Development of LiveLink for MATLAB code for topology optimisation of conjugate heat transfer problems

K.O.V.B.Rama Reddy[1] and Dr. Sourav Rakshit[1]

[1] Indian Institute of Technology Madras, Chennai, 600036, Tamil Nadu, India

Abstract: This work employs topology optimization of conjugate heat transfer problems in LiveLink for MATLAB, which couples COMSOL Multiphysics software with MATLAB programming via COMSOL Application Programming Interface (API). The code is explained for the design of a natural convection micropump where finite element analysis is complicated and unhandy to code. Laminar and heat transfer modules of COMSOL Multiphysics 6.1 are used to solve the governing equations. The inputs required for adjoint sensitvity analysis and optimisation (performed in MATLAB R2021b) are extracted from COMSOL Multiphysics 6.1 via LiveLink for MATLAB. A detailed code for the natural convection micropump example is provided in the appendix.

Keywords: Topology Optimisation, Conjugate Heat Transfer, LiveLink for MATLAB.

1 Introduction

Topology optimization can be defined as mathematical method of distributing a material quantity in a volume in an optimum manner to achieve desired objective while satisfying predefined constraints[1]. These constraints include the governing equations of the system, a prescribed volume fraction, manufacturing constraints etc. Applications of topology optimization include structural analysis, fluid flows, heat conduction, electromagnetism and conjugate heat transfer problem.

Topology optimization of conjugate heat transfer problems is a relatively intricate problem due to the coupled nature of Navier-Stokes equation, continuity equation and energy equation. In a general topology optimisation code, the major parts are finite element analysis, sensitivity analysis and core optimisation algorithm like Method of Moving Asymptotes(MMA), Optimality Criteria method etc. Writing a complete code from scratch is cumbersome due to finite element analysis and it's requirement of stabilisation techniques for thermo-fluid problems[2]. But most often, the problem requires rigorous tuning of parameters in interpolation schemes, sensitivity analysis and optimisation algorithm which may be difficult to access in commercial softwares. COMSOL LiveLink for MATLAB enables us to couple the commercial COMSOL Multiphysics software with MATLAB programming via COMSOL Application Programming Interface (API).

In this work, a comprehensive COMSOL LiveLink for MATLAB code for topology optimisation of conjugate heat transfer (inspired from [3]) is demonstrated. A natural convection example (design of a micropump) [2] is used to explain the details of the code. COMSOL Laminar Flow interface is used to solve the steady state Navier-Stokes equation and continuity equation. The COMSOL Heat Transfer in fluids module is used to solve energy equation. Even though the code is exlplained using an micropump example, it can be easily interpreted for other conjugate heat transfer problems. Study steps are defined in COMSOL Multiphysics for finite element analysis and adjoint sensitivity analysis. Section 2 provides theory regarding governing equations, optimisation, interpolation schemes, density and projection schemes, sensitivity analysis, volume constraint, equation for design variable field. Section 3 explains details of micropump problem. Section 4 provides important details of the code provided in the appendix.

2 Theory

2.1 Governing equations

The non-dimensional governing equations for natural convection problems for fluids and solids combined after assuming boussinesq approximation will be as follows [2]:

$$\vec{\nabla}.\vec{u} = 0 \tag{1}$$

$$(\vec{u}.\vec{\nabla})\vec{u} = \vec{\nabla}.(-pI) + Pr\vec{\nabla}.(\vec{\nabla}\vec{u} + (\vec{\nabla}\vec{u})^T) - \alpha\vec{u} - GrPr^2T\vec{e} \quad (2)$$

$$\vec{u}.\vec{\nabla}T - \vec{\nabla}.\left(K(\vec{x})\vec{\nabla}T\right) = s_T \quad (3)$$

where $\vec{u}$ is dimensionless velocity, T is dimensionless temperature, μ is dynamic viscosity, α is inverse local permeability, p is dimensionless pressure, $\vec{e}$ is unit vector along acceleration due to gravity, s_T is spatially varying volumetric heat generation. $K(\vec{x})$ is effective conductivity defined as:

$$\begin{aligned} K(\vec{x}) &= 1 \quad if \quad \mathbf{x} \in \Omega_f \quad (fluid) \\ &= \frac{1}{C_k} \quad if \quad \mathbf{x} \in \Omega_s \quad (solid) \end{aligned} \quad (4)$$

where

$$C_k = \frac{k_f}{k_s} \quad (5)$$

Here k_f is thermal conductivity of fluid, k_s is thermal conductivity of solid.

