eta)
            + tanh(beta*(1-eta)));
135         dc = HHs' * (dc .* dH);
136         dv = HHs' * (dv .* dH);
137     end
138
139     % Optimality Criteria Update
140     xold = x;
141     [xUpp, xLow] = deal(xold + move, xold - move);
142     OcC = xold .* sqrt(-dc ./ dv);
143
144     % Set limits for the Lagrange multiplier

```

```

145 if ft == 3
146     l1 = 0; l2 = 1e9; % For projection filter, use wide range to
        avoid numerical issues
147 else
148     l1 = 0; l2 = max(0cC) / volfrac;
149 end
150
151 while (l2-l1)/(l2+l1) > 1e-3
152     lmid = 0.5*(l2+l1);
153     x = max(0, max(xLow, min(1, min(xUpp, 0cC / lmid))));
154     if mean(x) > volfrac
155         l1 = lmid;
156     else
157         l2 = lmid;
158     end
159 end
160
161 % Update physical variables based on filter type
162 if ft == 0 || ft == 1
163     xPhys = x;
164 elseif ft == 2
165     xPhys = HHs * x;
166 elseif ft == 3
167     xTilde = HHs * x;
168     xPhys = (tanh(beta*eta) + tanh(beta*(xTilde-eta))) ./ (tanh(
        beta*eta) + tanh(beta*(1-eta)));
169 end
170
171 change = max(abs(x - xold));
172
173 % Print Results
174 fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n', loop, c,
        mean(xPhys), change);
175
176 % Plot intermediate designs
177 colormap('gray');
178 scatter(ct(:,1), ct(:,2), [], 1-xPhys, 'filled');
179 axis equal off;
180 drawnow;
181
182 % Update beta parameter (continuation approach)
183 if ft == 3 && mod(loop, 60) == 0 && beta < betamax
184     beta = 2 * beta;
185     fprintf(' Beta updated to: %g\n', beta);
186 end
187 end
188 end

```



Figure 1:

MATLAB Code for top88 with Helmholtz Filter

```

1  %%%% TOPOLOGY OPTIMIZATION WITH HELMHOLTZ FILTER FOR BEAM PROBLEM %%%%
2  function top88_helmholtz_q1(nelx,nely,volfrac,penalMax,rmin)
3  % input example: >> top88_beam(80,20,0.5,3,2)
4  % The beam has length L, with a = L/4
5
6  % Ensuring nelx is divisible by 4 for exact placement of supports
7  if mod(nelx, 4) ~= 0
8      fprintf('Warning: nelx should be divisible by 4 for exact placement
9              of supports\n');
10     nelx = 4 * round(nelx/4);
11     fprintf('Adjusted nelx to %d\n', nelx);
12 end
13
14 %% MATERIAL PROPERTIES
15 E0 = 1;
16 Emin = 1e-9;
17 nu = 0.3;
18 penal = 0.96;
19
20 %% PREPARE FINITE ELEMENT ANALYSIS
21 A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
22 A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
23 B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
24 B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
25 KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
26 nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
27 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
28 edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*
29     nely,1);
30 iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
31 jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
32
33 %% DEFINE LOADS AND SUPPORTS FOR BEAM PROBLEM
34 % Initialize force vector
35 F = sparse(2*(nely+1)*(nelx+1),1);

```

```

34 % Apply point loads at a=L/4, 2a=L/2, and 3a=3L/4
35 loadPositions = [nelx/4, nelx/2, 3*nelx/4];
36 for i = 1:length(loadPositions)
37     F(2*nodenrs(1, loadPositions(i)+1)) = -1; % Negative y-direction
        with force = 1
38 end
39
40 U = zeros(2*(nely+1)*(nelx+1),1);
41
42 % Define supports:
43 % Left support at x=0 (pin)
44 leftSupportNode = nodenrs(nely+1, 1);
45 leftFixedDofs = [2*leftSupportNode-1, 2*leftSupportNode]; % Constrain
        both x and y
46
47 % Right support at x=L (roller)
48 rightSupportNode = nodenrs(nely+1, nelx+1);
49 rightFixedDofs = [2*rightSupportNode]; % Only vertical constraint for
        roller
50
51 % Combine fixed DOFs
52 fixeddofs = union(leftFixedDofs, rightFixedDofs);
53 alldofs = 1:2*(nely+1)*(nelx+1);
54 freedofs = setdiff(alldofs, fixeddofs);
55
56 %% PREPARE HELMHOLTZ FILTER
57 Rmin = rmin/2/sqrt(3); % Conversion between classical and PDE filter
        radius
58 % Define filter stiffness matrix
59 KEF = [
60     2/3 -1/6 -1/3 -1/6
61     -1/6 2/3 -1/6 -1/3
62     -1/3 -1/6 2/3 -1/6
63     -1/6 -1/3 -1/6 2/3];
64 KEF = Rmin^2*KEF + [
65     4/9 1/9 1/9 1/9
66     1/9 4/9 1/9 1/9
67     1/9 1/9 4/9 1/9
68     1/9 1/9 1/9 4/9]/4;
69 % Setup filter FE matrices
70 edofVecF = reshape(nodenrs(1:end-1,1:end-1),nelx*nely,1);
71 edofMatF = repmat(edofVecF,1,4) + repmat([0 nely+[1 2] 1],nelx*nely,1);
72 iKF = reshape(kron(edofMatF,ones(4,1))',16*nelx*nely,1);
73 jKF = reshape(kron(edofMatF,ones(1,4))',16*nelx*nely,1);
74 sKF = reshape(KEF(:)*ones(1,nelx*nely),16*nelx*nely,1);
75 KF = sparse(iKF,jKF,sKF);
76 LF = chol(KF,'lower'); % Cholesky factorization for efficient solving
77 iTF = reshape(edofMatF,4*nelx*nely,1);
78 jTF = reshape(repmat([1:nelx*nely],4,1)',4*nelx*nely,1);
79 sTF = repmat(1/4,4*nelx*nely,1);
80 TF = sparse(iTF,jTF,sTF);
81
82 %% INITIALIZE ITERATION
83 x = repmat(volfrac,nely,nelx);
84 xPhys = x;
85 loop = 0;
86 change = 1;
87 %% START ITERATION

```

