Regenerative Nozzle Wall Cooling

AERO 423 - Aerospace Computational Methods

University of Michigan - Ann Arbor

April 20, 2020

Prepared by:

Drake Rundell

Image Source: https://en.wikipedia.org/wiki/Bell_nozzle

Table of Contents

I List of Figures

II List of Equations

III Problem Description

The hot gases inside rocket combustion chambers and nozzles can melt the surrounding materials. As a result, many rocket engines use regenerative cooling to make steady-state operation possible. In regenerative cooling, the fuel (or oxidizer) flows through an array of passages in the nozzle and thrust chamber walls, before being injected into the thrust chamber. The process is called regenerative because the heat lost from the hot combustion gases is not wasted: instead, it raises the temperature of the fuel before combustion.

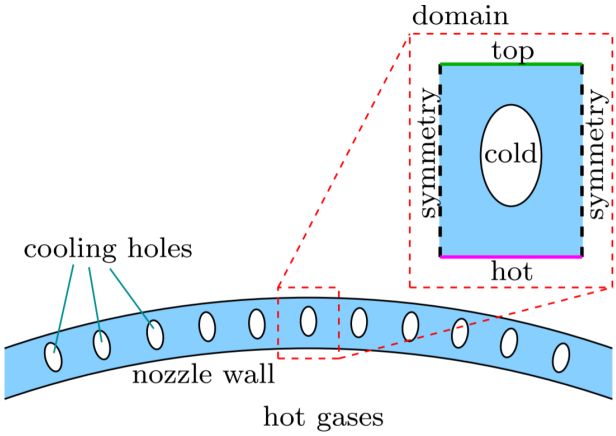

Figure 1: *Cross-Section of a regeneratively-cooled nozzle wall*

In this project, the heat transfer in a regeneratively-cooled nozzle wall will be simulated using a finite-element method. A diagram of the nozzle wall cross-section is shown in Figure 1. The nozzle wall is made of steel, with milled cooling holes. The nozzle diameter is much greater than the wall thickness, and this allows for the neglecting of the curvature and just analyze a rectangular domain. In particular, assuming the cooling passages are evenly-spaced, the analysis domain is limited to that of the dashed rectangular region shown in the figure.

Heat Transfer

Assume that the heat transfer between the steel and the hot/cold fluid occurs primarily by convection,

$$\vec{q} \cdot \vec{n} = h_{ext} \left(T - T_{ext} \right) \tag{1}$$

where $\vec{n}$ is the outward-pointing normal vector so that $\vec{q} \cdot \vec{n}$ is the heat flux out of the material. T is the material temperature on the interface, h_{ext} is the convective heat transfer coefficient. and T_{ext} is the bulk temperature of the external (hot or cold) gas. It is assumed that h_{ext} and T_{ext} for this analysis are constant. Within the steel, heat is transferred by conduction, according to,

$$-\nabla \cdot (k \nabla T) = 0 \tag{2}$$

where k is the thermal conductivity. The minus sign in Equation 2 is included for the consistency with the conservative form derivation $\nabla \cdot \vec{q} = 0$, $\vec{q} = -k \nabla T$.

The heat flux is zero on the symmetries and on the top portion of the domain, which is assumed to be insulated. Hence, on both of these boundaries, homogenous Neumann BC, $\vec{q} \cdot \vec{n} = 0$, is enforced.

V Properties

Table 1 lists the material and gas properties that will be used for this analysis. The subscripts c, h refer to the cold hot gas sides of the computational domain, respectively.

Table 1: *Material and Gas Properties*

Parameter	Description	Units	Baseline Value
$T_{ext,h}$	hot gas temperature	K	3,000
$T_{ext,c}$	cold gas temperature	K	300
$h_{ext,h}$	hot gas heat transfer coeff.	W/m^2.K	20,000
$h_{ext,c}$	cold gas heat transfer coeff.	W/m^2.K	2,000
k	steel thermal conductivity	W/m.K	40

VI Finite-Element Solver

The primary task is to write a finite-element solver for Equation 2 for the steady-state temperature T, of the steel, subject to the convective (Robin) boundary conditions in Equation 1. The solver should be a function that accepts arbitrary meshes, and sets up and solves the discrete linear system $\mathbf{AT} = \mathbf{F}$ for the nodal temperatures. The meshes are provided as part of the project.

The weak form corresponding to Equation 2 is obtained by multiplying the equation by test functions $\phi_i(\vec{x})$ and integrating by parts,

$$\int_\Omega \nabla\phi_i \cdot (k\nabla T)\, d\Omega + \int_{\partial\Omega} (k\nabla T)\cdot\vec{n}\, dl = 0 \tag{3}$$

Substituting Equation 1 for $\vec{q} = -k\nabla T$ on all of the boundaries yields,

$$\int_\Omega \nabla\phi_i \cdot (k\nabla T)\, d\Omega + \int_{\partial\Omega_{hc}} \phi_i h_{ext}(T - T_{ext})\, dl = 0 \tag{4}$$

where $\partial\Omega_{hc}$ is the portion of the domain boundary on the hot/cold gas sides. Note that on the other boundary (top and symmetry), $\vec{q}\cdot\vec{n} = 0$, so these do not contribute to the weak form. Moving the constant term to the right-hand side and substituting $T(\vec{x}) = \sum_j \phi_j(\vec{x})T_j$ yields,

$$\sum_j \left[\int_\Omega k\nabla\phi_i\cdot\nabla\phi_j\, d\Omega\right] T_j + \sum_j \left[\int_{\partial\Omega_{hc}} h_{ext}\phi_i\phi_j\, dl\right] T_j = \int_{\partial\Omega_{hc}} h_{ext}\phi_i T_{ext} \tag{5}$$

The left-hand side of this equation gives formulas for the i, j entry of the stiffness matrix, $\mathbf{A}$, whereas the right-hand sides gives the i^{th} entry of the right-hand side vector $\mathbf{F}$. Specifically,

$$A_{ij} = \int_\Omega k\nabla\phi_i\cdot\nabla\phi_j + \int_{\partial\Omega_{hc}} h_{ext}\phi_i\phi_j\, dl$$
$$F_i = \int_{\partial\Omega_{hc}} h_{ext}\phi_i T_{ext}$$

Note, that $\mathbf{A}$ and $\mathbf{F}$ should be assembled according to these formulas by looping over elements and boundary edges.

Output

The output of interest is the integrated heat flux, Q, from the hot-gas side to the material, which should be the same as the heat flux from the material to the cold-gas side, as the heat flux through the other boundaries is zero. The output is calculated through both boundaries (as a means of redundant checking),

$$Q = \int_{\partial\Omega_h} h_{ext,h}(T_{ext,h} - T)\, dl = \int_{\partial\Omega_c} h_{ext,c}(T - T_{ext,c})\, dl \tag{6}$$

Computational Domains

Three computational domains will be considered: baseline, hot-notched, and cold-notched.

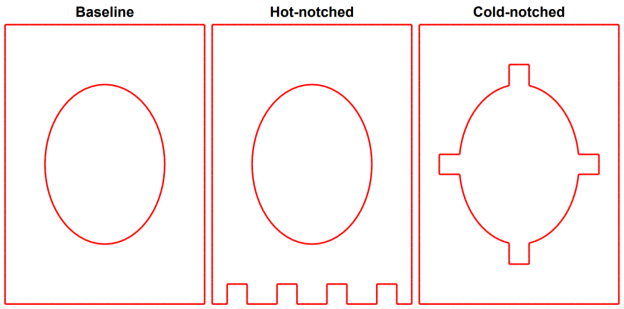

Figure 2: *Three computational domains*

The purpose of the notches in the latter two domains is to increase the heat transfer between the material and the hot/cold gases, respectively. One task will be to determine whether notches on the cold side are more effective than notches on the hot side.