The equations are non-dimensionalised using the following scales [2]:

$$\begin{aligned} \vec{u^*} &= U\vec{u} \\ \vec{x^*} &= L\vec{x} \\ p^* &= \rho_0 U^2 p \\ T^* &= \Delta T T + T_0 \\ \dot{s_T^*} &= \rho_0 c_p \Delta T \frac{U}{L} \dot{s_T} \end{aligned} \quad (6)$$

where the dimensional variables are marked with an asterisk(*). Here, L is a reference length, U is a reference velocity, ΔT is a reference temperature difference, T_0 is a reference temperature, β is coefficient of thermal expansion, ρ_0 is reference density at reference temperature T_0, c_p is specific heat capacity . Alexendarsen et.al [2] defined reference velocity as:

$$U = \frac{\Gamma}{L} \quad (7)$$

Γ is the thermal diffusivity of the fluid.

Inverse local permeability α is defined as:

$$\alpha = \frac{Pr}{Da} \quad (8)$$

Prandtl number Pr is defined as:

$$Pr = \frac{\nu}{\Gamma} \quad (9)$$

Darcy number Da is defined as:

$$Da = \frac{\kappa}{L^2} \quad (10)$$

Grashoff number is defined as

$$Gr = \frac{g\beta\Delta T L^3}{\nu^2} \quad (11)$$

An equation for design variable is defined in a weak form PDE to introduce it as an auxiliary dependent variable [3]:

$$\int_{\Omega_d} \delta\gamma d\Omega_d = 0 \quad (12)$$

where γ is design variable which is function of a point $\vec{x}$.

2.2 Optimization problem

Topology optimization of the micropump problem can be mathematically stated as follows:

$$\underset{\gamma \in R^n}{maximise} : \phi = \int_{\Gamma_{ob}} \vec{u}.\vec{n} d\Gamma \quad (13)$$

$$\begin{aligned} \text{subject to} &: R(\vec{\gamma}, \vec{s}(\gamma)) = 0 \\ &: \phi_1(\vec{\gamma}) = \frac{1}{\int_{\Omega_d} d\Omega} \int_{\Omega_d} \gamma dV - V_f \leq 0 \\ &: 0 \leq \gamma(\vec{x}) \leq 1 \quad \forall \vec{x} \in \Omega_d \end{aligned} \quad (14)$$

where $\vec{n}$ is normal unit vector, $\vec{s}$ is a vector of state variables, R is the residual of discretized physics problem , ϕ_1 limits the amounts the amount of material to be distributed in the design domain, V_f is the prescribed fluid volume fraction, Ω_d is the design domain. $\vec{\gamma}$ is a vector of density variables γ which is function of a point $\vec{x}$ in the domain.

2.3 Interpolation schemes

We used RAMP interpolation for thermal conductivity and α:

$$K = \frac{\gamma(C_k(1+q_f) - 1) + 1}{C_k(1 + q_f\gamma)} \quad (15)$$

$$\alpha = \alpha_f + (\alpha_s - \alpha_f)\frac{1-\gamma}{1+q_r\gamma} \quad (16)$$

where q_f, q_r are RAMP parameters. $\alpha_f = 0$ and $\alpha_s = \frac{\mu}{DaD^2}$.

2.4 Density filter and projection scheme

In order to avoid checkerboard patterns and mesh independent designs, density filters are generally used [2][3]. The predefined neighbourhood of an element is defined as follows:

$$N_e = \{i | \parallel \vec{x_i} - \vec{x_e} \parallel \leq R_{min}\} \quad (17)$$

where R is the filter radius and $\vec{x_i}$ is spatial location of element i and $\vec{x_e}$ is spatial location of element e. The physical relative densities are given by the relation:

$$\bar{\gamma} = \frac{\sum_{i \in N_e} W(\vec{x_i}) v_i \gamma_i}{\sum_{i \in N_e} W(\vec{x_i}) v_i} \quad (18)$$

where v_i is the volume of the element i, and $W(\vec{x_i})$ is a weighting function defined as :

$$W(\vec{x_i}) = max(0, R_{min} - \parallel \vec{x_i} - \vec{x_e} \parallel) \tag{19}$$