```

88 while change > 0.01
89     loop = loop + 1;
90     penal = min(penalMax, penal + 0.04);
91     %% FE-ANALYSIS
92     sK = reshape(KE(:)*(Emin+xPhys(:)).^penal*(E0-Emin)),64*nelx*nely,1);
93     K = sparse(iK,jK,sK); K = (K+K')/2;
94     tic; U(freedofs) = K(freedofs,freedofs)\F(freedofs); toc;
95     %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
96     ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
97     c = sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
98     dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
99     dv = ones(nely,nelx);
100
101     %% HELMHOLTZ FILTERING OF SENSITIVITIES
102     dc(:) = TF'*(LF'\(LF\((TF*dc(:)))));
103     dv(:) = TF'*(LF'\(LF\((TF*dv(:)))));
104
105     %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL
106     DENSITIES
107     l1 = 0; l2 = 1e9; move = 0.2;
108     while (l2-l1)/(l1+l2) > 1e-3
109         lmid = 0.5*(l2+l1);
110         xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid)))));
111
112         %% HELMHOLTZ FILTERING OF DENSITIES
113         xPhys(:) = TF'*(LF'\(LF\((TF*xnew(:)))));
114
115         if sum(xPhys(:)) > volfrac*nelx*nely, l1 = lmid; else l2 = lmid;
116         end
117     end
118     change = max(abs(xnew(:)-x(:)));
119     x = xnew;
120     %% PRINT RESULTS
121     fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, ...
122         mean(xPhys(:)),change);
123     %% PLOT DENSITIES
124     colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off;
125     drawnow;
126 end

```

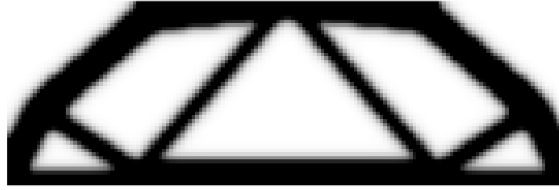



Figure 2:

2 Question 2

MATLAB Code for HoneyTop90 with Projection Filter

```

1 function honey_top90_projection_Q2(HNex, HNey, volfrac, penal, rfill,
   ft)
2 %% ----- Material Properties -----
3 E0 = 1;
4 Emin = E0 * 1e-9;
5
6 %% ----- Mesh Generation and DOF Assignment -----
7 NstartVs = reshape(1:(1+2*HNex)*(1+HNey), 1+2*HNex, 1+HNey);
8 DOFstartVs = reshape(2*NstartVs(1:end-1,1:end-1)-1, 2*HNex*HNey, 1);
9 NodeDOFs = repmat(DOFstartVs, 1, 8) + repmat([2*(2*HNex+1)+[2 3 0 1] 0
   1 2 3], 2*HNex*HNey, 1);
10
11 % Remove DOFs for edges based on honeycomb pattern
12 skip = (2*HNex:2*HNex:2*HNex*HNey)' + mod(1:HNey,2)';
13 ActualDOFs = NodeDOFs(setdiff(1:2*HNex*HNey, skip), :);
14 HoneyDOFs = [ActualDOFs(2:2:end,1:2), ActualDOFs(1:2:end,:), ActualDOFs
   (2:2:end,7:8)];
15
16 % Nodal coordinates
17 Ncyi = repmat(reshape(repmat([-0.25 0.25]', HNey+1, 1), 2, HNey+1) +
   ...
18               reshape(1.5*sort(repmat((0:HNey)',2,1)), 2, HNey+1), HNex
   +1, 1);
19 Ncyi(:,1:2:end) = flip(Ncyi(:,1:2:end));
20 Ncyf = Ncyi(1:end-1,:);
21
22 HoneyNCO = [repmat((0:cos(pi/6):2*HNex*cos(pi/6)),1,HNey+1)', Ncyf(:)];
23
24 if mod(HNey,2) == 0
25     HoneyDOFs(end:-1:end-HNex+2,1:6) = HoneyDOFs(end:-1:end-HNex+2,1:6)
   - 2;
26     idx = (2*HNex+1)*(HNey+1);

```

```

27     HoneyNCO([idx-1, idx], :) = [];
28 end
29
30 [Nelem, Nnode] = deal(size(HoneyDOFs,1), size(HoneyNCO,1));
31
32 %% ----- Force and Boundary Conditions -----
33 L = max(HoneyNCO(:,1));
34 F = sparse(2*Nnode,1);
35
36 % Apply force F = 2 at 45 angle at the top-right corner
37 F_magnitude = sqrt(2);
38 F_x = F_magnitude * cos(pi/4); % F*cos(45 )
39 F_y = F_magnitude * sin(pi/4); % F*sin(45 )
40
41 % Find top-right node (maximum x, minimum y)
42 maxX = max(HoneyNCO(:,1));
43 minY = min(HoneyNCO(:,2));
44 [~, topRightNodeIdx] = min(sum([(HoneyNCO(:,1) - maxX).^2, (HoneyNCO
    (:,2) - minY).^2], 2));
45
46 F(2*topRightNodeIdx-1) = F_x; % Horizontal component (positive x)
47 F(2*topRightNodeIdx) = -F_y; % Vertical component (negative y)
48
49 % Fix all DOFs at the left end (cantilever beam)
50 leftNodes = find(abs(HoneyNCO(:,1)) < 1e-6);
51 fixeddofs = [2*leftNodes-1; 2*leftNodes];
52 fixeddofs = reshape(fixeddofs, [], 1);
53
54 alldofs = 1:2*Nnode;
55 freeddofs = setdiff(alldofs, fixeddofs);
56
57 %% ----- Assembly Preparation -----
58 iK = reshape(kron(HoneyDOFs, ones(12,1))', 144*Nelem, 1);
59 jK = reshape(kron(HoneyDOFs, ones(1,12))', 144*Nelem, 1);
60
61 KE = (E0/1000) * [
62     616.43012  92.77147  -168.07333  65.54377  -232.28511  -0.00032
        -120.65312  -83.28564  -71.60020  -92.77115  -23.81836  17.74187;
63     92.77147  509.30685  101.02751  -71.90335  0.00032  -18.03857
        -83.28564  -24.48314  -92.77179  -178.72347  -17.74187  -216.15832;
64    -168.07333  101.02751  455.74522  0.00000  -168.07333  -101.02751
        -71.60020  -92.77179  23.60185  -0.00000  -71.60020  92.77179;
65     65.54377  -71.90335  0.00000  669.99176  -65.54377  -71.90335
        -92.77115  -178.72347  -0.00000  -168.73811  92.77115  -178.72347;
66    -232.28511  0.00032  -168.07333  -65.54377  616.43012  -92.77147
        -23.81836  -17.74187  -71.60020  92.77115  -120.65312  83.28564;
67    -0.00032  -18.03857  -101.02751  -71.90335  -92.77147  509.30685
        17.74187  -216.15832  92.77179  -178.72347  83.28564  -24.48314;
68    -120.65312  -83.28564  -71.60020  -92.77115  -23.81836  17.74187
        616.43012  92.77147  -168.07333  65.54377  -232.28511  -0.00032;
69    -83.28564  -24.48314  -92.77179  -178.72347  -17.74187  -216.15832
        92.77147  509.30685  101.02751  -71.90335  0.00032  -18.03857;
70    -71.60020  -92.77179  23.60185  -0.00000  -71.60020  92.77179
        -168.07333  101.02751  455.74522  0.00000  -168.07333  -101.02751;
71    -92.77115  -178.72347  -0.00000  -168.73811  92.77115  -178.72347
        65.54377  -71.90335  0.00000  669.99176  -65.54377  -71.90335;
72    -23.81836  -17.74187  -71.60020  92.77115  -120.65312  83.28564
        -232.28511  0.00032  -168.07333  -65.54377  616.43012  -92.77147;

```