Meshes

Eight computational meshes are provided with this project. These consist of: (1) four refinements of the baseline domain (meshes 0-3); (2) two refinements of the hot-notched domain (meshes 4,5); and (3) two refinements of the cold-notched domains (meshes 6,7). Figure 3 shows the coarsest mesh on each domain.

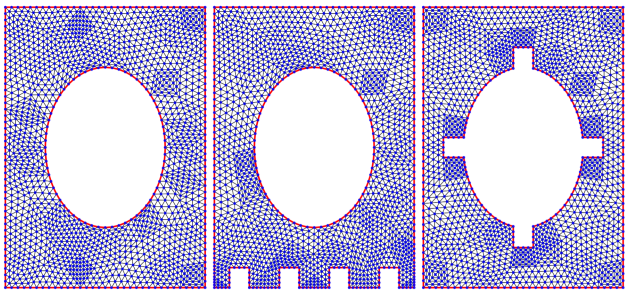

Figure 3: *Meshes (M2,M4,M6) around the three computational domains*

The meshes are provided in plain-text format, as three files for each mesh. These files consist of: `M#-V.txt`, which contains the (x, y) coordinates of each node, `M#-E.txt`, which contains the node numbers (1-based) of each triangle in the mesh, and `M#-B.txt`, which lists all of the boundary edges in the form of node pairs (1-based). Note that no boundary condition information is given with the boundary edges. The appropriate boundary condition (hot, cold, symmetry, top) should be determined based on the node coordinates.

1 Solving of Finite-Element System

The following section details the process of writing a computer program in MATLAB that sets up and solves the finite-element system for this problem using a given input mesh. The code structure will be explained with emphasis on the assembly of the stiffness matrix and how the boundary conditions were imposed.

The function that inputs a given mesh to setup and solve and the finite-element system was created using MATLAB. The function has been named `FiniteElement(M)`, where M is the mesh number specified through function output. The function uses this mesh number to input the nodes, elements, and boundaries contained in the given `.txt` files using the built-in function `dlmread()`. The initial values given in the properties table are defined in the function, along with the determination of the number of nodes, elements, and boundaries using the `length()` function.

Once the **A** matrix and **F** vector are preallocated, the function loops from 1 to the number of nodes, defined as k, using the given finite-element method.

1. The loop begins by defining a vector **g** that holds the 3 global nodes of element k
2. The **g** vector is then used in the constructing of a 3×3 matrix named **X** which houses the node coordinates
3. The next step is define the Jacobi matrix, **J** using the contents of the **X** matrix
4. The area used in the calculation of the local stiffness matrix is defined with the following equation,

$$\text{Area} = \frac{1}{2}\det(\mathbf{J})$$

5. To transform the area into the local stiffness matrix, ϕ_x is required and is calculated using the equation,

$$\phi_x = \phi_{xi}(\mathbf{J})^{-1}$$

   where ϕ_{xi} gives the relationship between ϕ, ξ, and η

$$\phi_{xi} = \begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

6. The local stiffness matrix is then assembled using,

$$\mathbf{A}_{local} = -k\phi_{xi}(\phi_x)^T \text{Area}$$

7. The local stiffness matrix is then added to the global **A** matrix to complete the loop.

$$\mathbf{A} = \mathbf{A} + \mathbf{A}_{local}$$

8. Once all nodes have been looped over, the boundaries are checked as detailed in the subsection below.

.

A snippet of the code which follows the above process is displayed below for comparison:

```
%Initial Values
k_mat = 40;
h_cold = 2000; h_hot = 20000;
t_cold = 300; t_hot = 3000;

%Number of Nodes
N_v = length(V);
%Number of Elements
N_e = length(E);
%Number of Boundaries
N_b = length(B);

%Preallocate
A = sparse(N_v, N_v);
F = zeros(N_v,1);

%Loop Over All Nodes
for k = 1:N_e
    g = E(k,:);
    x = V(g,:);
    %Jacobi Matrix
    J = [x(2,1) - x(1,1), x(3,1) - x(1,1);...
         x(2,2) - x(1,2), x(3,2) - x(1,2)];
    Area = .5 * det(J);
    %Phi x
    phi_xi = [-1 -1; 1 0; 0 1];
    phi_x = phi_xi / J;
    %Local Stiffness Matrix
    A_local = k_mat .* phi_x * phi_x' .* Area;
    %Sum to A Stiffness Matrix
    A(g,g) = A(g,g) + A_local;
end
```

Algorithm 1: *Code Snippet of Stiffness Matrix Assembly for Finite-Element Solver*

1.1 Boundary Conditions in Finite-Element System

Shown in Figure 4 is the boundary edge locations checked in the `FiniteElement(M)` through an `if` and `elseif` logic check. The given heat equations discussed in the problem description for the boundary conditions are imposed to the stiffness matrix **A** and **F** vector if the logic comes back true for the given node locations.

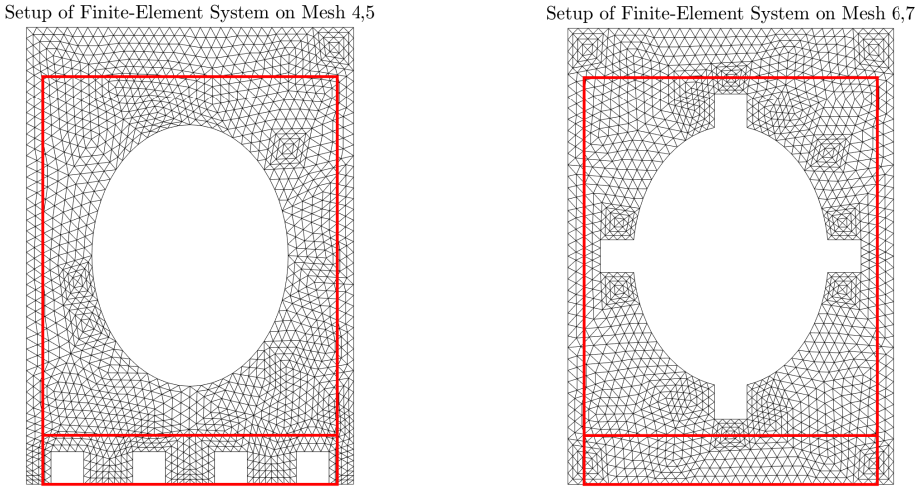

Figure 4: *Visual description of bound checking used for interior and bottom edges on the hot-notched and cold-notched domains, i.e. meshes 4 through 7.*

The boundary condition equations and logic check process is shown in the below code snippet, where the x and y locations in the `if/elseif` statements match the above figure to show the process of the code.

```
%Bottom Boundary Check
        if (abs(n1(2) + 0.006) < e && abs(n2(2) + 0.006) < e)  || ...
                (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) > -0.0045 && n2(1) < 0.0045) ...
                && (n1(2) > -0.004 && n2(2) < -0.004)
            A(g,g) = A(g,g) + h_hot * delta_l .* quad;
            F(g,1) = F(g,1) + h_hot * t_hot * delta_l/2;

        %Interior Boundary Check
        elseif (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) > -0.0045 && n2(1) < 0.0045) ...
                && (n1(2) > -0.0045 && n1(2) < 0.0065) && (n2(2) > -0.0045 && n2(2) < 0.0065)

            A(g,g) = A(g,g) + h_cold * delta_l .* quad;
            F(g,1) = F(g,1) + h_cold * t_cold * delta_l/2;
        end
```

Algorithm 2: *Code Snippet of Boundary Check for Finite-Element Solver*

The final step of the function is to calculate the temperature vector by solving the following equation,

$$\mathbf{T} = \mathbf{F}' \left(\mathbf{A}^{-1} \right) \tag{7}$$

Note: the complete finite-element solver function can be found in the appendix located at the end of the report.

