Steady-State Heat Conduction on Arbitrary 2D Mesh

EGME 541: Advanced Finite Element Methods
Instructor: Justin Tran

California State University, Fullerton

Department of Mechanical Engineering

Author:

Gerardo Gutierrez

Submitted on: May 20th, 2024

Contents

Introduction

The finite element method is a useful technique used to solve complex engineering problems. Many problems in the engineering field are more complex than what can be efficiently solved using analytical solving skills. Therefore, engineers apply the finite element method to solve these complex problems. These complexities can take the form of complex geometries or boundary conditions. For engineers, the Finite Element Method is crucial to quickly computing stress, heat distributions, and material properties of specified scenarios. This method involves breaking down a large complex problem into smaller parts that are easier to solve called, elements. Then, after applying solving methods specific to the finite element method, we can compute a solution for these elements and assemble them into a final full solution that represents our initial problems change. While this tool is useful, it can also be dangerous to use. Take the example of an inexperienced engineer solving a complex problem. If this engineer does not apply boundary conditions correctly, a solution can be still computed. The danger is in not realizing that the computed solution may be incorrect. This paper aims to develop the understanding necessary to solve a 2D Finite Element Mesh with an applied heat load and then apply this solution in a graphical representation.

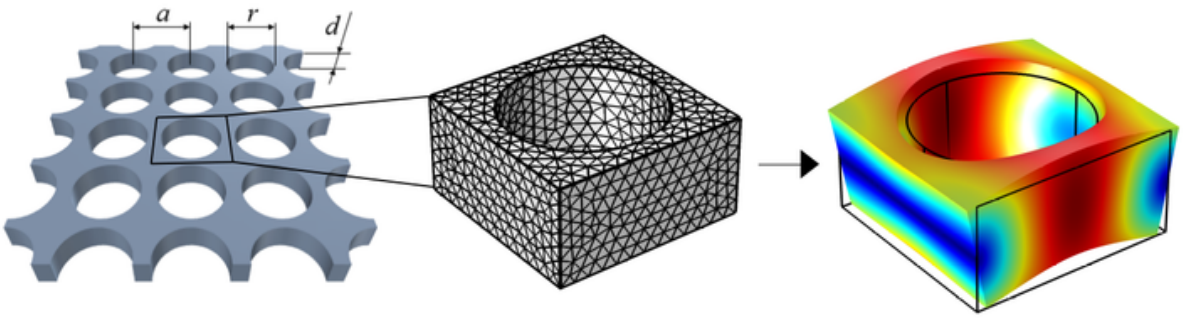


Figure 1. Example of Finite Element Method Application

This paper will solve a problem containing a steady-state heat conduction scenario on an arbitrary 2D mesh. This scenario will contain heat generation and an initial heat distribution field. To solve this, we will begin with the foundational equations and specific techniques that were used to compute this heat conduction on a 2D plot in Matlab.

Methodology

For the purposes of this paper, we used methods and strategies specified in the isoparametric approach using finite elements. In this section we will discuss the different equations used, their various terms, and why we used them.

Firstly, we need to discretize our problem. Discretization is the process of breaking down a large domain into smaller parts. We will then solve our problem over this discretized domain. For this example, we will use triangular elements when discretizing. We are also able to discretize with other shapes. However, discretizing with triangles allows us to keep our shape functions linear. We will utilize Matlab's PDE toolbox to generate a mesh for our needs. This section will focus on the math needed to compute a solution over a defined mesh.

We can now use shape functions in order to describe our elements mathematically. For triangular elements. we will have 3 nodes

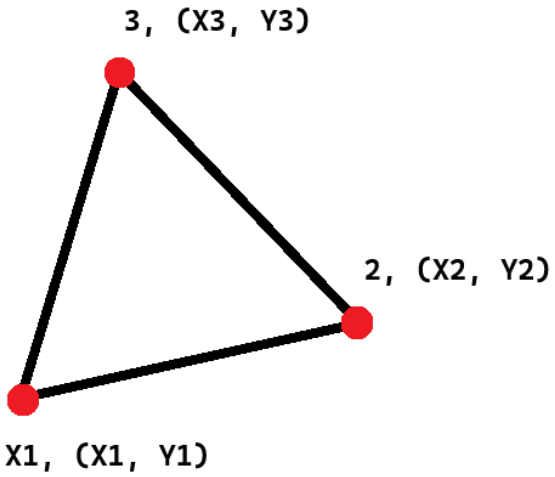


Figure 2. Arbitrary Triangular Element

The nodes attached to this triangle can be described using the following element solution