```

73      17.74187 -216.15832  92.77179 -178.72347  83.28564 -24.48314
      -0.00032 -18.03857 -101.02751 -71.90335 -92.77147  509.30685
74 ];
75
76 %% ----- Filter Preparation -----
77 Cxx = repmat([sqrt(3)/2*(1:2:2*HNex-1), sqrt(3)*(1:HNex-1)], 1, ceil(
      HNey/2));
78 Cyy = repmat(3/4 + 3/2*(0:HNey-1), HNex, 1);
79 Cyy(HNex+1:2*HNex:end) = [];
80 ct = [Cxx(1:length(Cyy))', Cyy(:)];
81
82 DD = cell(Nelem,1);
83 for j = 1:Nelem
84     dist = sqrt((ct(j,1) - ct(:,1)).^2 + (ct(j,2) - ct(:,2)).^2);
85     idx = find(dist <= rfill);
86     DD{j} = [idx, j*ones(size(idx)), dist(idx)];
87 end
88 DD = vertcat(DD{:});
89 HHs = sparse(DD(:,2), DD(:,1), 1 - DD(:,3)/rfill);
90 HHs = spdiags(1./sum(HHs,2), 0, Nelem, Nelem) * HHs;
91
92 %% ----- Heaviside Projection Filter Parameters -----
93 % Adding Heaviside projection filter parameters
94 beta = 1; % Initial beta value
95 betamax = 128; % Maximum beta value
96 eta = 0.5; % Threshold parameter
97
98 %% ----- Initialization -----
99 x = volfrac * ones(Nelem,1);
100 [loop, change, maxiter, dv, move] = deal(0, 1, 200, ones(Nelem,1), 0.2)
    ;
101
102 %% ----- Initialize variables based on filter type -----
103 if ft == 0 || ft == 1 % No filter or sensitivity filter
104     xPhys = x;
105 elseif ft == 2 % Density filter
106     xPhys = HHs * x;
107 elseif ft == 3 % Heaviside projection filter
108     xTilde = HHs * x; % First apply density filter
109     % Then apply Heaviside projection
110     xPhys = (tanh(beta*eta) + tanh(beta*(xTilde-eta))) ./ (tanh(beta*
        eta) + tanh(beta*(1-eta)));
111 end
112
113 %% ----- Optimization Loop -----
114 while (change > 0.01 && loop < maxiter)
115     loop = loop + 1;
116
117     % Finite Element Analysis
118     sK = reshape(KE(:) * (Emin + xPhys'.^penal * (E0-Emin)), 144*Nelem,
        1);
119     K = sparse(iK, jK, sK);
120
121     U = zeros(2*Nnode,1);
122     U(freedofs) = decomposition(K(freedofs,freedofs),'chol','lower') \
        F(freedofs);
123
124     % Objective and Sensitivities

```

```

125 ce = sum((U(HoneyDOFs) * KE) .* U(HoneyDOFs), 2);
126 c = sum((Emin + xPhys.^penal * (E0-Emin)) .* ce);
127 dc = -penal * (E0-Emin) * xPhys.^(penal-1) .* ce;
128
129 % Apply appropriate filtering to sensitivities
130 if ft == 0 % No filtering
131     % Do nothing
132 elseif ft == 1 % Sensitivity filtering
133     dc = HHs' * (x.*dc) ./ max(1e-3, x);
134 elseif ft == 2 % Density filtering
135     dc = HHs' * dc;
136     dv = HHs' * dv;
137 elseif ft == 3 % Heaviside projection filter
138     % Chain rule for sensitivities
139     dH = beta * (1 - tanh(beta*(xTilde-eta)).^2) ./ (tanh(beta*eta)
140         + tanh(beta*(1-eta)));
141     dc = HHs' * (dc .* dH);
142     dv = HHs' * (dv .* dH);
143 end
144
145 % Optimality Criteria Update
146 xold = x;
147 [xUpp, xLow] = deal(xold + move, xold - move);
148 OcC = xold .* sqrt(-dc ./ dv);
149
150 % Set limits for the Lagrange multiplier
151 if ft == 3
152     l1 = 0; l2 = 1e9; % For projection filter, use wide range to
153         avoid numerical issues
154 else
155     l1 = 0; l2 = max(OcC) / volfrac;
156 end
157
158 while (l2-l1)/(l2+l1) > 1e-3
159     lmid = 0.5*(l2+l1);
160     x = max(0, max(xLow, min(1, min(xUpp, OcC / lmid))));
161     if mean(x) > volfrac
162         l1 = lmid;
163     else
164         l2 = lmid;
165     end
166 end
167
168 % Update physical variables based on filter type
169 if ft == 0 || ft == 1
170     xPhys = x;
171 elseif ft == 2
172     xPhys = HHs * x;
173 elseif ft == 3
174     xTilde = HHs * x;
175     xPhys = (tanh(beta*eta) + tanh(beta*(xTilde-eta))) ./ (tanh(
176         beta*eta) + tanh(beta*(1-eta)));
177 end
178
179 change = max(abs(x - xold));
180
181 % Print Results

```

```

179     fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n', loop, c,
180           mean(xPhys), change);
181
182     % Plot intermediate designs
183     colormap('gray');
184     scatter(ct(:,1), ct(:,2), [], 1-xPhys, 'filled');
185     axis equal off;
186     drawnow;
187
188     % Update beta parameter (continuation approach)
189     if ft == 3 && mod(loop, 60) == 0 && beta < betamax
190         beta = 2 * beta;
191         fprintf(' Beta updated to: %g\n', beta);
192     end
193 end
end

```



Figure 3:

MATLAB Code for top88 with Helmholtz Filter

```

1  %%%% TOPOLOGY OPTIMIZATION WITH HELMHOLTZ FILTER FOR CANTILEVER BEAM
2  %%%%
3  function top88_helmholtz_Q2(nelx,nely,volfrac,penalMax,rmin)
4  % input example: >> top88_cantilever(80,20,0.5,3,2)
5  % Cantilever beam with angled force at the top-right corner
6
7  %% MATERIAL PROPERTIES
8  E0 = 1;
9  Emin = 1e-9;
10 nu = 0.3;
11 penal = 0.96;
12 %%% PREPARE FINITE ELEMENT ANALYSIS
13 A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
14 A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
15 B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
16 B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
17 KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);

```