2 Verification of Finite-Element Solver

In order to check if the code in the previous section is working properly a verification test will be performed on a two-element, four-node mesh, as shown in Figure 5. The `V`, `E`, and `B` files and printouts for this mesh will be included in the report. The boundary conditions are of the same type used in the nozzles meshes, so the code will not need to be changed to run this simplified domain. The test domain is advantageous in that it represents a one-dimensional heat transfer problem that can be solved by hand. Since the solution is linear, the linear finite element solution will be exact, to machine precision. A comparison can then be made of the node temperatures to 1D heat-transfer analysis that is done analytically.

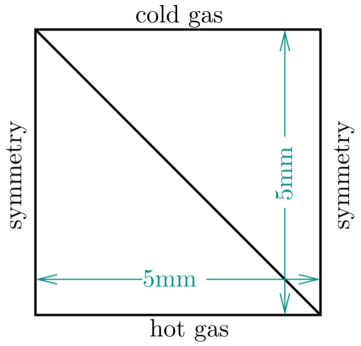

Figure 5: *Verification domain.*

The mesh of the above figure was made using three different `.txt` files for `V`, `E`, and `B`. Each file is named `Verification_x.txt`, with `x` denoting the different mesh files and is included with the report. The center of the square domain was used as the origin of (0,0). The files were created as follows:

- Node Locations File, `Verification_V.txt`:

```
-0.0025  -0.0025
 0.0025  -0.0025
 0.0025   0.0025
-0.0025   0.0025
```

- Node Elements File, `Verification_E.txt`:

```
1  2  4
2  3  4
```

- Node Boundaries File, `Verification_B.txt`:

```
1  2
2  3
1  4
3  4
```

The following test domain was then able to be inputted into the previously discussed finite-element solver to generate a temperature gradient across the simple 5 mm by 5 mm domain. The only change made to the previous solver was the inclusion of modified boundary conditions, since the bottom and top boundary are known. The following figure shows the boundaries used for the domain highlighted in red, as well as showing the input mesh elements and node locations.

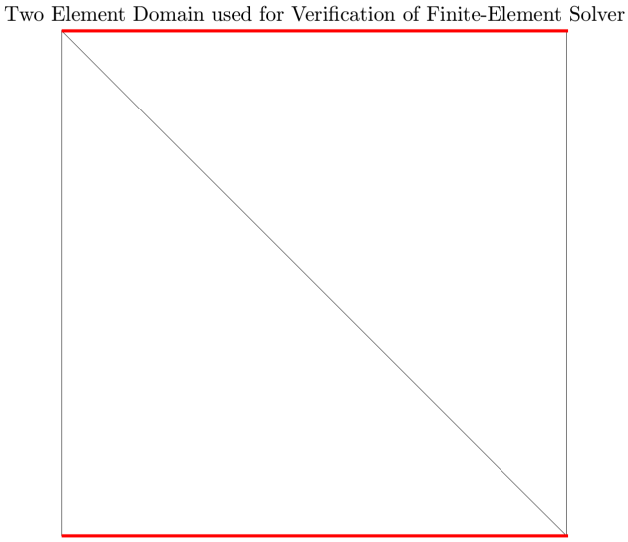

Figure 6: *Visual description of bound checking used for top and bottom edges on the verification domain.*

The finite-element solver was then ran on this domain to produce Figure 7, which represents a 1D heat transfer problem with a linear solution. As shown, the temperature does indeed decrease across the flat plate, as expected. Using the finite-element solver, the temperature range is shown as approximately 2800 K on the bottom of the plate, and approximately 2300 K on the top of the plate, with respect to machine precision.

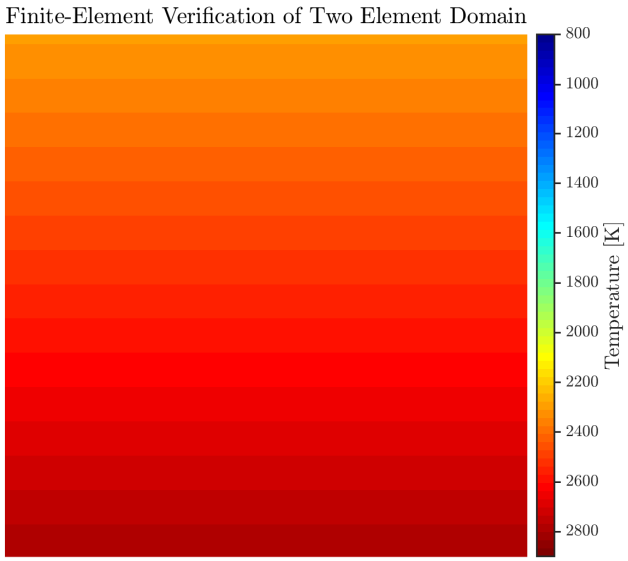

Figure 7: *Results of finite-element method on the verification domain.*

Due to the linear nature of the heat equation in a single dimension, the equation can also be solved analytically in order to compare to the simulated results. that can be solved by hand. Since the heat equation is given by,

$$-\nabla \cdot (k\nabla T) = 0$$

which can be rewritten explicitly as a description of a conducting plate with convective heat transfer in the one-dimensional domain,

$$\dot{q} = h(T_w) - T_\infty)$$

In the case of a heat transfer from the hot and cold sides of the plate,

$$\dot{q}_{hot} = h(T_{hot,wall} - T_{ext,h})$$
$$\dot{q}_{cold} = h(T_{cold,wall} - T_{ext,c})$$

The heat transfer across the plate is then given by the following equation where L is the length of the plate,

$$\dot{q} = \frac{k}{L}(T_{cold,wall} - T_{hot,wall})$$

Putting these three equations together will yield a relationship between the heat transfer and the temperature change, ΔT,

$$\begin{aligned}\Delta T &= T_{ext,c} - T_{ext,h} \\ &= (T_{ext,c} - T_{cold,wall}) + (T_{cold,wall} - T_{hot,wall}) + (T_{wall,h} - T_{ext,h}) \\ &= \dot{q}\left(\frac{1}{h_{ext,h}} + \frac{L}{k} + \frac{1}{h_{ext,c}}\right)\end{aligned}$$

Therefore, by simplifying the above expression for the wall temperatures for the hot and cold sides, the temperatures can be given and solved for as,

$$T_{wall,cold} = T_{ext,c} - \frac{T_{ext,c} - T_{ext,h}}{\frac{h_{ext,c}}{h_{ext,h}} + \frac{Lh_{ext,c}}{k} + 1} = 2300 \text{ K}$$
$$T_{wall,hot} = T_{ext,h} - \frac{T_{ext,h} - T_{ext,c}}{\frac{h_{ext,h}}{h_{ext,c}} + \frac{Lh_{ext,h}}{k} + 1} = 2800 \text{ K}$$

Plotting this distribution as done before with the finite-element solver, yields a color gradient that is extremely similar to that of the computational solver.