The most widely used projection method is hyperbolic projection method to get clear separation between solid and fluid which is used in this work:

$$\bar{\bar{\gamma}} = \frac{tanh(\beta\theta) + tanh(\beta(\bar{\gamma} - \theta))}{tanh(\beta\theta) + tanh(\beta(1 - \theta))} \tag{20}$$

where θ is a threshold control variable, β is a projection parameter. The sensitivities are modified accordingly using chain rule:

$$\frac{\partial f}{\partial \gamma_j} = \sum_{i \in \mathbb{N}} \frac{\partial f}{\partial \bar{\bar{\gamma}}_i} \frac{\partial \bar{\bar{\gamma}}_i}{\partial \bar{\gamma}_i} \frac{\partial \bar{\gamma}_i}{\partial \gamma_j} \tag{21}$$

where f represents objective function or constraint.

2.5 Sensitivity analysis

The sensitivities can be calculated as follows:

$$\frac{d\phi}{d\vec{\gamma}} = \frac{\partial \phi}{\partial \vec{\gamma}} + \vec{\lambda}^T \left(\frac{\partial R}{\partial \vec{\gamma}} \right) \tag{22}$$

Now the langrange multipliers($\vec{\lambda}$) are choosen such that it satisfies the following equation:

$$\frac{\partial \phi^T}{\partial \vec{s}} = -\vec{\lambda} \frac{\partial R^T}{\partial \vec{s}} \tag{23}$$

For the flow rate objective function,

$$\frac{\partial \phi}{\partial \vec{s}} = \int_{\partial \Omega_{obj}} (\psi_u) d\partial\Omega \tag{24}$$

where ψ_u is finite element shape function of velocity function in x-direction. The derivative of the constraint function ϕ_1 is :

$$\frac{\partial \phi_1}{\partial \vec{\gamma}} = \frac{1}{\int_{\Omega_d} d\Omega} \int_{\Omega_d} \psi_\gamma \partial\Omega \tag{25}$$

where ψ_γ is finite element shape function of design variable γ.

3 Micropump problem

The code is explained using micropump problem [2] as illustrated in figure 1. In micropump problem, fluid motion is due to the differential heating of walls. The objective is to find a fluid flow path through the solid area(grey area in Figure 1) to maximize the flow rate at the objective boundary Γ_{mf}. A maximum fluid volume fraction of 0.5 is considered. All lengths are relative to 'L'. Temperature profiles are given at Γ_1 and Γ_3. At every other boundary, it is thermally insulated. All boundaries have no-slip boundary conditions.

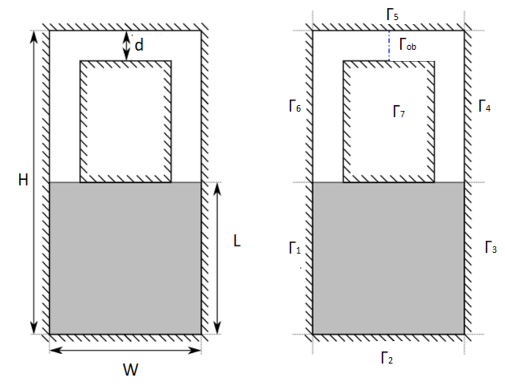
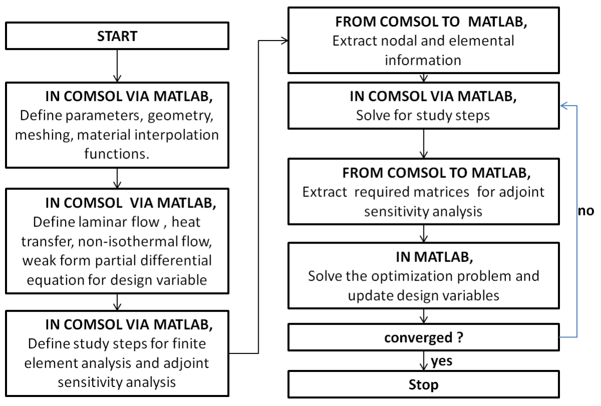


Figure 1: Schematic illustration of the layout and boundary conditions for the natural convection micropump problem. White denotes fluid and grey denotes the design domain and initial condition is solid.