```

17 nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
18 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
19 edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*
    nely,1);
20 iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
21 jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
22
23 %% DEFINE LOADS AND SUPPORTS FOR CANTILEVER BEAM PROBLEM
24 % Initialize force vector
25 F = sparse(2*(nely+1)*(nelx+1),1);
26
27 % Calculate the force components (F = 12 at 45 )
28 F_magnitude = sqrt(12);
29 F_x = F_magnitude * cos(pi/4); % F*cos(45 )
30 F_y = F_magnitude * sin(pi/4); % F*sin(45 )
31
32 % Apply force at the top-right corner
33 topRightNode = nodenrs(1, nelx+1);
34 F(2*topRightNode-1) = F_x; % Horizontal component (positive x)
35 F(2*topRightNode) = -F_y; % Vertical component (negative y)
36
37 U = zeros(2*(nely+1)*(nelx+1),1);
38
39 % Define supports: Fixed at left end (cantilever)
40 fixeddofs = [];
41 for i = 1:nely+1
42     leftNode = nodenrs(i, 1);
43     fixeddofs = [fixeddofs, 2*leftNode-1, 2*leftNode]; % Fix both x and
        y DOFs
44 end
45
46 alldofs = 1:2*(nely+1)*(nelx+1);
47 freedofs = setdiff(alldofs, fixeddofs);
48
49 %% PREPARE HELMHOLTZ FILTER
50 Rmin = rmin/2/sqrt(3); % Conversion between classical and PDE filter
    radius
51 % Define filter stiffness matrix
52 KEF = [
53     2/3 -1/6 -1/3 -1/6
54     -1/6 2/3 -1/6 -1/3
55     -1/3 -1/6 2/3 -1/6
56     -1/6 -1/3 -1/6 2/3];
57 KEF = Rmin^2*KEF + [
58     4/9 1/9 1/9 1/9
59     1/9 4/9 1/9 1/9
60     1/9 1/9 4/9 1/9
61     1/9 1/9 1/9 4/9]/4;
62 % Setup filter FE matrices
63 edofVecF = reshape(nodenrs(1:end-1,1:end-1),nelx*nely,1);
64 edofMatF = repmat(edofVecF,1,4) + repmat([0 nely+[1 2] 1],nelx*nely,1);
65 iKF = reshape(kron(edofMatF,ones(4,1))',16*nelx*nely,1);
66 jKF = reshape(kron(edofMatF,ones(1,4))',16*nelx*nely,1);
67 sKF = reshape(KEF(:)*ones(1,nelx*nely),16*nelx*nely,1);
68 KF = sparse(iKF,jKF,sKF);
69 LF = chol(KF,'lower'); % Cholesky factorization for efficient solving
70 iTF = reshape(edofMatF,4*nelx*nely,1);
71 jTF = reshape(repmat([1:nelx*nely],4,1)',4*nelx*nely,1);

```

```

72 sTF = repmat(1/4,4*nelx*nely,1);
73 TF = sparse(iTF,jTF,sTF);
74
75 %% INITIALIZE ITERATION
76 x = repmat(volfrac,nely,nelx);
77 xPhys = x;
78 loop = 0;
79 change = 1;
80 %% START ITERATION
81 while change > 0.01
82     loop = loop + 1;
83     penal = min(penalMax, penal + 0.04);
84     %% FE-ANALYSIS
85     sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
86     K = sparse(iK,jK,sK); K = (K+K')/2;
87     tic; U(freedofs) = K(freedofs,freedofs)\F(freedofs); toc;
88     %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
89     ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
90     c = sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
91     dc = -penal*(E0-Emin)*xPhys.^(penal-1).*ce;
92     dv = ones(nely,nelx);
93
94     %% HELMHOLTZ FILTERING OF SENSITIVITIES
95     dc(:) = TF'*(LF'\(LF\'(TF*dc(:)))));
96     dv(:) = TF'*(LF'\(LF\'(TF*dv(:)))));
97
98     %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL
99     DENSITIES
100     l1 = 0; l2 = 1e9; move = 0.2;
101     while (l2-l1)/(l1+l2) > 1e-3
102         lmid = 0.5*(l2+l1);
103         xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/lmid)))));
104
105         %% HELMHOLTZ FILTERING OF DENSITIES
106         xPhys(:) = TF'*(LF'\(LF\'(TF*xnew(:)))));
107
108         if sum(xPhys(:)) > volfrac*nelx*nely, l1 = lmid; else l2 = lmid;
109         end
110     end
111     change = max(abs(xnew(:)-x(:)));
112     x = xnew;
113     %% PRINT RESULTS
114     fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, ...
115             mean(xPhys(:)),change);
116     %% PLOT DENSITIES
117     colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off;
118     drawnow;
119 end

```



Figure 4:

3 Question 3

MATLAB Code for HoneyTop90 with Projection Filter

```

1 function honey_top90_projection_Q3(HNex, HNey, volfrac, penal, rfill,
   ft)
2 %% ----- Material Properties -----
3 E0 = 1;
4 Emin = E0 * 1e-9;
5
6 %% ----- Mesh Generation and DOF Assignment -----
7 NstartVs = reshape(1:(1+2*HNex)*(1+HNey), 1+2*HNex, 1+HNey);
8 DOFstartVs = reshape(2*NstartVs(1:end-1,1:end-1)-1, 2*HNex*HNey, 1);
9 NodeDOFs = repmat(DOFstartVs, 1, 8) + repmat([2*(2*HNex+1)+[2 3 0 1] 0
   1 2 3], 2*HNex*HNey, 1);
10
11 skip = (2*HNex:2*HNex:2*HNex*HNey)' + mod(1:HNey,2)';
12 ActualDOFs = NodeDOFs(setdiff(1:2*HNex*HNey, skip), :);
13 HoneyDOFs = [ActualDOFs(2:2:end,1:2), ActualDOFs(1:2:end,:), ActualDOFs
   (2:2:end,7:8)];
14
15 Ncyi = repmat(reshape(repmat([-0.25 0.25]', HNey+1, 1), 2, HNey+1) +
   ...
16               reshape(1.5*sort(repmat((0:HNey)',2,1)), 2, HNey+1), HNex
   +1, 1);
17 Ncyi(:,1:2:end) = flip(Ncyi(:,1:2:end));
18 Ncyf = Ncyi(1:end-1,:);
19
20 HoneyNCO = [repmat((0:cos(pi/6):2*HNex*cos(pi/6)),1,HNey+1)', Ncyf(:)];
21
22 if mod(HNey,2) == 0
23     HoneyDOFs(end:-1:end-HNex+2,1:6) = HoneyDOFs(end:-1:end-HNex+2,1:6)
   - 2;
24     idx = (2*HNex+1)*(HNey+1);
25     HoneyNCO([idx-1, idx], :) = [];
26 end

```



