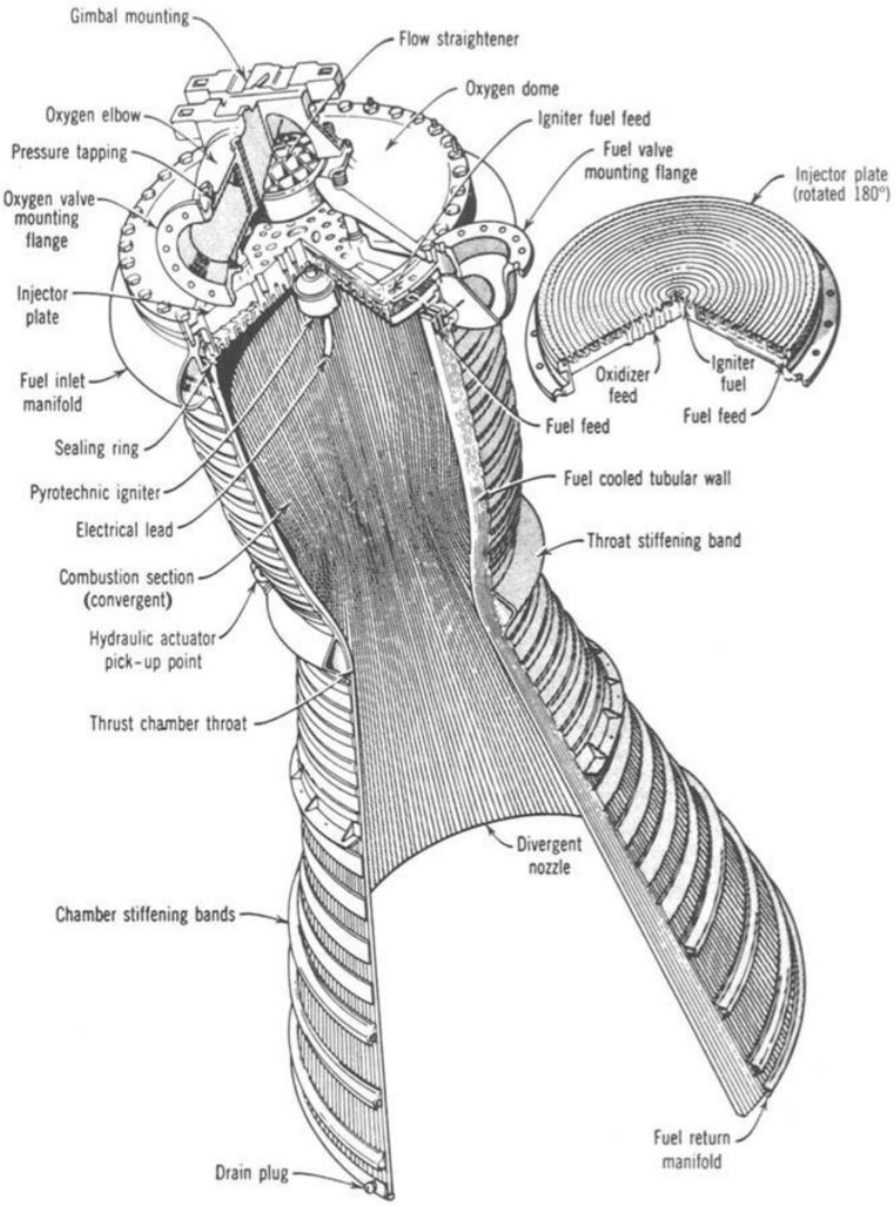
Project 3: Regenerative Nozzle Wall Cooling

Aerospace 423: Computational Methods in Aerospace Engineering
Undergraduate Aerospace Engineering
University of Michigan, Ann Arbor

By: Dan Card, dcard@umich.edu
Date: April 20, 2020



Rocket Propulsion Elements, 9th Ed.
Sutton, Biblarz

COLLEGE OF ENGINEERING
AEROSPACE ENGINEERING
UNIVERSITY OF MICHIGAN

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Overview

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.

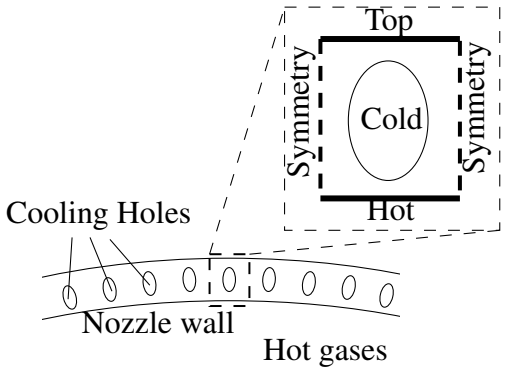


Figure 1: Cross-section of a regeneratively-cooled nozzle wall.

In this project, I will simulate heat transfer in a regeneratively-cooled nozzle wall 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 me to neglect curvature and analyze a rectangular domain. In particular, assuming the cooling passages are evenly-spaced, I can take as the analysis domain the dashed rectangular region shown in the figure.

1.2 Heat Transfer

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

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

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. I assume constant h_{ext} and T_{ext} for this analysis 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 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, enforce the homogeneous Neumann BC, $\vec{q} \cdot \vec{n} = 0$.