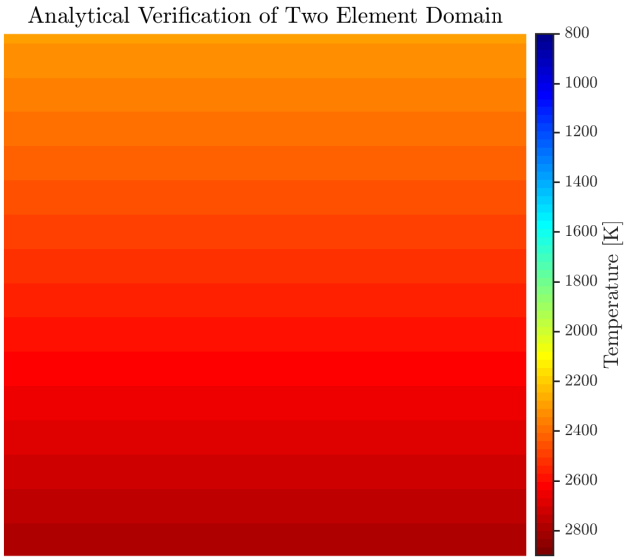

Figure 8: *Results of analytical solution on the verification domain.*

Numerically comparing the solution of the finite-element solver and the analytical solution shows error between the two methods is very small. The calculation of this error is shown below for the four nodes, from 1 to 4 in counterclockwise order,

$$\text{Error of Nodes } \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 1.0 \times 10^{-11} \times \begin{bmatrix} 0.045474735088646 & 0.045474735088646 \\ 0.090949470177293 & 0.136424205265939 \end{bmatrix}$$

Thus, the two methods produce almost identical results for the linear one-dimensional heat equation on the square metal plate.

3 Temperature Fields of the Three Domains

The following section will feature simulations of the finest meshes for all three domains (i.e., meshes 3, 5, 7) with corresponding plots for the temperature fields. The plots will lead a discussion of the impact of the notches on the temperature distribution in the steel.

The first simulation featured the baseline domain with the finest mesh. The results of the simulation are shown in Figure 9. As seen, the temperature decreases slowly from the bottom of the geometry to the top, moving from a temperature of approximately 2700 K to 1200 K across the domain.

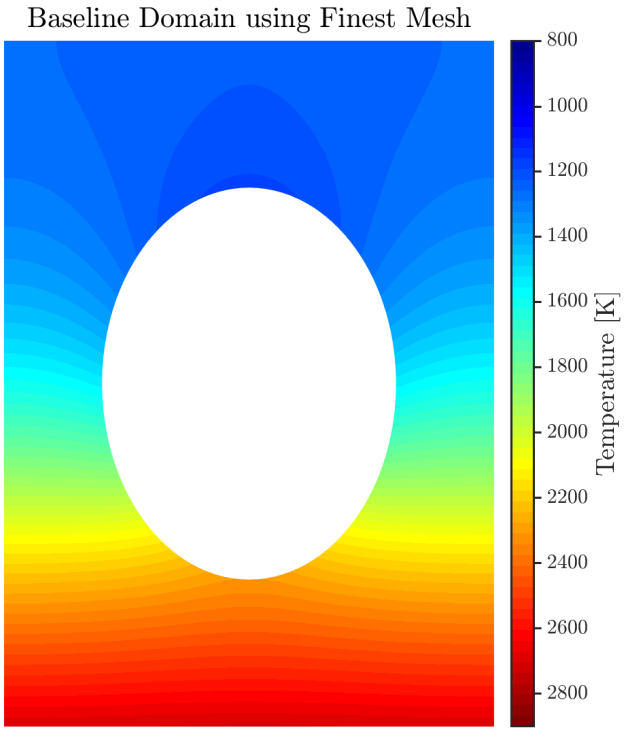

Figure 9: *Plot of the temperature field using the finest mesh for the baseline domain.*

The second simulation featured the hot-notched domain with the finest mesh. The results of the simulation are shown in Figure 10. As seen, the temperature decreases slowly from the bottom of the geometry to the top, moving from a temperature of approximately 2900 K to 1200 K across the domain. Therefore, the notches in the geometry of the bottom of the domain in the case of the hot-notched mesh is shown to lead to an overall decreased temperature distribution in the steel, as also proved through the colorbar on the side of the figure that is scaled the same as the baseline domain simulation results.

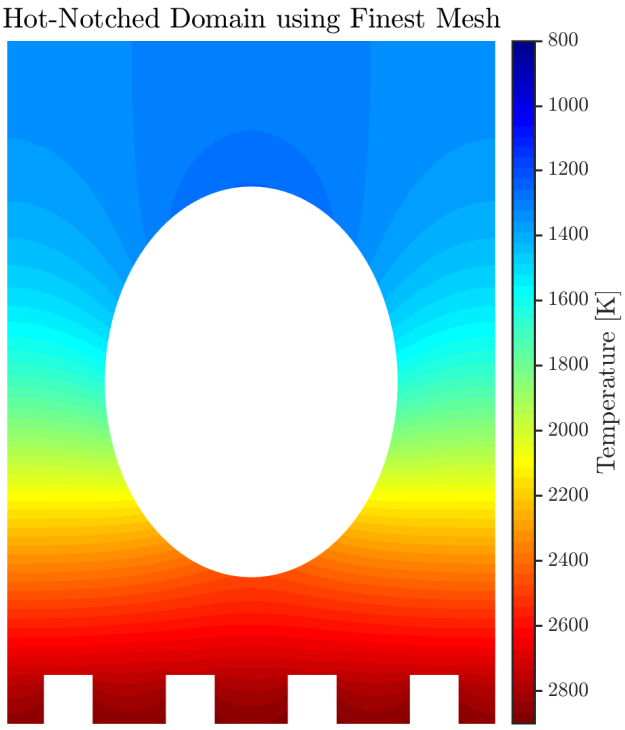

Figure 10: *Plot of the temperature field using the finest mesh for the hot-notched domain.*

The third and final simulation featured the cold-notched domain with the finest mesh. The results of the simulation are shown in Figure 11. As seen, the temperature decreases quicker than previously seen from the bottom of the geometry to the top, moving from a temperature of approximately 2650 K to 850 K across the domain. Therefore, the notches in the geometry of the oval domain in the case of the cold-notched mesh is shown to lead to the largest decreased temperature distribution in the steel, as also proved through the colorbar on the side of the figure that is scaled the same as the baseline and hot-notched domain simulation results.