```

27
28 [Nelem, Nnode] = deal(size(HoneyDOFs,1), size(HoneyNCO,1));
29
30 %% ----- Force and Boundary Conditions (Uniformly Distributed
    Simply Supported Beam) -----
31 L = max(HoneyNCO(:,1));
32 F = sparse(2*Nnode,1);
33
34 % Apply uniform downward load
35 totalLoad = -1; % Total load (negative = downward)
36 topNodes = find(abs(HoneyNCO(:,2) - max(HoneyNCO(:,2))) < 1e-6);
37 loadPerNode = totalLoad / length(topNodes);
38
39 F(2*topNodes) = F(2*topNodes) + loadPerNode;
40
41 % Find left and right nodes
42 leftNodes = find(abs(HoneyNCO(:,1)) < 1e-6);
43 rightNodes = find(abs(HoneyNCO(:,1) - max(HoneyNCO(:,1))) < 1e-6);
44
45 % Fix vertical DOFs at left and right nodes
46 fixeddofs = [2*leftNodes; 2*rightNodes];
47
48 % Additionally, fix horizontal movement at one left node
49 [~, idxMin] = min(HoneyNCO(:,1));
50 fixeddofs = [fixeddofs; 2*idxMin-1];
51
52 alldofs = 1:2*Nnode;
53 freedofs = setdiff(alldofs, fixeddofs);
54
55 %% ----- Assembly Preparation -----
56 iK = reshape(kron(HoneyDOFs, ones(12,1))', 144*Nelem, 1);
57 jK = reshape(kron(HoneyDOFs, ones(1,12))', 144*Nelem, 1);
58
59 KE = (E0/1000) * [
60     616.43012  92.77147  -168.07333  65.54377  -232.28511  -0.00032
        -120.65312  -83.28564  -71.60020  -92.77115  -23.81836  17.74187;
61     92.77147  509.30685  101.02751  -71.90335  0.00032  -18.03857
        -83.28564  -24.48314  -92.77179  -178.72347  -17.74187  -216.15832;
62    -168.07333  101.02751  455.74522  0.00000  -168.07333  -101.02751
        -71.60020  -92.77179  23.60185  -0.00000  -71.60020  92.77179;
63     65.54377  -71.90335  0.00000  669.99176  -65.54377  -71.90335
        -92.77115  -178.72347  -0.00000  -168.73811  92.77115  -178.72347;
64    -232.28511  0.00032  -168.07333  -65.54377  616.43012  -92.77147
        -23.81836  -17.74187  -71.60020  92.77115  -120.65312  83.28564;
65    -0.00032  -18.03857  -101.02751  -71.90335  -92.77147  509.30685
        17.74187  -216.15832  92.77179  -178.72347  83.28564  -24.48314;
66    -120.65312  -83.28564  -71.60020  -92.77115  -23.81836  17.74187
        616.43012  92.77147  -168.07333  65.54377  -232.28511  -0.00032;
67    -83.28564  -24.48314  -92.77179  -178.72347  -17.74187  -216.15832
        92.77147  509.30685  101.02751  -71.90335  0.00032  -18.03857;
68    -71.60020  -92.77179  23.60185  -0.00000  -71.60020  92.77179
        -168.07333  101.02751  455.74522  0.00000  -168.07333  -101.02751;
69    -92.77115  -178.72347  -0.00000  -168.73811  92.77115  -178.72347
        65.54377  -71.90335  0.00000  669.99176  -65.54377  -71.90335;
70    -23.81836  -17.74187  -71.60020  92.77115  -120.65312  83.28564
        -232.28511  0.00032  -168.07333  -65.54377  616.43012  -92.77147;
71     17.74187  -216.15832  92.77179  -178.72347  83.28564  -24.48314
        -0.00032  -18.03857  -101.02751  -71.90335  -92.77147  509.30685

```

```

72 ];
73
74 %% ----- Filter Preparation -----
75 Cxx = repmat([sqrt(3)/2*(1:2:2*HNex-1), sqrt(3)*(1:HNex-1)], 1, ceil(
    HNey/2));
76 Cyy = repmat(3/4 + 3/2*(0:HNey-1), HNex, 1);
77 Cyy(HNex+1:2*HNex:end)*

```



Figure 5:

MATLAB Code for top88 with Helmholtz Filter

```

1  %%%% TOPOLOGY OPTIMIZATION WITH HELMHOLTZ FILTER FOR CANTILEVER BEAM
   %%%%
2  function top88_helmholtz_Q3(nelx, nely, volfrac, penalMax, rmin)
3  % Example usage: >> top88_helmholtz_Q2(80, 20, 0.5, 3, 2)
4  % Cantilever beam with angled force at top-right corner.
5
6  %% MATERIAL PROPERTIES
7  E0 = 1; % Young's modulus of solid material
8  Emin = 1e-9; % Young's modulus of void-like material
9  nu = 0.3; % Poisson's ratio
10 penal = 0.96; % Starting penalization factor
11
12 %% PREPARE FINITE ELEMENT ANALYSIS
13 KE = element_stiffness_matrix(nu);
14 [nodenrs, edofMat, iK, jK] = prepare_fea(nelx, nely);
15
16 %% DEFINE LOADS AND SUPPORTS
17 [F, freedofs] = define_loads_supports(nelx, nely, nodenrs);
18
19 %% PREPARE HELMHOLTZ FILTER
20 [LF, TF] = prepare_helmholtz_filter(nelx, nely, nodenrs, rmin);
21
22 %% INITIALIZE ITERATION
23 x = repmat(volfrac, nely, nelx);
24 xPhys = x;
25 loop = 0;
26 change = 1;
27
28 %% ITERATIVE OPTIMIZATION LOOP
29 while change > 0.01
30     loop = loop + 1;
31     penal = min(penalMax, penal + 0.04);
32
33     % FE-Analysis
34     U = finite_element_analysis(xPhys, KE, iK, jK, freedofs, Emin, E0,
        penal);

```