1.3 Properties

Table 1 lists the material and gas properties that I will use for this project. 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.	$W/m^2 \cdot K$	$20,000$
$h_{ext,c}$	Cold gas heat transfer coeff.	$W/m^2 \cdot K$	$2,000$
k	Steel thermal conductivity	$W/m \cdot K$	40

1.4 Finite-Element Solver

My primary task is to write a finite-element solver to solve Equation 2 for the steady-state temperature, T, of the steel, subject to the convective (Robin) boundary conditions in Equation 1. The solver will be a function that accepts arbitrary meshes, and sets up and solves the discrete linear system, $\underline{\underline{A}}\ \underline{T} = \underline{F}$ 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,

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

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

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

where $\partial\Omega_{hc}$ is the portion of the domain boundary on the hot/cold gas sides. Note that on the other boundaries (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

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

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

$$\underline{\underline{A}}_{ij} = \int_{\Omega} k \nabla \phi_i \cdot \nabla \phi_j + \int_{\partial \Omega_{hc}} h_{ext} \phi_i \phi_j dl,$$
$$\underline{F}_i = \int_{\partial \Omega_{hc}} h_{ext} \phi_i T_{ext}.$$

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

1.5 Output

The output of interest is the integrated heat flux, Q, from the hot-gas side to the material, which will 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. Calculate the output through both boundaries(as a means of redundant checking),

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

1.6 Computational Domains

I will consider three computational domains: baseline, hot-notched, and cold-notched, as shown in Figure 2.

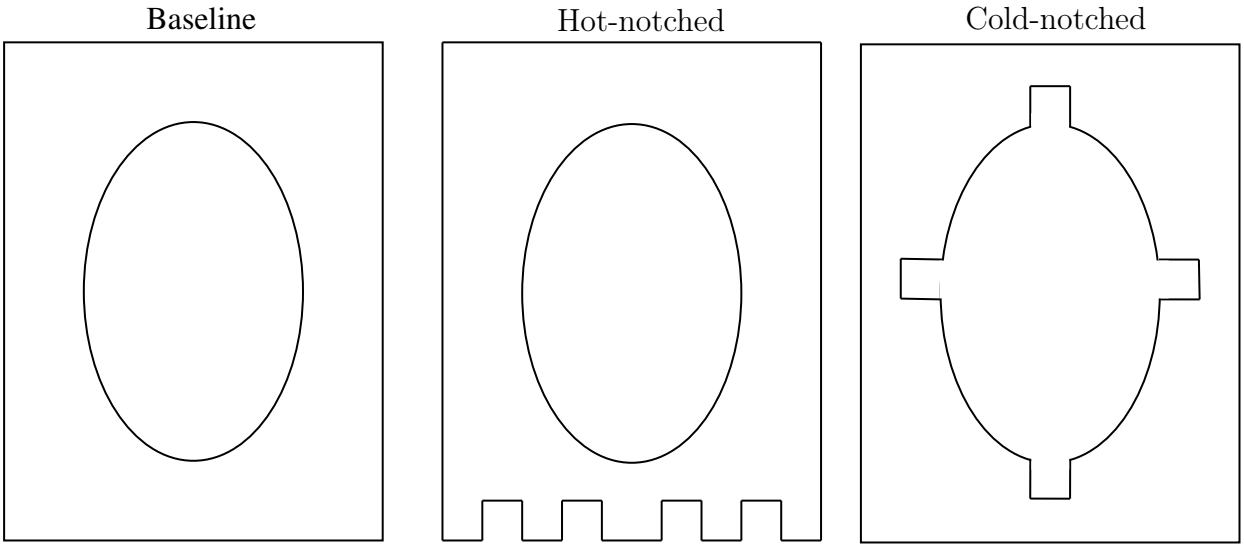


Figure 2: Three computational domains.

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 of the tasks will be to determine whether notches on the cold side are more effective than notches on the hot side.

1.7 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.

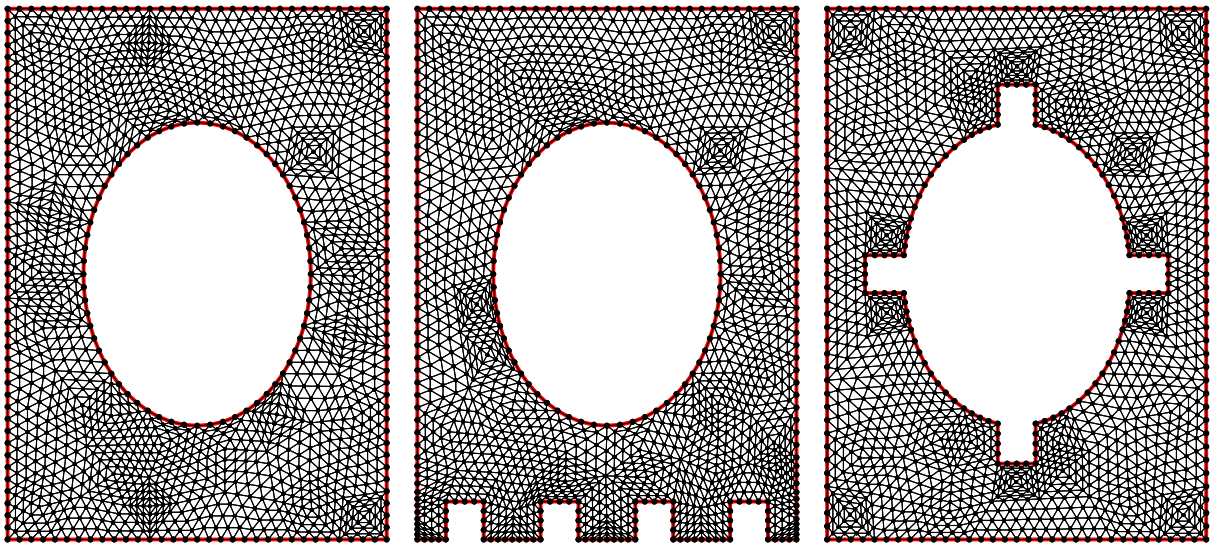


Figure 3: Meshes (M2,M4,M6) around the three computational domains.

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. I will determine the appropriate boundary condition (hot, cold, symmetry, top) based on the node coordinates.