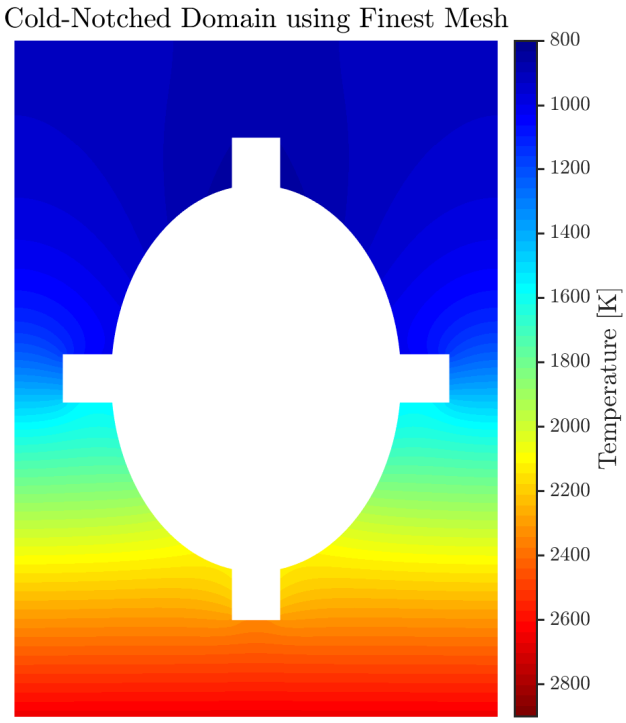

Figure 11: *Plot of the temperature field using the finest mesh for the cold-notched domain.*

As shown through these simulation results of the finest meshes for each domain, the impact of the notches on the temperature distribution in the steel is obvious. The cold-notched domain is the best choice for the nozzle wall cooling as it hinders the temperature distribution over the entire domain the best when compared to the baseline and hot-notched domains. However, it should be noted that the hot-notched domain did provide improvements in regards to heat dissipation when compared to the baseline domain. In all cases, the use of the simulation figures helps to support the superior choice for temperature distribution as all colorbars have been scaled to the same limits to ensure a quality display of data calculated in the simulations.

4 Integrated Heat Flux Calculation

The following section details the process of solving the integrated heat flux calculation for the given meshes, as well as on the verification mesh in Figure 5. The verification mesh will allow for the comparison of the cold/hot-side heat fluxes from the computations to that of the analytical results.

In order to calculate the heat flux of the hot and cold sides of the plate, a similar code structure was used in that of the finite-element solver. The function for the heat flux, named `heat_flux(T,M)` takes inputs of the temperature vector from the finite-element solver, as well as the desired mesh to run on. The initial given values are applied in the equation, followed by a determination of the number of boundaries used to loop over.

Within the loop that goes from 1 to the number of boundaries, k, the δl is defined as the norm between the two nodes that are being referenced at that particular loop. The temperature at these nodes is also indexed as T_i and T_j and averaged using the following equation,

$$T_{ij} = \frac{T_i + T_j}{2}$$

The same boundary conditions as used in the finite-element solver are used for the heat flux calculations. If the nodes are at a bottom boundary the following equation for the hot heat flux is applied,

$$\begin{aligned} Q_h &= \int_{\partial\Omega_h} h_{ext,h}(T_{ext,h} - T)\, dl \\ &= \sum_k h_{ext,h}(T_{ext,h} - T_{ij})(dl) \end{aligned}$$

If the nodes are at an interior boundary the following equation for the cold heat flux is applied,

$$\begin{aligned} Q_c &= \int_{\partial\Omega_c} h_{ext,c}(T_{ext,c} - T)\, dl \\ &= \sum_k h_{ext,c}(T_{ext,c} - T_{ij})(dl) \end{aligned}$$

The function was modified to accept input of the verification domain by changing the boundary conditions to apply the hot heat flux equation on the bottom of the domain and the cold heat flux equation on the top of the domain.

Giving the input of the verification mesh produces an approximate to machine precision output of the hot/cold heat flux as the following,

$$\begin{aligned} Q_h &= 20{,}000 \text{ W/m}^2 \\ Q_c &= 20{,}000 \text{ W/m}^2 \end{aligned}$$

To solve the heat flux of the domain analytically the heat flux equation can be rewritten using,

$$\begin{aligned} Q &= \int_{\partial\Omega_h} h_{ext,h}(T_{ext,h} - T)\, dl = \int_{\partial\Omega_c} h_{ext,c}(T - T_{ext,c})\, dl \\ Q_h &= h_{ext,h}\,(T_{ext,h} - T_{wall,hot})\, L \\ Q_c &= h_{ext,c}\,(T_{wall,cold} - T_{ext,c})\, L \end{aligned}$$

Now, Q_h and Q_c can be solved for analytically,

$$\begin{aligned} Q_h &= 20{,}000 \text{ W/m}^2 \\ Q_c &= 20{,}000 \text{ W/m}^2 \end{aligned}$$

Numerically comparing the solution of the computational solver for heat flux and the analytical solution shows error between the two methods is very small. The calculation of this error is shown below for the two values,

$$\text{Error of Heat Flux} \begin{bmatrix} Q_h \\ Q_c \end{bmatrix} = 1.0 \times 10^{-10} \times \begin{bmatrix} 0.072759576141834 \\ 0.472937244921923 \end{bmatrix}$$

Thus, the two methods produce almost identical results for the heat flux through the verification domain using the linear one-dimensional heat equation on the square metal plate.

5 Convergence Study of Heat Flux Output

Using the baseline domain, a convergence study can be performed of the heat flux output. To identify the heat flux error the finest mesh result of the baseline domain will be used as the exact value. Using the created function `heat_flux(T,M)` from the previous section. The heat flux output, Q, can be computed for all mesh types in the baseline domain.

Using the finest mesh result (i.e. mesh 3), the exact heat flux outputs are computed as,

$$Q_{exact,h} = 59253.975 \text{ W/m}^2$$
$$Q_{exact,c} = 59253.975 \text{ W/m}^2$$

Inputting the other meshes into the function will produce values for the hot and cold heat flux that can be used to approximate the convergence rate of the heat flux output, as well as create a plot showing the absolute value of the error versus the mesh spacing.

The mesh spacing, h, is defined as,

$$h = 1/\sqrt{N_e}$$

where N_e is the number of elements in the mesh.

The absolute error is defined using the heat flux output of each mesh size of the baseline domain, and the exact solution as determined by the finest mesh,

$$\text{Error} = |Q_{exact} - Q|$$

Plotting error from meshes 1 through 3 against the finest mesh as a function of the mesh size yielded the following convergence study plot.

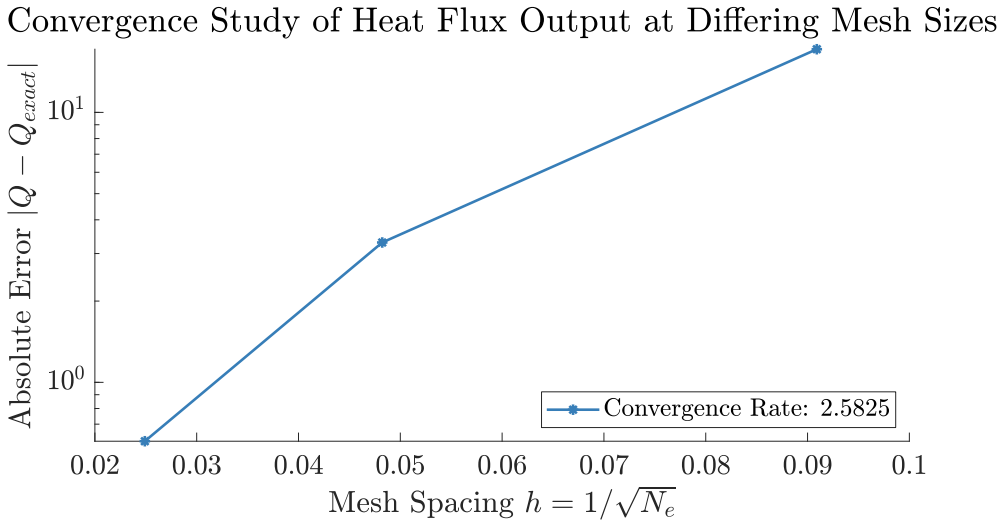

Figure 12: *Convergence study of the heat flux output using the finest mesh result as the exact value.*

As shown on the figure, the convergence rate of the heat flux output for the baseline domain is approximately 2.5825. It is also important to note that the error decreases as the mesh spacing gets smaller, which is the expected behavior for a scenario such as this and is indicative of a convergent solution.