```

35
36 % Objective function and sensitivity
37 [c, dc] = objective_and_sensitivity(xPhys, U, KE, nelx, nely, Emin,
    E0, penal);
38
39 % Filtering sensitivities
40 dc(:) = TF' * (LF' \ (LF \ (TF * dc(:))));
41
42 % Optimality criteria update
43 [xnew, xPhys] = optimality_criteria(x, dc, TF, LF, volfrac, nelx,
    nely);
44
45 % Compute change
46 change = max(abs(xnew(:) - x(:)));
47 x = xnew;
48
49 % Print iteration history
50 fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n', loop, c,
    mean(xPhys(:)), change);
51
52 % Plot densities
53 plot_densities(xPhys);
54 end
55 end
56
57 %% SUBFUNCTIONS
58
59 function KE = element_stiffness_matrix(nu)
60 A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
61 A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
62 B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
63 B12 = [ 2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
64 KE = 1/(1-nu^2)/24 * ([A11 A12; A12' A11] + nu*[B11 B12; B12' B11])
    ;
65 end
66
67 function [nodenrs, edofMat, iK, jK] = prepare_fea(nelx, nely)
68 nodenrs = reshape(1:(1+nelx)*(1+nely), 1+nely, 1+nelx);
69 edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1, nelx*nely, 1);
70 edofMat = repmat(edofVec, 1, 8) + repmat([0 1 2*nely+[2 3 0 1] -2
    -1], nelx*nely, 1);
71 iK = reshape(kron(edofMat, ones(8,1))', 64*nelx*nely, 1);
72 jK = reshape(kron(edofMat, ones(1,8))', 64*nelx*nely, 1);
73 end
74
75 function [F, freedofs] = define_loads_supports(nelx, nely, nodenrs)
76 ndof = 2*(nelx+1)*(nely+1);
77 F = sparse(ndof, 1);
78
79 F_magnitude = sqrt(12);
80 F_x = F_magnitude * cos(pi/4);
81 F_y = F_magnitude * sin(pi/4);
82
83 topRightNode = nodenrs(1, nelx+1);
84 F(2*topRightNode-1) = F_x;
85 F(2*topRightNode) = -F_y;
86
87 fixeddofs = [];

```

```

88     for i = 1:nely+1
89         leftNode = nodenrs(i, 1);
90         fixeddofs = [fixeddofs, 2*leftNode-1, 2*leftNode];
91     end
92
93     alldofs = 1:ndof;
94     freedofs = setdiff(alldofs, fixeddofs);
95 end
96
97 function [LF, TF] = prepare_helmholtz_filter(nelx, nely, nodenrs, rmin)
98     Rmin = rmin / (2*sqrt(3));
99     KEF = [2/3 -1/6 -1/3 -1/6;
100          -1/6 2/3 -1/6 -1/3;
101          -1/3 -1/6 2/3 -1/6;
102          -1/6 -1/3 -1/6 2/3];
103     KEF = Rmin^2 * KEF + [4/9 1/9 1/9 1/9;
104                          1/9 4/9 1/9 1/9;
105                          1/9 1/9 4/9 1/9;
106                          1/9 1/9 1/9 4/9]/4;
107     edofVecF = reshape(nodenrs(1:end-1,1:end-1), nelx*nely, 1);
108     edofMatF = repmat(edofVecF, 1, 4) + repmat([0 nely+[1 2] 1], nelx*
109         nely, 1);
109     iKF = reshape(kron(edofMatF, ones(4,1))', 16*nelx*nely, 1);
110     jKF = reshape(kron(edofMatF, ones(1,4))', 16*nelx*nely, 1);
111     sKF = reshape(KEF(:) * ones(1, nelx*nely), 16*nelx*nely, 1);
112     KF = sparse(iKF, jKF, sKF);
113     LF = chol(KF, 'lower');
114
115     iTF = reshape(edofMatF, 4*nelx*nely, 1);
116     jTF = reshape(repmat(1:nelx*nely, 4, 1)', 4*nelx*nely, 1);
117     sTF = repmat(1/4, 4*nelx*nely, 1);
118     TF = sparse(iTF, jTF, sTF);
119 end
120
121 function U = finite_element_analysis(xPhys, KE, iK, jK, freedofs, Emin,
122     E0, penal)
123     sK = reshape(KE(:) * (Emin + xPhys(:)'.^penal * (E0 - Emin)), 64*
124         numel(xPhys), 1);
125     K = sparse(iK, jK, sK);
126     K = (K + K')/2;
127     F = evalin('caller', 'F');
128     U = zeros(length(F), 1);
129     U(freedofs) = K(freedofs, freedofs) \ F(freedofs);
130 end
131
132 function [c, dc] = objective_and_sensitivity(xPhys, U, KE, nelx, nely,
133     Emin, E0, penal)
134     edofMat = evalin('caller', 'edofMat');
135     ce = reshape(sum((U(edofMat) * KE) .* U(edofMat)), 2), nely, nelx);
136     c = sum(sum((Emin + xPhys.^penal * (E0 - Emin)) .* ce));
137     dc = -penal * (E0 - Emin) * xPhys.^(penal-1) .* ce;
138 end
139
140 function [xnew, xPhys] = optimality_criteria(x, dc, TF, LF, volfrac,
141     nelx, nely)
142     l1 = 0; l2 = 1e9; move = 0.2;
143     dv = ones(size(x));
144     dv(:) = TF' * (LF' \ (LF \ (TF * dv(:))));

```

```

141
142 while (l2-l1)/(l1+l2) > 1e-3
143     lmid = 0.5*(l2+l1);
144     xnew = max(0, max(x - move, min(1, min(x + move, x .* sqrt(-dc
145         ./ dv / lmid)))));
146     xPhys(:) = TF' * (LF' \ (LF \ (TF * xnew(:))));
147     if sum(xPhys(:)) > volfrac * nelx * nely
148         l1 = lmid;
149     else
150         l2 = lmid;
151     end
152 end
153
154 function plot_densities(xPhys)
155     colormap(gray);
156     imagesc(1 - xPhys);
157     caxis([0 1]);
158     axis equal;
159     axis off;
160     drawnow;
161 end

```



Figure 6:

4 Question 4

MATLAB Code for top88 with Helmholtz Filter

```

1 % code
2 function top88(nelx, nely, volfrac, penal, rmin, ft)
3     % MATERIAL PROPERTIES
4     E0 = 1;
5     Emin = 1e-9;
6     nu = 0.3;
7     % PREPARE FINITE ELEMENT ANALYSIS
8     A11 = [12 3 -6 -3; 3 12 3 0; -6 3 12 -3; -3 0 -3 12];
9     A12 = [-6 -3 0 3; -3 -6 -3 -6; 0 -3 -6 3; 3 -6 3 -6];
10    B11 = [-4 3 -2 9; 3 -4 -9 4; -2 -9 -4 -3; 9 4 -3 -4];
11    B12 = [2 -3 4 -9; -3 2 9 -2; 4 9 2 3; -9 -2 3 2];
12    KE = 1/(1-nu^2)/24*([A11 A12; A12' A11]+nu*[B11 B12; B12' B11]);
13    nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
14    edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
15    edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],
16        nelx*nely,1);
17    iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
18    jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);

```