2 Tasks and Deliverables

In this project I will implement a finite-element solver to approximate the temperature field around a nozzle's regenerative channel and verify its functionality. Also, I will perform an in-depth analysis of the effects of changing the configuration of the cooling channels.

2.1 Implementing a Finite-Element Solver

Firstly to solve this mesh for the Temperature field I will write a Matlab script that will calculate and approximate the solution. Implementing the finite element solver from a pseudo-code level is as follows:

1. Input what mesh to use # 0-8 (8 is the verification mesh) and account for mesh inputs that are outside of this range by requesting the user to input another suitable mesh.
2. Use Matlab's `dlmread` function to input mesh information (E, B, V) from the user requested mesh
3. Declare all variables defined in the project specification $T_{ext,h}, T_{ext,c}, \ldots$ etc.
4. Pre-allocate the $(N_{nodes} \times N_{nodes})$ `sparse` $\underline{\underline{A}}$ matrix and the $(N_{nodes} \times 1)$ $\underline{F}$ vector
5. Construct the global $\underline{\underline{A}}$ matrix by:
   (a) Iterating through all the elements in the given mesh in a `for` loop
   (b) In each iteration formulate g from the E matrix to get what nodes correlate to each triangle
   (c) From g pass into the V matrix to determine the node (x, y) locations of each node to be used for the Jacobian matrix shown below as:

$$\underline{\underline{J}} = \begin{bmatrix} x(2,1) - x(1,1) & x(3,1) - x(1,1) \\ x(2,2) - x(1,2) & x(3,2) - x(1,2) \end{bmatrix} \Bigg\} \text{ Jacobian Matrix}$$

   (d) Formulate the Jacobian matrix to transform the gradient from reference space to global space and arrive at the local stiffness matrix A_{ij} simply shown as,

$$\phi_x = \underbrace{\phi_{xi}}_{\begin{bmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}} \cdot \underline{\underline{J}}^{-1}$$

$$\underline{\underline{A}}^k = k\phi_x \cdot \phi_x^T \underline{\underline{J}}^{-1}$$

   (e) Add the local stiffness matrix to the global stiffness matrix $\underline{\underline{A}}$

$$\underline{\underline{A}}(g, g) = \underline{\underline{A}}(g, g) + \underline{\underline{A}}^k$$

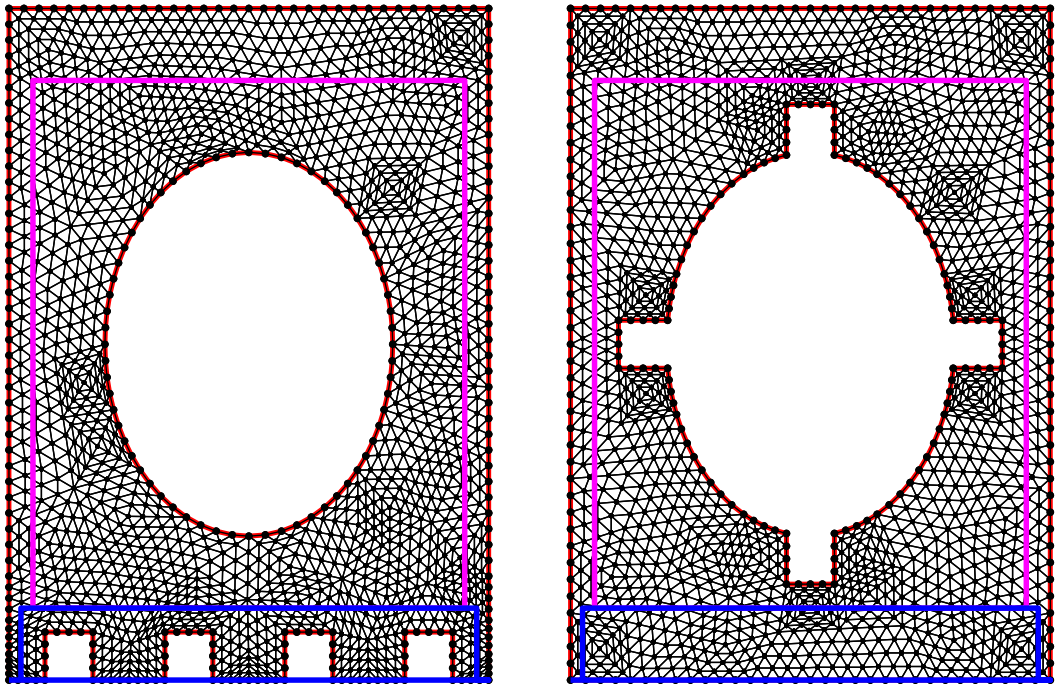
6. Implementing the boundary conditions can be done by first:

   (a) Iterating through all the *boundary* elements in the given mesh in a `for` loop

   (b) Use the node locations to then determine if these node locations are on an *interior* or *boundary* edge by an `if` and `elseif` statement

   Interior Edge `if` $\vec{x}_1 \in [\vec{x}_{min}, \vec{x}_{max}]$ & $\vec{x}_2 \in [\vec{x}_{min}, \vec{x}_{max}]$ then formulate A_{ij} from imposed Robin boundary conditions by integrating over the edges to determine $H_{ext}\underline{\underline{M}}^{\text{edge}}$ which will go to the corresponding global stiffness matrix $\underline{\underline{A}}$ matrix. For the boundary edges imposed in my problem statement the boundary edge will be deemed as the cold side of the heat transfer

   Boundary Edge `elseif` $(\vec{x}_1 \cdot [0,1] = y_{bottom,edge}$ & $\vec{x}_2 \cdot [0,1] = y_{bottom,edge})$ `or` $(\vec{x}_1 \in [\vec{x}_{min}, \vec{x}_{max}]$ & $\vec{x}_2 \in [\vec{x}_{min}, \vec{x}_{max}])$ then formulate A_{ij} from imposed Robin boundary conditions by integrating over the edges to determine $H_{ext}\underline{\underline{M}}^{\text{edge}}$ which will go to the corresponding global stiffness matrix $\underline{\underline{A}}$ matrix. For the boundary edges imposed in my problem statement the boundary edge will be deemed as the hot side of the heat transfer



(a) Boundary edges visualized for notched nozzle wall configuration.