4 Implementation in LiveLink for MATLAB

The implementation of the code given in this paper is very similar to the approach in [3]. The structure of the code is given as a flow chart in figure 2. Only important details of the code are explained in the remainder of this section. The reader is referred to paper by Kook.J et al.[3] for further details of the code implementation and LiveLink for MATLAB manual [5] for simplified API syntax. The constrained optimization problem is solved using a MATLAB implementation of the MMA algorithm provided by Svanberg [6][7].



START
IN COMSOL VIA MATLAB, Define parameters, geometry, meshing, material interpolation functions.
IN COMSOL VIA MATLAB, Define laminar flow , heat transfer, non-isothermal flow, weak form partial differential equation for design variable
IN COMSOL VIA MATLAB, Define study steps for finite element analysis and adjoint sensitivity analysis
FROM COMSOL TO MATLAB, Extract nodal and elemental information
IN COMSOL VIA MATLAB, Solve for study steps
FROM COMSOL TO MATLAB, Extract required matrices for adjoint sensitivity analysis
IN MATLAB, Solve the optimization problem and update design variables
converged ?
no
yes
Stop

Figure 2: Schematic illustration of the layout of the code.

All the parameters related to geometry, boundary conditions, initial conditions, density, projection and interpolation schemes, dimensionless numbers are defined in the very beginning of the code (lines 10-22). A mapped mesh is used (lines 41-46). Two sets of variables are defined (lines 48-63) where $var1$ is for design domain $var2$ is for rest of the domain. As discussed earlier, the governing equations of conjugate heat transfer problems are conservation of mass, momentum and energy equations of both solid and fluid domains combined. These equations are defined in laminar flow module and heat transfer in fluids module of COMSOL Multiphysics as given in lines $65-97$.A non-isothermal flow module is defined in lines $98-102$.

Two study steps are defined in lines $112-135$. First study step $std1$ is for finite element analysis of the problem where the state variables are solved for keeping the design variable as constant.The matrix $\frac{\partial R}{\partial \vec{s}}$ in equation 23 is also obtained using this study step (line 187). The second study step $std2$ is used for calculation of $\frac{\partial \mathbf{R}}{\partial \vec{\gamma}}$ in equation 22 (lines $193-194$) and in evaluating volume constraint (lines $155-160$). Mesh and nodal information are obtained in lines 154-159. The main loop (lines 184-244) is very similar to the code in Kook.J et.al. [3] except for MMA subroutine and convergence criteria.

5 Results

The optimized design for maximizing mass flow rate in the micropump problem is given in figure 3. The ratio of thermal conductivities of fluid to solid is taken as 0.01. The length ‘L’ and width ‘W’ of design domain are taken as 1. Total height ‘H’ of micropump is taken as 2. Thickness ‘d’ is taken as 0.2. Prandtl number is taken as 1. Grashoff number is taken as 1000. Temperatures at Γ_1 and Γ_3 are taken as 1 and 0 respectively. The total number of mesh elements are 10007. The velocity and temperature profiles of the final output are given in figures 4 and 5 respectively.

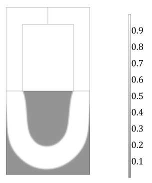


Figure 3: Optimized design for maximum clockwise mass flow for natural convection micropump . White color represents fluid and grey represents solid

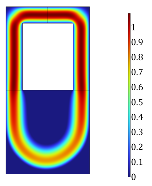


Figure 4: Velocity profile for maximum clockwise mass flow for natural convection micropump.

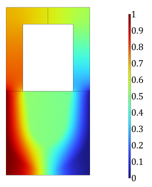


Figure 5: Temperature profile for maximum clockwise mass flow for natural convection micropump.

6 Conclusion

This work demonstrates that COMSOL LiveLink for MATLAB offers the possibility to reduce the effort of implementing finite element analysis while being able to control all the parameters in topology optimisation of conjugate heat transfer problems. The code is explained using a natural convection micropump example. The well-structured code given in the Appendix can be extended to other conjugate heat transfer problems.