```

18 % DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
19 F = sparse([2*(nely+1)*nelx+2,2*(nely+1)*(nelx+1)], ...
20 [1 2],[1-1],2*(nely+1)*(nelx+1),2);
21 U = zeros(2*(nely+1)*(nelx+1),2);
22 fixeddofs = [1:2*nely+1];
23 alldofs = [1:2*(nely+1)*(nelx+1)];
24 freedofs = setdiff(alldofs,fixeddofs);
25 % PREPARE FILTER
26 Rmin = rmin/2/sqrt(3);
27 KEF = Rmin^2*[4 -1 -2 -1;-1 4 -1 -2;-2 -1 4 -1;-1 -2 -1 4]/6 + ...
28 [4 2 1 2; 2 4 2 1; ...
29 1 2 4 2; 2 1 2 4]/36;
30 edofVecF = reshape(nodenrs(1:end-1,1:end-1),nelx*nely,1);
31 edofMatF = repmat(edofVecF,1,4) + ...
32 repmat([0 nely+[1:2] 1], nelx*nely,1);
33 iKF = reshape(kron(edofMatF,ones(4,1))', 16*nelx*nely,1);
34 jKF = reshape(kron(edofMatF,ones(1,4))', 16*nelx*nely,1);
35 sKF = reshape(KEF(:)*ones(1,nelx*nely), 16*nelx*nely,1);
36 KF = sparse(iKF,jKF,sKF);
37 LF = chol(KF,'lower');
38 iTF = reshape(edofMatF,4*nelx*nely,1);
39 jTF = reshape(repmat([1:nelx*nely],4,1)',4*nelx*nely,1);
40 sTF = repmat(1/4,4*nelx*nely,1);
41 TF = sparse(iTF,jTF,sTF);
42 % INITIALIZE ITERATION
43 x = repmat(volfrac,nely,nelx);
44 xPhys = x;
45 loop = 0;
46 change = 1;
47 % START ITERATION
48 while (change > 0.01)
49     loop = loop + 1;
50     % FE-ANALYSIS
51     sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*
52         nely,1);
53     K = sparse(iK,jK,sK); K = (K+K')/2;
54     U(freedofs,:) = K(freedofs,freedofs) \F(freedofs,:);
55     % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
56     c=0;
57     dc=0;
58     for i = 1:size(F,2)
59         Ui = U(:,i);
60         ce = reshape(sum((Ui(edofMat)*KE).*Ui(edofMat),2), ...
61             nely,nelx);
62         c = c + sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
63         dc = dc- penal*(E0-Emin)*xPhys.^(penal-1).*ce;
64     end
65     dv = ones(nely,nelx);
66     % FILTERING/MODIFICATION OF SENSITIVITIES
67     if ft == 1
68         dc(:) = (TF'*(LF'\(LF\((TF*(dc(:).*xPhys(:))))) ...
69             ./max(1e-3,xPhys(:)));
70     elseif ft == 2
71         dc(:) = TF'*(LF'\(LF\((TF*dc(:))));
72         dv(:) = TF'*(LF'\(LF\((TF*dv(:))));
73     end
74     % OPTIMALITY CRITERIA UPDATE
75     l1 = 0; l2 = 1e9; move = 0.2;

```

```

75 while (l2-l1)/(l1+l2) > 1e-3
76     lmid = 0.5*(l2+l1);
77     xnew = max(0,max(x-move,min(1,min(x+move,x.*sqrt(-dc./dv/
78         lmid)))));
79     if ft == 1
80         xPhys = xnew;
81     elseif ft == 2
82         xPhys(:) = (TF'*(LF'\(LF\'(TF*xnew(:)))));
83     end
84     if sum(xPhys(:)) > volfrac*nex*nely, l1 = lmid; else l2 =
85         lmid; end
86 end
87 change = max(abs(xnew(:)-x(:)));
88 x = xnew;
89 % PRINT RESULTS
90 fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c
91     ,...
92     mean(xPhys(:)),change);
93 % PLOT DENSITIES
94 colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal;
95 axis off; drawnow;
96 end
97 end

```

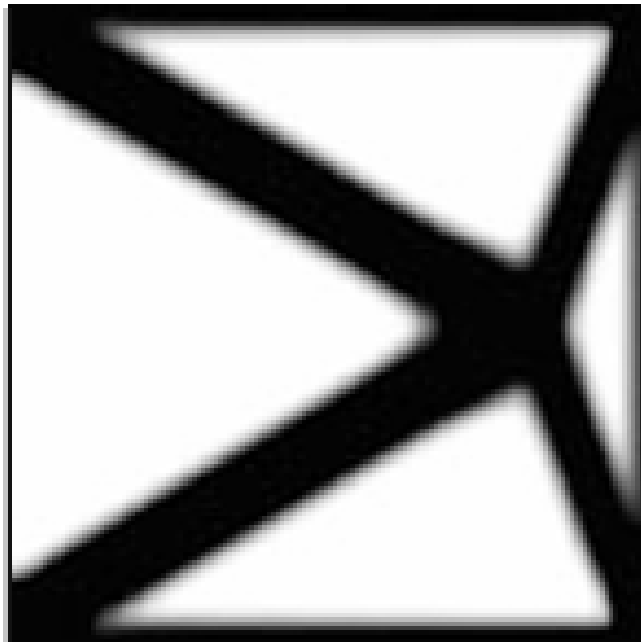


Figure 7:

MATLAB Code for HoneyTop90 with Projection Filter

```

1 function HoneyTop90(HNex,HNey,volfrac,penal,rfill,ft)
2 %% ----- Material properties -----
3 E0 = 1;
4 Emin = E0*1e-9;

```