(b) Boundary edges visualized for notched interior channel configuration.

Figure 4: Implementing and constructing boundary edges into Matlab simulation.

By implementing these `if` and `elseif` statements then this allows for an area of points that allows for easy calculation of what remains on the the interior or boundary edges. These `if` and `elseif` statements will result in the following boundaries shown in Figure 4 where the *blue* section is the boundary edge and *magenta* is the interior edge

(c) After determining which edges will be heated/chilled I will impose Robin boundary conditions in which will require a separate `for` loop in which I again formulate g and x for each node to determine the heat flux through each segment or,

$$\underline{\underline{A}}(g,g) = \underline{\underline{A}}(g,g) + h_{ext}\Delta L \begin{bmatrix} 1/3 & 1/6 \\ 1/6 & 1/3 \end{bmatrix}$$

Where, h_{ext} is the heat transfer coefficient unique to the gas and ΔL is the length of the edge segment. Then to formulate the $\underline{F}$ I used the following formula,

$$\underline{F}(g,1) = \underline{F}(g,1) + h_{ext}T_{ext}\frac{\Delta l}{2}$$

7. Lastly, after populating $\underline{\underline{A}}$ and $\underline{F}$ simply solve for the temperature field by:

$$\underline{T} = \underline{F} \cdot \underline{\underline{A}}^{-1}$$

This temperature field $\underline{T}$ gives the temperatures for each corresponding node location.

8. Plotting the temperature field can be done using Matlab's `patch` command to effectively give a contoured field of the temperature gradients

9. Solving for the integrated heat flux will be done numerically

$$Q = \int_{\partial\Omega_h} h_{ext}(T_{ext} - T)dl$$

This integral will be computed numerically and will approximate T to be approximately linear along the edge such that this integral can be approximated through a summation represented by,

$$Q_h \approx \sum_{\text{Edges}} h_{ext,h}(T_{ext,h} - \overline{T})\Delta L$$
$$Q_c \approx \sum_{\text{Edges}} h_{ext,c}(\overline{T} - T_{ext,c})\Delta L$$
$$\overline{T} = \frac{1}{2}(T_i + T_j)$$

This method has been implemented into my Matlab code and for further reference see Appendices 1,2, 3 on Pages 17-20.

2.2 Verification of the Finite-Element System Solver

To verify that my finite-element solver is working properly, I will perform a verification test over a simple domain that I can solve analytically for. Below in Figure 5 is a simple 2D case that is symmetric across the y-plane allowing for a simple 1D linearization to allow for a simple analytical answer to verify the functionality of my code.

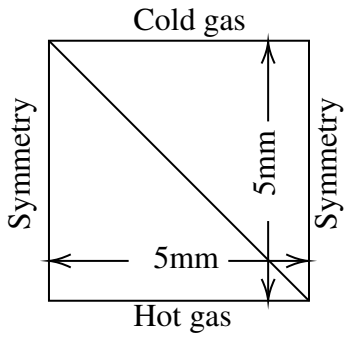


Figure 5: Verification domain.

To verify my code's functionality, I will generate the `V,E,B` matrices to input into my code and check the approximated solution against the analytical. Since node location numbering is conventionalized as increasing in a counter-clockwise direction (Node 1 in the bottom left, 2 bottom right, 3 top right, 4 top left) this gives the following input matrices.

$$E = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad V = \begin{bmatrix} 0 & 0 \\ 5 & 0 \\ 5 & 5 \\ 0 & 5 \end{bmatrix} \cdot 10^{-3}$$

2.2.1 Analytical Solution to One-Dimensional Heat Equation

Since there is symmetry across the the left and right-hand side of the domain, this partial-differential equation can be reduced into a one-dimensional differential equation. This will allow for the confirmation and verification that my code has been correctly implemented and the results for the complicated meshes are accurate and approximated to the analytical solution.

Firstly the general form of the steady-state heat equation is given as,

$$k\nabla^2 T = 0 \tag{5}$$

Where in Equation 5 above, k is the conductivity.

This expression can be rewritten as

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

In 1D, this simplfies to,

$$\frac{d}{dx}\left(k\frac{dT}{dx}\right) = 0$$

So that,

$$k\frac{dT}{dx} = q = \text{constant}$$

Since the heat flux rate is constant, temperature gradient will also be constant thus,

$$q_{ext,h} = q_{ext,c}$$

Where $q_{ext,h}$, $q_{ext,c}$ are,

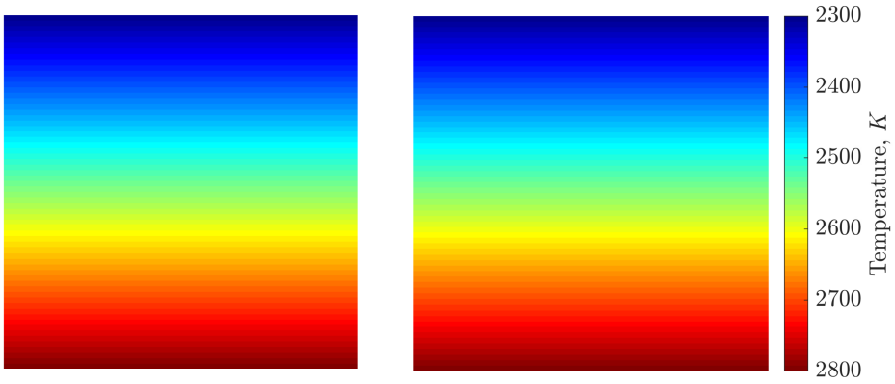
$$q_{ext,h} = h_{ext,h}(T_{ext,h} - T_{int,h})$$
$$q_{ext,c} = h_{ext,c}(T_{int,c} - T_{ext,c})$$

Using the above relations, considering the steady-state solution* the equation for the temperature field within the simple 2D symmetric case is,

$$\boxed{T(x) = 2800 - 100 \cdot 10^3 x,\ K} \tag{6}$$

Where x has units of meters.