Appendix

```
clc ;close ALL ; clear ALL;
import com.comsol.model.*
import com.comsol.model.util.*
model = ModelUtil.create('Model');
model.modelNode.create('comp1', true);
model.geom.create('geom1', 2);
model.geom('geom1').model('comp1');
model.component('comp1').baseSystem('none');
%% PARAMETERS
model.param.set('L', '1', 'length and width of the design domain');
model.param.set('W', '0.2', 'width of the channel');
model.param.set('H', '2', 'height of the pump');
model.param.set('T0', '300[K]', 'reference temperature');
model.param.set('Gr', '10^3', 'Grashoff number');
model.param.set('Pr', '1', 'Prandtl number');
model.param.set('Ck', '10^(-2)', 'thermal conductivity ratio');
model.param.set('q', '10^4', 'convex parameter');
model.param.set('alpha_max', '10^6', 'maximum brinkmann coefficient');
model.param.set('alpha_min', '0.001', 'minimum brinkmann coeffiecient');
model.param.set('qf', '10^4', 'RAMP parameter conductivity interpolation');
BETA = 1;
eta = 0.5; Rmin = 0.0282;
%% FUNCTION DEFINITIONS
model.component('comp1').func.create('an1', 'Analytic');
model.component('comp1').func('an1').set('expr', ['alpha_min + (alpha_max -
     alpha_min)*((1-x)/(1+q*x))']);
model.component('comp1').func.create('an2', 'Analytic');
model.component('comp1').func('an2').set('expr', '(x*(Ck*(1+qf)-1)+1)/(Ck
    *(1+qf*x))');
%% GEOMETRY
model.component('comp1').geom('geom1').create('sq1', 'Square');
model.component('comp1').geom('geom1').feature('sq1').set('size', 'L');
model.component('comp1').geom('geom1').create('pol1', 'Polygon');
model.component('comp1').geom('geom1').feature('pol1').set('source', 'table
    ');
model.component('comp1').geom('geom1').feature('pol1').set('table', {'0' 'L
    ';  '0' 'H';  'L' 'H';  'L' 'L'; 'L-W' 'L'; 'L-W' 'H-W'; 'W' 'H-W'; 'W'
    'L'; '0' 'L'});
model.component('comp1').geom('geom1').create('ls1', 'LineSegment');
model.component('comp1').geom('geom1').feature('ls1').set('specify1', '
    coord');
model.component('comp1').geom('geom1').feature('ls1').set('coord1', {'L/2'
    'H-W'});
model.component('comp1').geom('geom1').feature('ls1').set('specify2', '
    coord');
model.component('comp1').geom('geom1').feature('ls1').set('coord2', {'L/2'
    'H'});
model.component('comp1').geom('geom1').run;
%% MESHING
model.component('comp1').mesh.create('mesh1');
model.component('comp1').mesh('mesh1').create('fq1', 'FreeQuad');
model.component('comp1').mesh('mesh1').create('map1', 'Map');
model.component('comp1').mesh('mesh1').feature('size').set('hauto', 2);
model.component('comp1').mesh('mesh1').feature('size').set('table', 'cfd');
model.component('comp1').mesh('mesh1').run;
%% VARIABLES
model.component('comp1').variable.create('var1');
model.component('comp1').variable('var1').set('alpha', 'an1(gamma)');
model.component('comp1').variable('var1').set('K_var', 'an2(gamma)');
model.component('comp1').variable('var1').set('mu_var', 'Pr');
model.component('comp1').variable('var1').set('cp_var', '1');
model.component('comp1').variable('var1').set('rho_var', '1');
model.component('comp1').variable('var1').selection.geom('geom1', 2);
model.component('comp1').variable('var1').selection.set([1]);
model.component('comp1').variable.create('var2');
model.component('comp1').variable('var2').set('alpha', '0');
model.component('comp1').variable('var2').set('K_var', '1');
model.component('comp1').variable('var2').set('mu_var', 'Pr');
model.component('comp1').variable('var2').set('cp_var', '1');
model.component('comp1').variable('var2').set('rho_var', '1');
model.component('comp1').variable('var2').selection.geom('geom1', 2);