6 Heat Flux Outputs of the Domains

The heat flux outputs for the finest meshes of the baseline, hot-notched, and cold-notched domains were computed using the previously mentioned function `heat_flux(T,M)`. The effect of the notches on the heat flux will be discussed in this section. The computations are shown below for the hot heat fluxes across the domains.

$$Q_{h,baseline} = 59,253.98 \text{ W/m}^2$$
$$Q_{h,hot-notched} = 64,190.71 \text{ W/m}^2$$
$$Q_{h,cold-notched} = 68,219.49 \text{ W/m}^2$$

The below figure shows a visual representation of the hot/cold heat flux using the different geometric cutouts. The displayed data helps to show the comparison in a visual format instead of just simple values; however, it also represents the trend of the increase of heat flux magnitude from baseline to hot-notched to cold-notched domains.

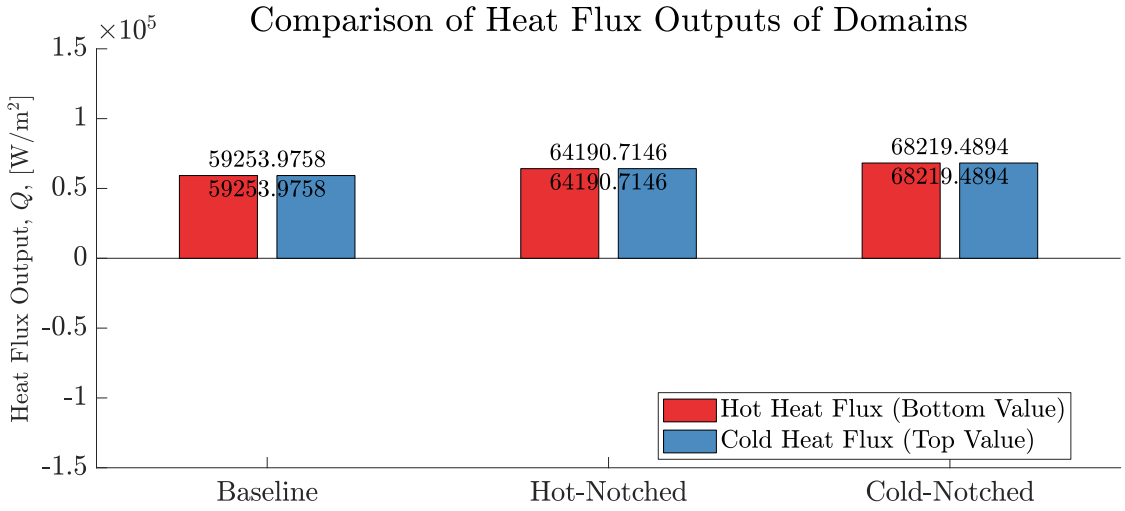

Figure 13: *Comparison of the heat flux outputs of the three given domains at the finest meshes available for them.*

The notches in the hot-notched and cold-notched domains greatly impact the heat flux across the geometry. The baseline domain, with no notches, features the smallest in magnitude heat flux. This means the baseline domain is absorbing the most heat and not dissipating it well in comparison to the other domains. The hot-notched domain has a greater heat flux magnitude than the baseline; indicative of better heat dissipation. **However, the cold-notched domain has the greatest magnitude of heat flux and is the best choice in terms of heat depletion in the steel plate.** This result is backed up by the idea that the more surface boundary area exposed to outside fluid flow, the more heat will be able to dissipate in the material.

The heat flux computations shown above for the three different domains show the impact of the notches on the heat flux for the real world example of regenerative cooling. This maximum heat flux magnitude is vital to known in order to ensure steady-state operation is possible within rocket combustion chambers and nozzles.

7 Conclusion

The following document details the process used to solve the heat transfer equation using a finite-element solver on the three different domains in order to simulate heat transfer in a regeneratively cooled nozzle. The three domains included a baseline, hot-notched, and cold-notched geometries. The successful implementation of the finite-element solver was completed in MATLAB and allowed for the plotting of temperature gradients across the three different analyzed rectangular domains. An analytical solution was created to coincide with a two-element verification domain in order to ensure the solutions generated by the finite-element solver were correct.

The heat flux across the domains was also computed in order to study the effects of mesh size and geometry on the heat flux values. Included in the heat flux analysis was a convergence study to determine the convergence rate of the baseline domain with respect to the finite-element solver and heat flux calculation. The heat flux computations also extended out to the three different domains in an effort to discuss the effect of the notches on the heat flux for the real world example of regenerative cooling to make steady-state operation possible within rocket combustion chambers and nozzles.

A Appendix

The MATLAB code ran to generate the plots and numerical results in this document is attached below. Some function calls have been commented out in order to save time when running the code as locally stored matrices of values resulted in much better performance run time.