Plotting the approximated solution to the analytical gives,



(a) Finite-Element approximated solution.

(b) Analytical solution to 1-D heat equation.

Figure 6: Direct comparison between the approximated 1D-heat equation to the analytical solution.

2.2.2 Discussion of Approximated Solutions:

As shown above these are shown that they are indeed similar in values but a closer look to the node temperatures shows that the ΔT at each nodes are given as,

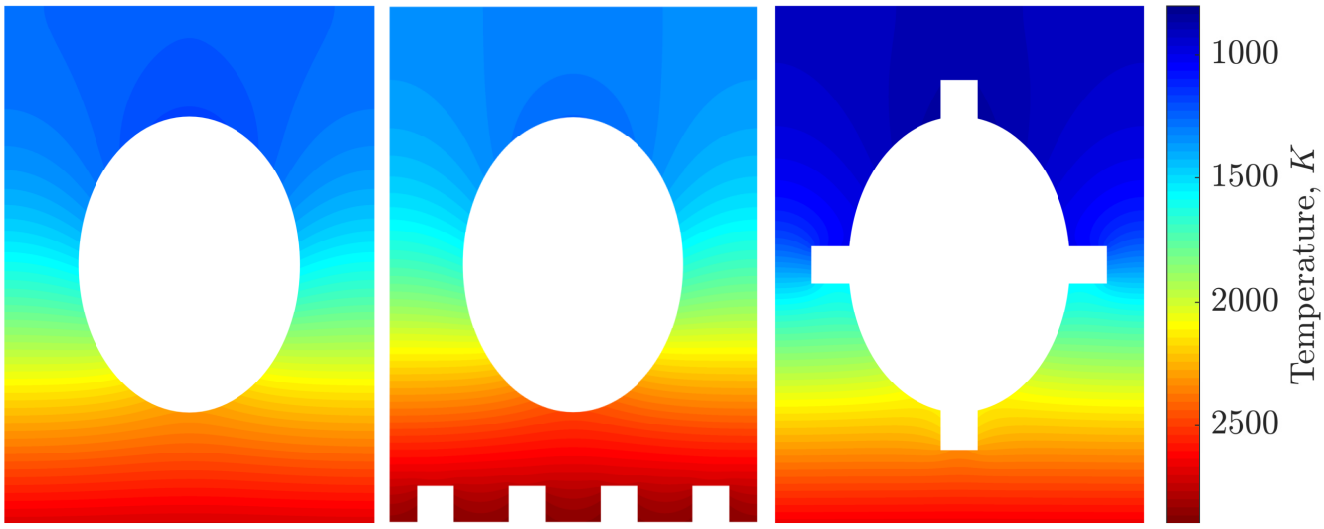
$$\Delta T = \begin{bmatrix} -0.45474735088646 \\ -0.45474735088646 \\ 0.90949470177293 \\ 1.36424205265939 \end{bmatrix} \cdot 10^{-12} K$$

From the ΔT values above, I can confirm that my finite-element solver is indeed implemented correctly and will return the approximated answer to machine precision or $\mathcal{O}(10^-10)$ or smaller. This confirms that my simulations are true and accurate.

*Where the heat fluxing into the system is equivalent to the heat fluxing out.

2.3 Nozzle Wall Temperature Fields

After implementing the finite-element solver, I was able to numerically approximate the temperature field within the nozzle walls. Below in Figure 7 are the results of varying notches in the nozzle wall and the approximated temperature fields.



(a) Temperature field for baseline nozzle wall.

(b) Temperature field for notched interior nozzle wall.

(c) Temperature field for notched regeneratively cooled channel.

Figure 7: Nozzle wall temperature fields for meshes (M3,M5,M7) in K.

2.3.1 Impact of Notches on Temperature Distribution

As shown in Figures 7a,7b,7c the temperature follows a mostly linear gradient from the interior nozzle wall to about halfway up the interior channel. This linearity is not unique in the sense that it is shared across all three the channeled regions.

However, when looking closely to the Figures the main deviations can be seen on Figures 7b, 7c. In Figure 7b the notched interior wall (in contact with the hot gasses) allows more heat to be transferred to the nozzle wall and can be seen as having the hottest temperatures along the interior nozzle wall of the three simulated. In Figure 7c the top of the channel notch contours at $\approx 1000K$ follow closely to the notch and diffuse through the rest of the top of the nozzle wall.

As observed in Figures 7a,7b,7c, the most efficient cooling method is a notched interior channel also known as the cold-notched channel.