```

5 %% ---Element connectivity, nodal coordinates, Finite element analysis
  preparation--
6 NstartVs = reshape(1:(1+2*HNex)*(1+HNey),1+2*HNex,1+HNey);
7 DOFstartVs = reshape(2*NstartVs(1:end-1,1:end-1)-1,2*HNex*HNey,1);
8 NodeDOFs = repmat(DOFstartVs,1,8) + repmat([2*(2*HNex+1) + [2 3 0 1] 0
  1 2 3 ],2*HNex*HNey,1);
9 ActualDOFs = NodeDOFs(setdiff(1:2*HNex*HNey,(2*HNex:2*HNex:2*HNex*HNey)
  ' + mod(1:HNey,2)'),' ,:);
10 HoneyDOFs = [ActualDOFs(2:2:end,1:2), ActualDOFs(1:2:end,:), ActualDOFs
  (2:2:end,7:8)];
11 Ncyi = repmat(reshape(repmat([-0.25 0.25]',HNey+1,1),2,HNey+1)+reshape
  (1.5*sort(repmat((0:HNey)',2,1)),2,(HNey+1)),HNex+1,1);
12 Ncyi(:,1:2:end) = flip(Ncyi(:,1:2:end));
13 Ncyf = Ncyi(1:end-1,:); % final arrays containing y-coordinates
14 HoneyNCO=(1)*[repmat((0:cos(pi/6):2*HNex*cos(pi/6)),1,HNey+1)' Ncyf(:)
  ];%node co
15 if(mod(HNey,2)==0)
16   HoneyDOFs(end:-1:end-(HNex)+2,1:6) = HoneyDOFs(end:-1:end-(HNex)
  +2,1:6)-2; % Updating
17   HoneyNCO([(2*HNex+1)*HNey+1;(2*HNex+1)*(HNey+1)],:) = []; % Removing
  hanging nodes
18 end
19 [Nelem,Nnode] = deal(size(HoneyDOFs,1),size(HoneyNCO,1)); % elem #,
  node #
20 F = sparse(2*((2*HNex+1)*HNey+1),1,-1,2*Nnode,1); % Applied
  load
21 U = zeros(2*Nnode,1); %
  Initializing U
22 fixeddofs = [2*(1:2*HNex+1:(2*HNex+1)*HNey+1)-1,(2*(2*HNex+1))]; %
  Fixed DOFs
23 alldofs = 1:2*Nnode; %
  Total DOFs
24 freedofs = setdiff(alldofs,fixeddofs); %
  Free DOFs
25 iK = reshape(kron(HoneyDOFs,ones(12,1))',144*Nelem,1);
26 jK = reshape(kron(HoneyDOFs,ones(1,12))',144*Nelem,1);
27 KE = E0*[616.43012 92.77147 -168.07333 65.54377 -232.28511 -0.00032
  -120.65312 -83.28564 -71.60020 -92.77115 -23.81836 17.74187;
28 92.77147 509.30685 101.02751 -71.90335 0.00032 -18.03857 -83.28564
  -24.48314 -92.77179 -178.72347 -17.74187 -216.15832;
29 -168.07333 101.02751 455.74522 0.00000 -168.07333 -101.02751 -71.60020
  -92.77179 23.60185 -0.00000 -71.60020 92.77179;
30 65.54377 -71.90335 0.00000 669.99176 -65.54377 -71.90335 -92.77115
  -178.72347 -0.00000 -168.73811 92.77115 -178.72347;
31 -232.28511 0.00032 -168.07333 -65.54377 616.43012 -92.77147 -23.81836
  -17.74187 -71.60020 92.77115 -120.65312 83.28564;
32 -0.00032 -18.03857 -101.02751 -71.90335 -92.77147 509.30685 17.74187
  -216.15832 92.77179 -178.72347 83.28564 -24.48314;
33 -120.65312 -83.28564 -71.60020 -92.77115 -23.81836 17.74187 616.43012
  92.77147 -168.07333 65.54377 -232.28511 -0.00032;
34 -83.28564 -24.48314 -92.77179 -178.72347 -17.74187 -216.15832 92.77147
  509.30685 101.02751 -71.90335 0.00032 -18.03857;
35 -71.60020 -92.77179 23.60185 -0.00000 -71.60020 92.77179 -168.07333
  101.02751 455.74522 0.00000 -168.07333 -101.02751;
36 -92.77115 -178.72347 -0.00000 -168.73811 92.77115 -178.72347 65.54377
  -71.90335 0.00000 669.99176 -65.54377 -71.90335;
37 -23.81836 -17.74187 -71.60020 92.77115 -120.65312 83.28564 -232.28511
  0.00032 -168.07333 -65.54377 616.43012 -92.77147;

```



```

38 17.74187 -216.15832 92.77179 -178.72347 83.28564 -24.48314 -0.00032
   -18.03857 -101.02751 -71.90335 -92.77147 509.30685;]/1000;% elem
   stiffness
39 %% ----- Filter preperation -----
40 Cxx = repmat([sqrt(3)/2*(1:2:2*HNex-1) sqrt(3)*(1:1:HNex-1)],1,ceil(
   HNey/2));
41 Cyy = (repmat(3/4,HNex,HNey) + repmat(3/2*(0:HNey-1),HNex,1));
42 Cyy(HNex+1:2*HNex:length(Cyy(:))) = [];
43 ct = [Cxx(1:length(Cyy))' Cyy']*(1); % Centre coordinates
44 DD = cell(Nelem,1); % Initializing
45 for j = 1:Nelem
46     Cent_dist = sqrt((ct(j,1)-ct(:,1)).^2+((ct(j,2)-ct(:,2)).^2));
47     [I,J] = find(Cent_dist<=rfill);
48     DD{j} = [I,J+(j-1),Cent_dist(I)];
49 end
50 DD = cell2mat(DD);
51 HHs = sparse(DD(:,2),DD(:,1),1-DD(:,3)/rfill);
52 HHs = spdiags(1./sum(HHs,2),0,Nelem,Nelem)*HHs;
53 %% ----- Initialization -----
54 x = volfrac*ones(Nelem,1); % Initial guess
55 [xPhys,loop,change,maxiter,dv,move] = deal(x,0,1,200,ones(Nelem,1),0.2)
   ; % Parameters
56 %% ----- Start optimization -----
57 while (change > 0.01 && loop < maxiter)
58     loop = loop + 1;
59     %% ----- Finite element analysis -----
60     sK = reshape(KE(:)*(Emin + xPhys'.^penal*(EO-Emin)),144*Nelem,1);
61     K = sparse(iK,jK,sK); %
   Global stiffness
62     U(freedofs) = decomposition(K(freedofs,freedofs),'chol','lower')\F(
   freedofs);
63     %% ----- Objective and sensitivities evaluation -----
64     ce = sum((U(HoneyDOFs)*KE).*U(HoneyDOFs),2);
65     c = sum(sum((Emin+xPhys.^penal*(EO-Emin)).*ce)); %
   Finding objective
66     dc = -penal*(EO-Emin)*xPhys.^(penal-1).*ce; % Obj
   . sensitivities
67     %% ----- Using Filters -----
68     if ft == 1
69         dc = HHs'*(x.*dc)./max(1e-3,x);
70     elseif ft == 2
71         dc = HHs'*(dc);
72         dv = HHs'*(dv);
73     end
74     %% ----- Optimality criteria update -----
75     xOpt = x;
76     [xUpp, xLow] = deal (xOpt + move, xOpt - move); % Upp. &
   low. limits
77     OcC = xOpt.*(sqrt(-dc./dv)); % Opt.
   parameter
78     inL = [0, mean(OcC)/volfrac]; % Lag.
   Mul. range
79     while (inL(2)-inL(1))/(inL(2)+inL(1))> 1e-3
80         lmid = 0.5*(inL(2)+ inL(1));
81         x = max(0,max(xLow,min(1,min(xUpp,OcC/lmid))));
82         if mean(x)>volfrac, inL(1) = lmid; else, inL(2) = lmid; end
83     end

```

```

84  if ft == 1 || ft == 0, xPhys = x; elseif ft == 2, xPhys = HHs'*x; end
85  change = max(abs(xOpt-x));
86  %% ----- Results printing -----
87  fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, mean(
      xPhys),change);
88  %% ----- Plotting intermediate designs -----
89  colormap('gray'); scatter(ct(:,1),ct(:,2),[],1-xPhys,'filled'); axis
      off equal; pause(1e-6);
90 end

```

References

- 1 The code used to generate plots is in the zip file.