model.component('comp1').variable('var2').selection.set([2 3]);
%% PHYSICS DEFINITION
model.component('comp1').physics.create('spf', 'LaminarFlow', 'geom1');
model.component('comp1').physics('spf').create('vf1', 'VolumeForce', 2);
model.component('comp1').physics('spf').feature('vf1').selection.all ;
model.component('comp1').physics('spf').feature('vf1').set('F', {'-alpha*u'; '-alpha*v'; '0'});
model.component('comp1').physics('spf').create('vf2', 'VolumeForce', 2);
model.component('comp1').physics('spf').feature('vf2').selection.all;
model.component('comp1').physics('spf').feature('vf2').set('F', {'0'; 'Gr*(Pr^2)*T'; '0'});
model.component('comp1').physics('spf').create('prpc1', 'PressurePointConstraint', 0);
model.component('comp1').physics('spf').feature('prpc1').selection.set([12]);
model.component('comp1').physics('spf').create('weak1', 'WeakContribution',1);
model.component('comp1').physics('spf').feature('weak1').selection.set([9]);
model.component('comp1').physics('spf').feature('weak1').set('weakExpression', '-test(u)');
model.component('comp1').physics('spf').feature('fp1').set('rho_mat', 'userdef');
model.component('comp1').physics('spf').feature('fp1').set('rho', 'rho_var');
model.component('comp1').physics('spf').feature('fp1').set('mu_mat', 'userdef');
model.component('comp1').physics('spf').feature('fp1').set('mu', 'mu_var');
model.component('comp1').physics('spf').prop('AdvancedSettingProperty').set('PseudoTimeSetting', 'Off');
model.component('comp1').physics('spf').prop('ShapeProperty').set('order_fluid', 2);
model.component('comp1').physics.create('ht', 'HeatTransferInFluids', 'geom1');
model.component('comp1').physics('ht').create('temp1', 'TemperatureBoundary', 1);
model.component('comp1').physics('ht').feature('temp1').selection.set([1]);
model.component('comp1').physics('ht').feature('temp1').set('T0', 1);
model.component('comp1').physics('ht').create('temp2', 'TemperatureBoundary
```

```matlab
', 1);
model.component('comp1').physics('ht').feature('temp2').selection.set([14]);
model.component('comp1').physics('ht').feature('temp2').set('T0', 0);
model.component('comp1').physics('ht').feature('fluid1').set('Cp_mat', 'userdef');
model.component('comp1').physics('ht').feature('fluid1').set('Cp', 'cp_var');
model.component('comp1').physics('ht').feature('fluid1').set('rho_mat', 'userdef');
model.component('comp1').physics('ht').feature('fluid1').set('rho', 'rho_var');
model.component('comp1').physics('ht').feature('fluid1').set('k_mat', 'userdef');
model.component('comp1').physics('ht').feature('fluid1').set('k', 'K_var');
model.component('comp1').physics('ht').feature('init1').set('Tinit', 0);
model.component('comp1').physics('ht').prop('ShapeProperty').set('boundaryFlux_temperature', false);
model.component('comp1').multiphysics.create('nitf1', 'NonIsothermalFlow', 2);
model.component('comp1').multiphysics('nitf1').set('SpecifyDensity', 'Custom');
model.component('comp1').multiphysics('nitf1').set('BoussinesqApproximation', true);
model.component('comp1').multiphysics('nitf1').set('rho', 'rho_var');
model.component('comp1').multiphysics('nitf1').set('includeViscousDissipation', false);
%% PHYSICS FOR GAMMA
model.physics.create('w', 'WeakFormPDE', 'geom1');
model.physics('w').field('dimensionless').field('gamma');
model.physics('w').field('dimensionless').component({'gamma'});
model.physics('w').selection.set([1]);
model.physics('w').prop('ShapeProperty').set('order', 1);
model.physics('w').prop('EquationForm').set('form', 'Automatic');