$$T^e(x,y) = T_1 N_1(x,y) + T_2 N_2(x,y) + T_3 N_3(x,y) \tag{1}$$

Where $T_{1,2,3}$ are described using a linear functions on both axes

$$\begin{aligned} T_1 &= a_0 + a_1 x_1 + a_2 y_1 \\ T_2 &= a_0 + a_1 x_2 + a_2 y_2 \\ T_3 &= a_0 + a_1 x_3 + a_2 y_3 \end{aligned} \tag{2}$$

With the associated shape functions

$$\begin{aligned} N_1(x,y) &= \frac{1}{2a}[x_2 y_3 - x_3 y_2 + (y_2 - y_3)x + (x_3 - y_2)y] \\ N_1(x,y) &= \frac{1}{2a}[x_2 y_3 - x_3 y_2 + (y_2 - y_3)x + (x_3 - y_2)y] \\ N_1(x,y) &= \frac{1}{2a}[x_2 y_3 - x_3 y_2 + (y_2 - y_3)x + (x_3 - y_2)y] \end{aligned} \tag{3}$$

We can now use a concept called the Gauss quadrature to approximate the integral solution. We do this instead of taking the integral of a function in order to simplify the amount of math we must to do solve a problem.

For example. The following double integral can be approximated as such

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy \approx \sum w_i f(x_i, y_i) \tag{4}$$

Note that w_i is defined as our weighting function. this is an addition to the Gauss quadrature that accounts for error when solving. Also note that x_i and y_i are used as substitutions for the variables in our original integrated functions,

For our use case, we will use the following weights and points

Locations	Weights
$\varepsilon_1 = \frac{2}{3}$, $\eta_1 = \frac{1}{6}$	$w_1 = 1/3$
$\varepsilon_2 = \frac{1}{6}$, $\eta_2 = \frac{2}{3}$	$w_2 = 1/3$
$\varepsilon_3 = \frac{1}{6}$, $\eta_3 = \frac{3}{6}$	$w_3 = 1/3$

Table 1. Gauss Points and Locations

When generating a mesh to solve over, we notice that the elements are arranged haphazardly over our described area. This is beneficial for being able to effectively cover any size of mesh we can think of. Where this becomes less useful is when we have to solve for anything over this mesh. We want to solve these problems by computing integrals on these 2D elements.

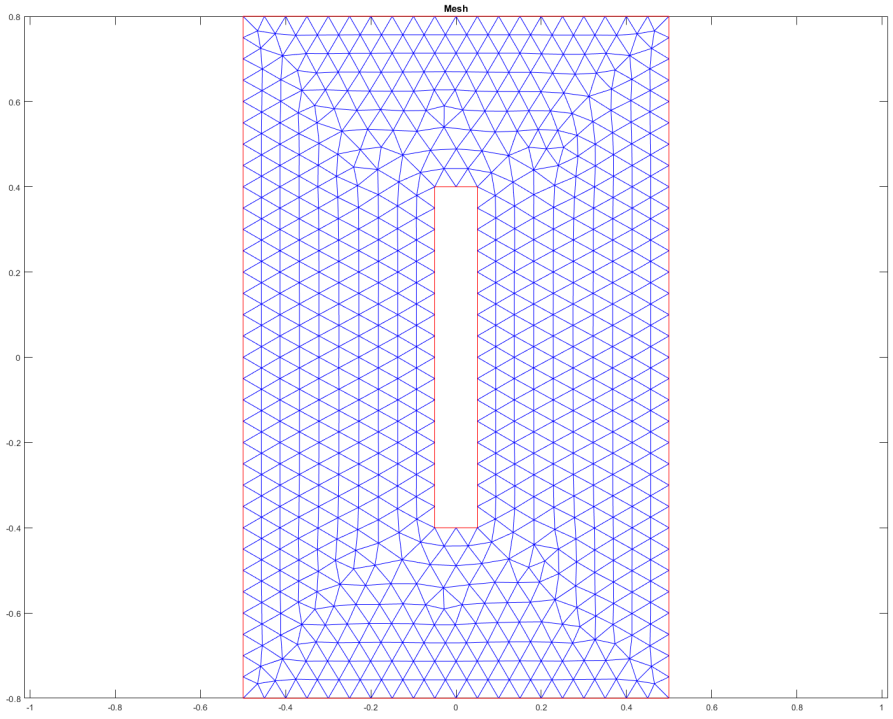


Figure 3. Project Mesh

Because of this issue, we will introduce the isoparametric concept which does a change of coordinates on our arbitrary element from ”physical space” to ”parametric space”