2.4 Heat Flux Verification

Implementing the integrated heat flux will be done numerically to approximate the heat flux which is defined as,

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

To approximate this numerically I will simplify the expression to allow for the summation over the interior points which results in,

$$Q_h \approx \sum_{\text{Edges}} h_{ext,h}(T_{ext,h} - \overline{T})\Delta L$$
$$Q_c \approx \sum_{\text{Edges}} h_{ext,c}(\overline{T} - T_{ext,c})\Delta L$$

Where $\overline{T}$ is defined as,

$$\overline{T} = \frac{1}{2}(T_i + T_j)$$

2.4.1 Analytical Solutions to Heat Flux

The heat flux through the hot/cold edges can be defined from the following equation,

$$Q = \int_{\partial\Omega_h} \vec{q} \cdot \hat{n} dl = h_{ext}(T_{edge} - T_{\text{ext}})\Delta L$$

Using the analytical solution for the 1D temperature field gives that for the simple 2D case that the heat flow through the edges are,

$$\begin{aligned} Q_{ext,h} &= 20,000(3,000 - 2,800) \cdot 0.005 \\ &= \boxed{20,000 \ \frac{W}{m}} \\ Q_{ext,c} &= 2,000(2,300 - 300) \cdot 0.005 \\ &= \boxed{20,000 \ \frac{W}{m}} \end{aligned}$$

2.4.2 Verifying Matlab Implementation

Comparing the approximated answers to the analytical solutions gives that $\Delta Q_{ext} = Q_{approx} - Q_{exact}$ is,

$$\Delta Q_{ext,h} = -4.7294 \cdot 10^{-11} \ \frac{W}{m}$$
$$\Delta Q_{ext,c} = -7.2760 \cdot 10^{-12} \ \frac{W}{m}$$

Looking above these values are *very* close to each other and is on the same order of magnitude as machine precision $\mathcal{O}(10^{-10})$ or smaller. At this point these approximations are essentially equivalent to the analytical solution. Thus, this verifies that my heat flux approximation is functional and is accurate.

2.5 Convergence Study with Mesh Sizes

Using the baseline domain, I will perform a convergence study on the heat flux output and the effects of varying the mesh size. I will be using three meshes, where each is more refined from the previous. However, since there is no analytical solution to this problem, the most refined mesh will be approximated as the *analytical solution*. The error will be defined as,

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

Using the approximated Q values and the Q_{Exact} from the refined mesh, conducting the convergence study results in Figure 8 shown below.

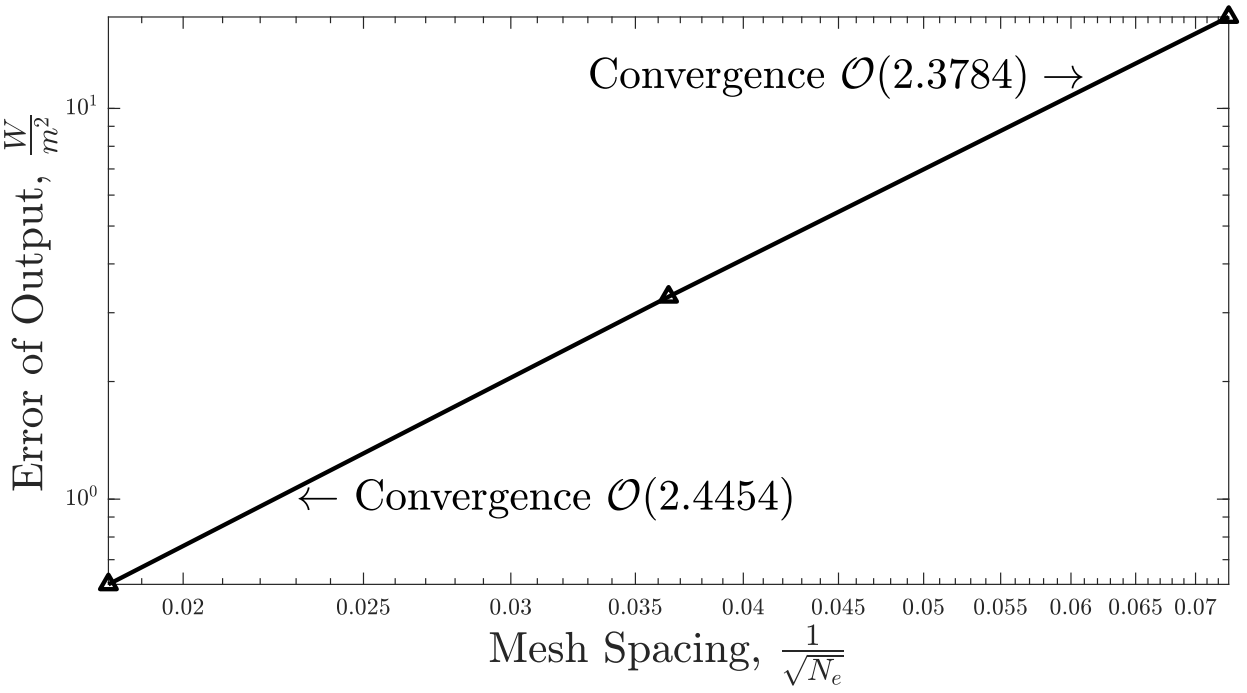


Figure 8: Effects of mesh sizing on the approximated heat flux.

2.5.1 Discussion of Convergence Rates with Various Mesh Sizes

As shown above in Figure 8, as the mesh spacing decreases the error of the approximated Q decreases monotonically and will minimize the error. Again, from ths Figure I determined the convergence rate to be $\mathcal{O}(\Delta h^{2.5})$ where Δh would be the mesh sizing. This means that if you halved the mesh size that the error would decrease by the error at the large step raised to the inverse of the convergence rate.