model.physics('w').feature('wfeq1').set('weak', 'test(gamma)');
model.physics('w').feature('init1').set('gamma', 0);
%% STUDY STEP FOR FINITE ELEMENT ANALYSIS
std1 = model.study.create('std1');
std1.create('stat', 'Stationary');
std1.feature('stat').set('useadvanceddisable', true);
model.study('std1').feature('stat').set('activate', {'spf' 'on' 'ht' 'on' 'w' 'off' 'frame:spatial1' 'on' 'frame:material1' 'on'});
std1.feature('stat').set('disabledphysics', {'spf/weak1'});
sol1 = model.sol.create('sol1');
sol1.study('std1');
sol1.attach('std1');
st1 = sol1.create('st1', 'StudyStep');
v1 = sol1.create('v1', 'Variables');
s1 = sol1.create('s1', 'Stationary');
sol1.attach('std1');
sol1.runAll;
%% STUDY STEP FOR ADJOINT SENSITIVITY ANALYSIS
std2 = model.study.create('std2');
std2.create('stat', 'Stationary');
std2.feature('stat').set('useadvanceddisable', true);
std2.feature('stat').set('disabledphysics', {'ht/temp1' 'ht/temp2' 'spf/
    prpc1'});
sol2 = model.sol.create('sol2');
sol2.study('std2');
sol2.create('st1', 'StudyStep');
sol2.create('v1', 'Variables');
sol2.runAll;
%% POSTPROCESSING - DESIGN PLOT
 pg1 = model.result.create('pg1', 2);
surf1 = pg1.feature.create('surf1', 'Surface');
surf1.set('expr', 'gamma');
surf1.set('coloring', 'gradient');
surf1.set('bottomcolor', 'custom');
surf1.set('custombottomcolor', [0.58 0.58 0.58]);
 pg1.run;
%% MESH AND NODAL INFORMATION
info = mphxmeshinfo(model);
id_names = info.dofs.nameinds;
id_dofnames = find(strcmp(info.dofs.dofnames, 'comp1.gamma')) - 1;
igamma = find(id_names == id_dofnames);
iuvpT1 = find(id_names ~= id_dofnames);
iuvpT = iuvpT1(2:end) -1;
coords = info.dofs.coords;
gx = coords(1, igamma);
gy = coords(2, igamma);
%% VOLUME CONSTRAINT
Lstrt = mphmatrix(model, 'sol2', 'Out', {'L'}, 'initmethod', 'init');
Ltilde = Lstrt.L(iuvpT);
U0 = mphgetu(model, 'soltag', 'sol1');
Vgamma = Lstrt.L(igamma-1);
Vdomain = sum(Vgamma);
gamma = U0(igamma);
%% MMA PARAMETERS
 mma.m = 1; mma.n = numel(gamma) ;mma.xold1 = gamma; mma.xold2 = mma.xold1;
 mma.U = ones(mma.n,1); mma.L = zeros(mma.n,1); mma.epsimin = 10^(-7) ;mma.
     raa0 =0.00001; mma.move = 0.05;
 mma.albefa = 0.1; mma.asyinit = 0.5; mma.asyincr = 1; mma.asydecr = 0.6;
     mma.a0 = 1;
 mma.a = 0; mma.c = 100; mma.d = 1;
 mma.xmin = 0;mma.xmax = 1;
%% PREPARATION FOR WHILE LOOP
 iter = 1; iterMax = 150; change = Inf; f_old = Inf;
 Phi_old = Inf;
 change_limit = 0.001; change_count = 5; count = 0;
 gammaTilde = gamma; gammaPhys = gammaTilde; volfrac = 0.5;
 q_in = 10^0;
 mphplot(model, 'pg1');
 %% PREPARE FILTER
iH = cell(length(igamma), 1); jH = cell(length(igamma), 1); kH = cell(
    length(igamma), 1);
for i = 1:length(igamma)
    dist = sqrt((gx - gx(i)).^2 + (gy - gy(i)).^2)';
    jH{i} = find(dist <= Rmin);
    kH{i} = max(0, Rmin - dist(jH{i}));
    iH{i} = i * ones(length(kH{i}), 1);
end
iH = cell2mat(iH); jH = cell2mat(jH); kH = cell2mat(kH);
```

```matlab
H = sparse(iH, jH, kH); Hs = sum(H, 2);
%% WHILE LOOP
while (iter < iterMax + 1 && count < change_count)
%% DSA STEP 1: ADJOINT SOLUTION
Kstrt1 = mphmatrix(model, 'sol1', 'Out', {'Kc', 'Null', 'Nullf', 'uscale'},
    'initmethod', 'sol', 'initsol', 'sol1');
U0 = mphgetu(model, 'soltag', 'sol1');
Ladj = Ltilde;
lambdac = Kstrt1.Kc \ (Kstrt1.Nullf.'*-Ladj);
lambda = Kstrt1.uscale .*(Kstrt1.Null* lambdac);
 %% DSA STEP 2: DERIVATIVE of RESIDUAL
 Kstrt2 = mphmatrix(model, 'sol2', 'Out', {'K'}, 'initmethod', 'sol', '
    initsol', 'sol1');
 dR = Kstrt2.K(iuvpT, igamma-1);
 %% OBJECTIVE FUNCTION & SENSITIVITY
 Phi = mphint2(model, '-u', 'line', 'selection', [9]);