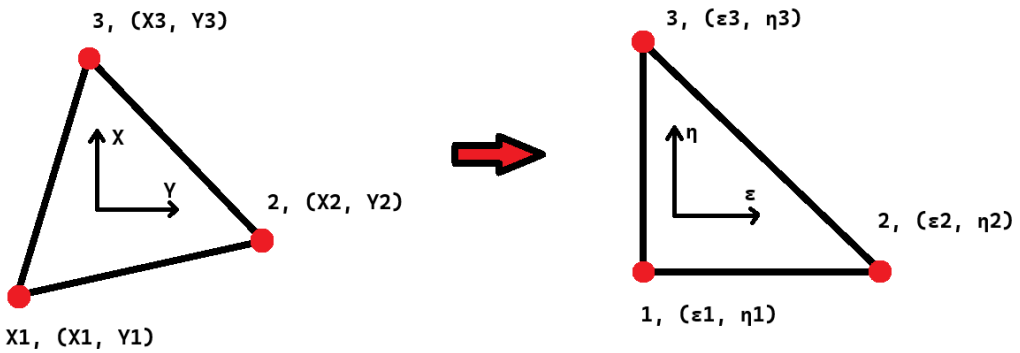


Figure 4. Arbitrary Element Mapped to Parametric Space

We now describe the element solution in this parametric space using an interpolation, or shift, factor.

$$T^e(\varepsilon,\eta) = T_1N_1(\varepsilon,\eta) + T_2N_2(\varepsilon,\eta) + T_3N_3(\varepsilon,\eta) \tag{5}$$

With the associated shape functions

$$\begin{aligned} N_1(\varepsilon,\eta) &= \varepsilon \\ N_1(\varepsilon,\eta) &= \eta \\ N_1(\varepsilon,\eta) &= 1-\varepsilon-\eta \end{aligned} \tag{6}$$

It is crucial to now develop mapping equations to express our physical problem in parametric space. They will take the form of x and y equations

$$x(\varepsilon,\eta) = X_1N_1(\varepsilon,\eta) + X_2N_2(\varepsilon,\eta) + X_3N_3(\varepsilon,\eta) \tag{7}$$

$$y(\varepsilon,\eta) = Y_1N_1(\varepsilon,\eta) + Y_2N_2(\varepsilon,\eta) + Y_3N_3(\varepsilon,\eta) \tag{8}$$

In Matlab, you will take the location of each node in your given element and compute a location matrix based on all 6 of those computed values

$$N_{location} = \begin{bmatrix} X_1N_1(\varepsilon,\eta) & Y_1N_1(\varepsilon,\eta) \\ X_2N_2(\varepsilon,\eta) & Y_2N_2(\varepsilon,\eta) \\ X_3N_3(\varepsilon,\eta) & Y_3N_3(\varepsilon,\eta) \end{bmatrix} \tag{9}$$

We now can find the derivatives of these shape functions. We call ε and η the Gauss points. The derivatives of these equations are used to map.

$$\begin{aligned}
\frac{\partial x}{\partial \varepsilon} &= x_1\frac{\partial N_1}{\partial \varepsilon} + x_2\frac{\partial N_2}{\partial \varepsilon} + x_3\frac{\partial N_3}{\partial \varepsilon} \\
\frac{\partial x}{\partial \eta} &= x_1\frac{\partial N_1}{\partial \eta} + x_2\frac{\partial N_2}{\partial \eta} + x_3\frac{\partial N_3}{\partial \eta} \\
\frac{\partial y}{\partial \varepsilon} &= y_1\frac{\partial N_1}{\partial \varepsilon} + y_2\frac{\partial N_2}{\partial \varepsilon} + y_3\frac{\partial N_3}{\partial \varepsilon} \\
\frac{\partial y}{\partial \eta} &= y_1\frac{\partial N_1}{\partial \eta} + y_2\frac{\partial N_2}{\partial \eta} + y_3\frac{\partial N_3}{\partial \eta}
\end{aligned} \tag{10}$$