A.1 List of Algorithms

A.2 MATLAB Script Implementation

```
%Drake Rundell
%AE 423 - Project 3
%Due: 4/20/2020

clear; clc; close all;
set(groot,'DefaultTextInterpreter','LaTeX');
set(groot,'DefaultAxesTickLabelInterpreter','LaTeX');
set(groot,'DefaultLegendInterpreter','LaTeX');
set(groot,'defaultLegendFontSize',8);
set(groot,'defaultAxesFontSize',16);
set(groot,'defaultLineLineWidth',1.5);

%% Task 1
%Use Mesh 4 as Example
[V_4,E_4,T_4] = FiniteElement(4);
%Use Mesh 6 as Example
[V_6,E_6,T_6] = FiniteElement(6);

figure
title('Setup of Finite-Element System on Mesh 6,7','FontSize',20);
patch('Vertices', V_6, 'Faces',E_6,'FaceVertexCData',zeros(1648,1),'
    Facecolor','white','EdgeColor','black');
patch([0.0045 -0.0045 -0.0045 0.0045], [0.0065 0.0065 -0.0045 -0.0045],
    [0 0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
patch([0.0045 -0.0045 -0.0045 0.0045], [-0.006 -0.006 -0.0045 -0.0045],
    [0 0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
axis equal; axis tight; axis off;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/ExampleHowItWorks1.eps -native

figure
title('Setup of Finite-Element System on Mesh 4,5','FontSize',20);
patch('Vertices', V_4, 'Faces',E_4,'FaceVertexCData',zeros(max(size(V_4))
    ,1),'Facecolor','white','EdgeColor','black');
patch([0.0045 -0.0045 -0.0045 0.0045], [0.0065 0.0065 -0.0045 -0.0045],
    [0 0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
patch([0.0045 -0.0045 -0.0045 0.0045], [-0.006 -0.006 -0.0045 -0.0045],
    [0 0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
axis equal; axis tight; axis off;
```

```
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/ExampleHowItWorks2.eps -native

%% Task 2
[V_v,E_v,T_v] = FiniteElement(-1);

figure
title('Two Element Domain used for Verification of Finite-Element Solver'
    ,'FontSize',20);
patch('Vertices', V_v, 'Faces',E_v,'FaceVertexCData',zeros(max(size(V_v))
    ,1),'Facecolor','white','EdgeColor','black');
patch([0.0025 -0.0025 -0.0025 0.0025], [-0.0025 -0.0025 -0.0025 -0.0025],
     [0 0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
patch([0.0025 -0.0025 -0.0025 0.0025], [0.0025 0.0025 0.0025 0.0025], [0
    0 0 0],'FaceColor','none','EdgeColor','red','Linewidth',3)
axis equal; axis tight; axis off;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/VerificationMesh_setup.eps -native

figure
title('Finite-Element Verification of Two Element Domain','FontSize',20);
patch('Vertices', V_v, 'Faces',E_v,'FaceVertexCData',T_v','Facecolor','
    interp','EdgeColor','none');
axis equal; axis tight; axis off;
h = colorbar;
caxis([800 2900])
h.Label.Interpreter = 'latex';
h.Label.String = 'Temperature [K]';
h.Label.FontSize = 18;
set(h, 'YDir', 'reverse','TickLabelInterpreter','LaTeX','TickDirection','
    out');
h.LineWidth = 1.5;
axis equal; axis tight; axis off;
colormap jet;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/VerificationMesh.eps -native

%Initial Values
k= 40;
h_cold = 2000; h_hot = 20000;
t_cold = 300; t_hot = 3000;
L = 0.005;
y = [0 0 0.005 0.005];

T_wall_cold = t_cold - (t_cold-t_hot) / ((h_cold/h_hot) + L * (h_cold/k)
    + 1);
T_wall_hot = t_hot - (t_hot-t_cold) / ((h_hot/h_cold) + L * (h_hot/k) +
    1);

T = [T_wall_hot T_wall_hot T_wall_cold T_wall_cold];
figure
title('Analytical Verification of Two Element Domain','FontSize',20);
patch('Vertices', V_v, 'Faces',E_v,'FaceVertexCData',T','Facecolor','
    interp','EdgeColor','none');
axis equal; axis tight; axis off;
h = colorbar;
caxis([800 2900])
h.Label.Interpreter = 'latex';
h.Label.String = 'Temperature [K]';
```

```
h.Label.FontSize = 18;
set(h, 'YDir', 'reverse','TickLabelInterpreter','LaTeX','TickDirection','out');
h.LineWidth = 1.5;
axis equal; axis tight; axis off;
colormap jet;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/VerificationMesh_anal.eps -native

%% Task 3
%Mesh 3, 5, 7
[V_3,E_3,T_3] = FiniteElement(3);
[V_5,E_5,T_5] = FiniteElement(5);
[V_7,E_7,T_7] = FiniteElement(7);

figure
title('Baseline Domain using Finest Mesh','FontSize',20);
patch('Vertices', V_3, 'Faces',E_3,'FaceVertexCData',T_3','Facecolor','interp','EdgeColor','none');
h = colorbar;
caxis([800 2900])
h.Label.Interpreter = 'latex';
h.Label.String = 'Temperature [K]';
h.Label.FontSize = 18;
set(h, 'YDir', 'reverse','TickLabelInterpreter','LaTeX','TickDirection','out');
h.LineWidth = 1.5;
axis equal; axis tight; axis off;
colormap jet;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/Mesh_fine_base.eps -native

figure
title('Hot-Notched Domain using Finest Mesh','FontSize',20);
patch('Vertices', V_5, 'Faces',E_5,'FaceVertexCData',T_5','Facecolor','interp','EdgeColor','none');
h = colorbar;
caxis([800 2900])
h.Label.Interpreter = 'latex';
h.Label.String = 'Temperature [K]';
h.Label.FontSize = 18;
set(h, 'YDir', 'reverse','TickLabelInterpreter','LaTeX','TickDirection','out');
h.LineWidth = 1.5;
axis equal; axis tight; axis off;
colormap jet;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/Mesh_fine_hot.eps -native

figure
title('Cold-Notched Domain using Finest Mesh','FontSize',20);
patch('Vertices', V_7, 'Faces',E_7,'FaceVertexCData',T_7','Facecolor','interp','EdgeColor','none');
h = colorbar;
caxis([800 2900])
h.Label.Interpreter = 'latex';
h.Label.String = 'Temperature [K]';
h.Label.FontSize = 18;
```

```
set(h, 'YDir', 'reverse','TickLabelInterpreter','LaTeX','TickDirection','
    out');
h.LineWidth = 1.5;
axis equal; axis tight; axis off;
colormap jet;
set(gcf, 'Color', 'w', 'Position', [0 0 850 850]);
export_fig Figures/Mesh_fine_cold.eps -native

%% Task 4
[q_h_v, q_c_v] = heat_flux(T_v,-1);

%Analytical Solution
Q_c_a = h_cold * (T_wall_cold - t_cold) * 0.005;
Q_h_a = h_hot * (t_hot - T_wall_hot) * 0.005;

%Error Calculation
error_HF_verify = abs([Q_h_a Q_c_a] - [q_h_v q_c_v]);

%% Task 5
%All meshes of baseline domain
[V_3,E_3,T_3] = FiniteElement(3);
[V_2,E_2,T_2] = FiniteElement(2);
[V_1,E_1,T_1] = FiniteElement(1);
[V_0,E_0,T_0] = FiniteElement(0);

%Values of heat flux at all baseline domain meshes
[q_h_3, q_c_3] = heat_flux(T_3,3);
[q_h_2, q_c_2] = heat_flux(T_2,2);
[q_h_1, q_c_1] = heat_flux(T_1,1);
[q_h_0, q_c_0] = heat_flux(T_0,0);

%Calculate Error
error_HF_baseline_h = [abs(q_h_3 - q_h_2) abs(q_h_3 - q_h_1) abs(q_h_3 -
    q_h_0)];
N = 1 ./ sqrt([length(V_2) length(V_1) length(V_0)]);

%Calculate slopes of all Stencils with Direct Solve
slope = log10(error_HF_baseline_h(3)/error_HF_baseline_h(1))/log10(N(3)/N
    (1));

figure
axes('NextPlot','replacechildren','ColorOrder',linspecer(1));
semilogy(N, error_HF_baseline_h,'-*');
xlabel('Mesh Spacing $h = 1/\sqrt{N_e}$','interpreter','latex');
ylabel('Absolute Error $|Q-Q_{exact}|$','interpreter','latex');
title('Convergence Study of Heat Flux Output at Differing Mesh Sizes','
    FontSize',20,'Interpreter','LaTeX');
legend({['Convergence Rate: ', num2str(slope)]},'Location', 'Southeast','
    Interpreter','LaTeX');
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
export_fig Figures/ConvergenceStudy_task5.eps -native

%% Task 6
%Meshes of finest domains
[V_3,E_3,T_3] = FiniteElement(3);
[V_5,E_5,T_5] = FiniteElement(5);
[V_7,E_7,T_7] = FiniteElement(7);

%Values of heat flux at finest meshes
```

```
[q_h_3, q_c_3] = heat_flux(T_3,3);
[q_h_5, q_c_5] = heat_flux(T_5,5);
[q_h_7, q_c_7] = heat_flux(T_7,7);

%Comparison of Heat Flux Outputs
mesh_num = [3 5 7];
HF_fine_h = [q_h_3 q_h_5 q_h_7];
HF_fine_c = [q_c_3 q_c_5 q_c_7];
HF_fine = [HF_fine_h' HF_fine_c'];

X = categorical({'Baseline','Hot-Notched','Cold-Notched'});
X = reordercats(X,{'Baseline','Hot-Notched','Cold-Notched'});
Y = HF_fine(:,2)';
Z = HF_fine(:,1)';

figure
axes('NextPlot','replacechildren','ColorOrder',linspecer(6));
b = bar(X,HF_fine);
ylim([-1.5 1.5] .* 1e5);
text(1:length(Y),Y,num2str(Y'),'vert','bottom','horiz','center','fontsize'
    ,14,'Interpreter','LaTeX');
text(1:length(Z),Z,num2str(Z'),'vert','top','horiz','center','fontsize'
    ,14,'Interpreter','LaTeX');
ylabel('Heat Flux Output, $Q$, [W/m$^2$]','fontsize',14,'interpreter','
    latex');
title('Comparison of Heat Flux Outputs of Domains','FontSize',20,'
    Interpreter','LaTeX');
legend('Hot Heat Flux (Bottom Value)','Cold Heat Flux (Top Value)','
    Location', 'best','Interpreter','LaTeX');
set(gcf, 'Color', 'w', 'Position', [200 200 1000 400]);
export_fig Figures/HeatFluxComparison.eps -native
```