 dPhi = dR.' * lambda;
 constr =  ((gammaPhys)' * Vgamma / (volfrac * Vdomain))-1;
 dconst =  Vgamma / (volfrac * Vdomain);
 %% INPUT: OBJECTIVE & CONSTRAINT
  f = Phi;
  g = constr;
   %% SENSITIVITY
  df = dPhi;
 %% FILTERING/MODIFICATION OF SENSITIVITY
        dgamma = (BETA * sech(BETA * (gammaTilde - eta)).^2) / (tanh(BETA *
            eta) + tanh(BETA * (1 - eta)));
        df = (H * ((dPhi .* dgamma) ./ Hs));
        dg = H * ((dconst .* dgamma) ./ Hs);
        %% OPTIMIZER: MMA
        [gammaNew,y,z,lambda1,ksi,eta1,mu1,zeta,s,mma.xmin,mma.xmax] =
           mmasub(mma.m,mma.n,iter,gamma,mma.L,mma.U,mma.xold1,mma.xold2,f,
           df,g,dg',mma.xmin,mma.xmax,mma.a0,mma.a,mma.c,mma.d);
          change = abs(Phi_old - Phi)/abs(Phi_old);
          if change < change_limit
              count = count + 1;
          else count = 0;
          end
          f_old = f; Phi_old = Phi;
        mma.xold2=mma.xold1; mma.xold1=gamma; gamma=gammaNew;
        %% BETA CONTINUATION SCHEME
        if (mod(iter, 25) == 0 && BETA < 32 && iter > 1)
            BETA = min(2 * BETA, 32); fprintf('Parameter BETA increased to
               %g.\n', BETA);
        end
         %% CONVEX PARAMETER CONTINUATION SCHEME
        if (mod(iter, 25) == 0 && q_in < 10^7 && iter > 1)
            q_in = min(10 * q_in, 10^7);
            fprintf('Parameter q increased to %g.\n', q_in);
             model.param.set('q', q_in);
        end
        %% APPLY FILTER
        gammaTilde = (H * gamma(:))./ Hs;
        gammaPhys(:) = (tanh(BETA * eta) + tanh(BETA * (gammaTilde - eta)))
            / (tanh(BETA * eta) + tanh(BETA * (1 - eta)));
        %% UPDATE gamma
        U0(igamma) = gammaPhys;
        sol1.setU(1, U0);
        sol1.createSolution;
        v1.set('initmethod', 'sol'); v1.set('initsol', 'sol1'); v1.set('
           notsolmethod', 'sol'); v1.set('notsol', 'sol1');
        model.sol('sol1').runAll;
        %% PRINT RESULTS/PLOT DENSITIES
        fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n', iter,-Phi,
           mean(gammaPhys(:)), change);
        mphplot(model, 'pg1'); title(['Iteration = ', num2str(iter)]);
        axis equal; axis off; drawnow; shg; pause(0.1)
        iter = iter + 1;
 end
mphsave(model, 'Natural_convection_pump.mph');
mphlaunch(model);
```

References

[1] Dbouk T. A review about the engineering design of optimal heat transfer systems using topology optimization. Applied Thermal Engineering. 2017 Feb 5;112:841-54.

[2] Alexandersen J, Aage N, Andreasen CS, Sigmund O. Topology optimisation for natural convection problems. International Journal for Numerical Methods in Fluids. 2014 Dec 10;76(10):699-721.

[3] Kook J, Chang JH. A high-level programming language implementation of topology optimization applied to the acoustic-structure interaction problem. Structural and Multidisciplinary Optimization. 2021 Dec;64(6):4387-408.

[4] Wang F, Lazarov BS, Sigmund O. On projection methods, convergence and robust formulations in topology optimization. Structural and multidisciplinary optimization. 2011 Jun;43:767-84.

[5] COMSOL Inc. (2020a). COMSOL Multiphysics LiveLink For MATLAB Users Guide (version 5.6).

[6] Svanberg K. The method of moving asymptotes—a new method for structural optimization. International journal for numerical methods in engineering. 1987 Feb;24(2):359-73.

[7] A MATLAB implementation, mmasub, of the MMA optimization algorithm [6] can be obtained (free of charge) for academic purposes) from Krister Svanberg, KTH, Sweden. E-mail: krille@math.kth.se