2.6 Effects of Notched Nozzle Walls

The most common way of analyzing the effectiveness of the regeneratively cooled channels is to check the Q of the steady state heat transfer. In this analysis a higher Q is preferred, as it directly means that more heat is available to be dissipated from the nozzle walls and raise the efficiency of the rocket engine. Computing the heat flux transfer, Q from each mesh gives for each mesh that,

$$Q_{h,M3} = 59,253.98 \ \frac{W}{m}$$
$$Q_{h,M5} = 64,190.71 \ \frac{W}{m}$$
$$Q_{h,M7} = 68,219.49 \ \frac{W}{m}$$

2.6.1 Analysis of Notching Nozzle Walls

As shown above, there is a clear and evident advantage to notching the walls of nozzle wall. In **both** notched cases, they were able to flux more heat in the steady state solution. The reason why the notches can have a higher heat flux is because there is more surface area that comes in contact with the boundary edges and allows for more diffusion of heat.

As shown from the Q approximations, the cold-notched configuration allows for the more heat flux and would most benefit the rocket engines performance. The reason why this configuration is most efficient is because the notches allow more surface area to allow for heat transfer but the cold gases combine to give it the highest heat flux.

Appendices

A Driving Code for Simulating Heat Transfer

Algorithm 1 shown below is the main driving code that is used in this simulation. This code is responsible for generating the plots that have been used in this report and calls all the supporting functions to generate the temperature field T or the heat flux Q.

Algorithm 1: Main Matlab Driving Function.

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%      Dan Card, Aero 423 [Project 3]
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
clear all; clc; close all
format short
format compact
set(groot,'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/test.eps')
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% Input mesh from user
% error = true;
% while error == true
%     % Ask for input
%     mesh = input('\n\nWhat Mesh to analyze (Number) : ');
%     if mesh > 8 || mesh < 0
%         % Error handling
%         fprintf('\nError: Wrong mesh number, try again')
%         error = true;
%     else
%         error = false;
%     end
% end

[E,B,V,T] = calcTemp(8);
[qh,qc] = heat_flux(E,B,V,T,8)

figure()
patch('Vertices', V, 'Faces', E,'FaceVertexCData',T', 'FaceColor','interp','
    EdgeColor','none');
axis equal; axis tight; axis off;
colormap jet
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/analytical.eps')
c = colorbar;
c.FontSize = 16;
c.Direction = 'reverse';
c.TickLabelInterpreter = 'latex';
c.Label.String = 'Temperature, $K$';
c.Label.Interpreter = 'latex';
%export_fig('figs/verification_mesh.eps')

%% Task 3
[E,B,V,T] = calcTemp(3);
min(T)
max(T)
[qh,qc] = heat_flux(E,B,V,T,3);

figure()
patch('Vertices', V, 'Faces', E,'FaceVertexCData',T', 'FaceColor','interp','
    EdgeColor','none');
axis equal; axis tight; axis off;
caxis([800 2900])
colormap jet
```

```
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/temp_mesh3.eps')

[E,B,V,T] = calcTemp(5);
min(T)
max(T)
[qh,qc] = heat_flux(E,B,V,T,5);
figure()
subplot(1,3,2)
patch('Vertices', V, 'Faces', E,'FaceVertexCData',T', 'FaceColor','interp','
    EdgeColor','none');
axis equal; axis tight; axis off;
caxis([800 2900])
colormap jet
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/temp_mesh5.eps')

[E,B,V,T] = calcTemp(7);
min(T)
max(T)
[qh,qc] = heat_flux(E,B,V,T,7);

figure()
patch('Vertices', V, 'Faces', E,'FaceVertexCData',T', 'FaceColor','interp','
    EdgeColor','none');
axis equal; axis tight; axis off;
colormap jet
c = colorbar;
caxis([800 2900])
c.FontSize = 16;
c.Direction = 'reverse';
c.TickLabelInterpreter = 'latex';
c.Label.String = 'Temperature, $K$';
c.Label.Interpreter = 'latex';
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/temp_mesh7.eps')
%% Task 5

[E,B,V,T] = calcTemp(0);
nodes_0 = length(E);
[qh_1,~] = heat_flux(E,B,V,T,1);

[E,B,V,T] = calcTemp(1);
nodes_1 = length(E);
[qh_2,~] = heat_flux(E,B,V,T,2);

[E,B,V,T] = calcTemp(2);
nodes_2 = length(E);
[qh_3,~] = heat_flux(E,B,V,T,3);

[E,B,V,T] = calcTemp(3);
[qh_exact,~] = heat_flux(E,B,V,T,3);

err = [abs(qh_1-qh_exact), abs(qh_2-qh_exact), abs(qh_3 - qh_exact)];

srqt_inv_el = [1/sqrt(nodes_0), 1/sqrt(nodes_1), 1/sqrt(nodes_2)];
m_conv1 = log10(err(2)/err(1))/log10(srqt_inv_el(2)/srqt_inv_el(1));
m_conv2 = log10(err(3)/err(2))/log10(srqt_inv_el(3)/srqt_inv_el(2));

figure()
plot(srqt_inv_el,err, 'k-^', 'linewidth', 1.8)
text(0.033, 12, ['Convergence $\mathcal{O}($',num2str(m_conv1),'$)\rightarrow$'], '
    interpreter', 'latex', 'fontsize', 18)
text(0.023, 1, ['$\leftarrow$Convergence $\mathcal{O}($',num2str(m_conv2),'$)$'],
    'interpreter', 'latex', 'fontsize', 18)
xlabel('Mesh Spacing, $\frac{1}{\sqrt{N_{e}}}$', 'fontsize', 18)
ylabel('Error of Output, $\frac{W}{m^2}$', 'fontsize', 18)
set(gca, 'yscale', 'log')
set(gca, 'xscale', 'log')
set(gcf, 'Color', 'w', 'Position', [200 200 800 400]);
%export_fig('figs/convergence.eps')
```

B Functional Code to Approximate Temperature Field

Algorithm 2 shown below is responsible for approximating $\underline{\underline{A}}$ and $\underline{F}$ and returning the temperature field $\underline{T}$. This code imposes the correct boundary conditions for all meshes.

Algorithm 2: Calculating the Temperature Field.