Algorithm 3: *MATLAB Script used to call implemented functions*

A.3 MATLAB Function Implementation

```
function [V,E,T] = FiniteElement(M)

    %Input mesh
    if (M == -1)
        V = dlmread('Meshes/Verification_V.txt');
        E = dlmread('Meshes/Verification_E.txt');
        B = dlmread('Meshes/Verification_B.txt');
    else
        V = dlmread(['Meshes/M',num2str(M),'-V.txt']);
        E = dlmread(['Meshes/M',num2str(M),'-E.txt']);
        B = dlmread(['Meshes/M',num2str(M),'-B.txt']);
    end

    %Initial Values
    k_mat = 40;
    h_cold = 2000; h_hot = 20000;
    t_cold = 300; t_hot = 3000;

    %Number of Nodes
    [N_v,~] = size(V);
    %Number of Elements
    [N_e,~] = size(E);
    %Number of Boundaries
    [N_b,~] = size(B);

    %Preallocate
    A = sparse(N_v, N_v);
    F = zeros(N_v,1);

    %Loop Over All Nodes
    for k = 1:N_e
        g = E(k,:);
        x = V(g,:);
        %Jacobi Matrix
        J = [x(2,1) - x(1,1), x(3,1) - x(1,1);...
             x(2,2) - x(1,2), x(3,2) - x(1,2)];
        Area = .5 * det(J);
        %Phi x
        phi_xi = [-1 -1; 1 0; 0 1];
        phi_x = phi_xi / J;
        %Local Stiffness Matrix
        A_local = k_mat .* phi_x * phi_x' .* Area;
        %Sum to A Stiffness Matrix
        A(g,g) = A(g,g) + A_local;
    end

    %Loop over and check boundaries
    e = 10e-10;
    for k = 1:N_b
        g = B(k,:);
        n1 = V(B(k,1),:);
        n2 = V(B(k,2),:);
        delta_l = norm(n2-n1);

        %Quadrature Weights
        quad = [1/3 1/6; 1/6 1/3];
```

```
57          if (M == -1)
58              %Bottom Boundary Check
59              if (n1(2) == -0.0025 && n2(2) == -0.0025)
60                  A(g,g) = A(g,g) + h_hot * delta_l .* quad;
61                  F(g,1) = F(g,1) + h_hot * t_hot * delta_l/2;
62
63              %Top Boundary Check
64              elseif (n1(2) == 0.0025 && n2(2) == 0.0025)
65                  A(g,g) = A(g,g) + h_cold * delta_l .* quad;
66                  F(g,1) = F(g,1) + h_cold * t_cold * delta_l/2;
67              end
68          else
69              %Bottom Boundary Check
70              if (abs(n1(2) + 0.006) < e && abs(n2(2) + 0.006) < e)  || ...
71                      (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) >
    -0.0045 && n2(1) < 0.0045) ...
72                      && (n1(2) < -0.004 && n2(2) < -0.004)
73                  A(g,g) = A(g,g) + h_hot * delta_l .* quad;
74                  F(g,1) = F(g,1) + h_hot * t_hot * delta_l/2;
75
76              %Interior Boundary Check
77              elseif (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) >
    -0.0045 && n2(1) < 0.0045) ...
78                      && (n1(2) > -0.0045 && n1(2) < 0.0065) && (n2(2) >
    -0.0045 && n2(2) < 0.0065)
79
80                  A(g,g) = A(g,g) + h_cold * delta_l .* quad;
81                  F(g,1) = F(g,1) + h_cold * t_cold * delta_l/2;
82              end
83          end
84      end
85
86      T = F' * inv(A);
87  end
```

Algorithm 4: *Function responsible for solving finite-element system*

```
function [q_c, q_h] = heat_flux(T,M)
    %Input mesh
    if (M == -1)
        V = dlmread('Meshes/Verification_V.txt');
        E = dlmread('Meshes/Verification_E.txt');
        B = dlmread('Meshes/Verification_B.txt');
    else
        V = dlmread(['Meshes/M',num2str(M),'-V.txt']);
        E = dlmread(['Meshes/M',num2str(M),'-E.txt']);
        B = dlmread(['Meshes/M',num2str(M),'-B.txt']);
    end
    %Initial Values
    h_cold = 2000; h_hot = 20000;
    t_cold = 300; t_hot = 3000;
    q_h = 0; q_c = 0;
    %Number of Boundaries
    [N_b,~] = size(B);
    %Loop over and check boundaries
    e = 10e-10;
    for k = 1:N_b
        n1 = V(B(k,1),:);
        n2 = V(B(k,2),:);

        delta_l = norm(n2-n1);

        T_i = T(B(k,1)); T_j = T(B(k,2));
        T_ij = (T_j + T_i) / 2;

        if (M == -1)
            %Bottom Boundary Check
            if (n1(2) == -0.0025 && n2(2) == -0.0025)
                q_h = q_h + (h_hot * (t_hot - T_ij) * delta_l);

            %Top Boundary Check
            elseif (n1(2) == 0.0025 && n2(2) == 0.0025)
                q_c = q_c + (h_cold * (T_ij - t_cold) * delta_l);
            end
        else
            %Bottom Boundary Check
            if (abs(n1(2) + 0.006) < e && abs(n2(2) + 0.006) < e)  || ...
                    (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) >
-0.0045 && n2(1) < 0.0045) ...
                    && (n1(2) < -0.004 && n2(2) < -0.004)
                    q_h = q_h + (h_hot * (t_hot - T_ij) * delta_l);
            %Interior Boundary Check
            elseif (n1(1) > -0.0045 && n1(1) < 0.0045) && (n2(1) >
-0.0045 && n2(1) < 0.0045) ...
                    && (n1(2) > -0.0045 && n1(2) < 0.0065) && (n2(2) >
-0.0045 && n2(2) < 0.0065)

                    q_c = q_c + (h_cold * (T_ij - t_cold) * delta_l);
            end
        end
    end
end
```

Algorithm 5: *Function responsible for solving heat flux through finite-element solver*