```
function [E,B,V,T] = calcTemp(mesh)
    thot = 3000; % Hot gas temperature
    tcold = 300; % Cold gas temperature
    hhot = 20e3; % Hot gas heat transfer coefficient
    hcold = 2e3; % Cold gas heat transfer coefficient
    k_con = 40; % Steel thermal conductivity
    E = dlmread(['Meshes/M',num2str(mesh),'-E.txt']);
    B = dlmread(['Meshes/M',num2str(mesh),'-B.txt']);
    V = dlmread(['Meshes/M',num2str(mesh),'-V.txt']);
    if(mesh == 8) verify = true; else verify = false;
    end
    [num,~] = size(V); N = num;  % Number of Nodes
    [num,~] = size(E); Ne = num; % Number of Elements
    [num,~] = size(B); Nb = num; % Number of Boundaries
    % Pre-allocate and build global A matrix
    A = sparse(N, N); F = zeros(N,1);
    for k = 1:Ne
        g = E(k,:);
        x = V(g,:);
        Jacobi = [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 = 1/2*det(Jacobi);
        phi_x = [-1,-1; 1, 0; 0, 1]/Jacobi;
        ak = k_con.*phi_x * phi_x' .* area;
        A(g,g) = A(g,g) + ak;
    end
    epsi = 10^-10;
    % Determine the edge boundaries
    for k = 1:Nb
        node1 = V(B(k,1),:);
        node2 = V(B(k,2),:);
        g = B(k,:);
        deltal = norm(node2 - node1);
        if verify == true
            % Bottom Edge
            if node1(2) == 0 && node2(2) == 0
                A(g,g) = A(g,g) + hhot*deltal.*[1/3, 1/6; 1/6, 1/3];
                F(g,1) = F(g,1) + hhot*thot*deltal/2;
            % Interior Edge
            elseif node1(2) == 0.005 && node2(2) == 0.005
                A(g,g) = A(g,g) + hcold*deltal.*[1/3, 1/6; 1/6, 1/3];
                F(g,1) = F(g,1) + hcold*tcold*deltal/2;
            end
        else    % Bottom Edge
           if (abs(node1(2)+0.006)< epsi && abs(node2(2)+ 0.006)<epsi) || ...
                   ((node1(1) > -0.00475 && node1(1) < 0.00475) && (node2(1) >
                       -0.00475 && node2(1) < 0.00475))...
                   && (node1(2) < -0.0045 && node2(2) < -0.0045)
                A(g,g) = A(g,g) + hhot*deltal.*[1/3, 1/6; 1/6, 1/3];
                F(g,1) = F(g,1) + hhot*thot*deltal/2;
                % Interior Edge
            elseif (node1(1) > -0.0045 && node1(1) < 0.0045) && (node2(1) > -0.0045 &&
                 node2(1) < 0.0045) && ...
                   (node1(2) > -0.0045 && node1(2) < 0.0065) && (node2(2) > -0.0045
                       && node2(2) < 0.0065)
                A(g,g) = A(g,g) + hcold*deltal.*[1/3, 1/6; 1/6, 1/3];
                F(g,1) = F(g,1) + hcold*tcold*deltal/2;
            end
        end
    end
    T = F'*inv(A);
end
```

C Functional Code to Approximate Heat Transfer Q

Algorithm 3 below takes in the `E,B,V,T` matrices and the mesh number. This code loops through the node boundaries and numerically approximates the heat flux through each boundary edges and sums the flux through all the edges and returns the flux through the hot and cold boundaries.

Algorithm 3: Calculating the Heat Transfer Q.

```
function [qh, qc] = heat_flux(E,B,V,T,mesh)
    % Material Properties
    thot = 3000; % Hot gas temperature
    tcold = 300; % Cold gas temperature
    hhot = 20e3; % Hot gas heat transfer coefficient
    hcold = 2e3; % Cold gas heat transfer coefficient
    qh = 0;      % Pre-allocate q value
    qc = 0;      % Pre-allocate q value
    if(mesh == 8) verify = true; else verify = false;
    end
    [num,~] = size(V); N = num;  % Number of Nodes
    [num,~] = size(E); Ne = num; % Number of Elements
    [num,~] = size(B); Nb = num; % Number of Boundaries

    % Determine the edge boundaries
    epsi = 10 ^(-10);
    for k = 1:Nb
        node1 = V(B(k,1),:);
        node2 = V(B(k,2),:);
        Ti = T(B(k,1));
        Tj = T(B(k,2));
        tbar = 1/2*(Ti + Tj);
        deltal = norm(node2 - node1);

        if verify == true
            % Bottom Edge
            if node1(2) == 0 && node2(2) == 0
                qh = qh + hhot*(thot - tbar)*deltal;
            % Interior Edge
            elseif node1(2) == 0.005 && node2(2) == 0.005
                qc = qc + hcold*(tbar - tcold)*deltal;
            end
        else    % Bottom Edge
           if (abs(node1(2)+0.006)< epsi && abs(node2(2)+ 0.006)<epsi) || ...
                   ((node1(1) > -0.00475 && node1(1) < 0.00475) && (node2(1) >
                       -0.00475 && node2(1) < 0.00475))...
                   && (node1(2) < -0.004 && node2(2) < -0.004)
                qh = qh + hhot*(thot - tbar)*deltal;
                % Interior Edge
            elseif (node1(1) > -0.0045 && node1(1) < 0.0045) && (node2(1) > -0.0045 &&
                 node2(1) < 0.0045) && ...
                   (node1(2) > -0.005 && node1(2) < 0.007) && (node2(2) > -0.005 &&
                       node2(2) < 0.007)
                qc = qc + hcold*(tbar - tcold)*deltal;
            end
        end
    end
end
```