Where,

$$\begin{aligned}
\frac{\partial N_1}{\partial \varepsilon} = 1,\ \frac{\partial N_2}{\partial \varepsilon} = 0,\ \frac{\partial N_3}{\partial \varepsilon} = -1 \\
\frac{\partial N_1}{\partial \eta} = 0,\ \frac{\partial N_2}{\partial \eta} = 1,\ \frac{\partial N_3}{\partial \eta} = -1
\end{aligned} \tag{11}$$

Save these in Matlab in a matrix for future use

$$dN_{mat} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \tag{12}$$

To simplify this, we will summarize these equations into a Jacobian Matrix

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \varepsilon} & \frac{\partial x}{\partial \varepsilon} \\ \frac{\partial y}{\partial \eta} & \frac{\partial x}{\partial \eta} \end{bmatrix} = [N_{location}]\,[dN_{mat}] \tag{13}$$

let us also take the inverse of this Jacobian matrix. The reasoning for this will be clear later.

$$[J^{-1}] = \begin{bmatrix} \frac{\partial x}{\partial \varepsilon} & \frac{\partial x}{\partial \varepsilon} \\ \frac{\partial y}{\partial \eta} & \frac{\partial x}{\partial \eta} \end{bmatrix}^{-1} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \varepsilon}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \tag{14}$$

Now in order to use these in an element stiffness matrix for a 3 node triangle. We must first define the format our solution will take and work our way to be able to solve for each part of it

$$\int_{-1}^{1}\int_{-1}^{1} \Delta w{\cdot}k{\cdot}\Delta T dA = \begin{bmatrix} w1 & w2 & w3 \end{bmatrix} \int_{-1}^{1}\int_{-1}^{1} \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_1}{\partial \eta} \\ \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_2}{\partial \eta} \\ \frac{\partial N_3}{\partial \varepsilon} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} J \end{bmatrix}^{-1} k \begin{bmatrix} J \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_3}{\partial \varepsilon} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} dA \tag{15}$$

Where k is our thermal conductivity. For this problem we will define $k = 50 W m^{-1} k^{-1}$. We can see that we factor in our weighting values and our inverse Jacobian to solve for our element stiffness matrix.

Lets now convert our shape function derivatives to be expressed in physical space using appropriate conversion factors. We cannot keep our shape function derivatives in the same form they are in. This is because they are given in terms of the parametric space and not the needed physical space. This will be used in our element stiffness matrix solution

$$\begin{aligned} \frac{\partial N_i}{\partial x} &= \frac{\partial N_i}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial x} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial x} \\ \frac{\partial N_i}{\partial y} &= \frac{\partial N_i}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial y} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial y} \end{aligned} \tag{16}$$

The keen observer would notice that the conversion factors in red are our entries from our inverse Jacobian matrix. This is why it is useful to calculate.

Let us now write our physical coordinate shape functions derivatives in terms of our new parametric shape function derivatives and inverse Jacobian matrix.

$$B = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial \varepsilon}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \varepsilon}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_3}{\partial \varepsilon} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \tag{17}$$

Similarly, we can express the weighting function gradient, Δw, as the transposed parametric

shape function matrix multiplied by the inverse Jacobian

$$\begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial y} \\ \frac{\partial N_3}{\partial x} & \frac{\partial N_3}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \varepsilon} & \frac{\partial N_1}{\partial \eta} \\ \frac{\partial N_2}{\partial \varepsilon} & \frac{\partial N_2}{\partial \eta} \\ \frac{\partial N_3}{\partial \varepsilon} & \frac{\partial N_3}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial \varepsilon}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \varepsilon}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix} \tag{18}$$

We can now combine these contributions to compute our element stiffness matrix for a given point in a mesh. We can then use our Gauss Quadrature to simplify this integral into a sum for the purposes of adapting our code into Matlab. We loop over this sum and map our values accordingly into a global matrix defined by the user.

$$K_{elem} = \int_{-1}^{1} \int_{-1}^{1} B' k B det(J) d\varepsilon d\eta = \sum_{g=1}^{N_g} (B_g^T k B_g det(J)_g w_g) \tag{19}$$

By following these steps, we have developed a basic algorithm needed to solve for the element stiffness matrix contributions on a mesh and apply them onto a global stiffness matrix.

The next step for our purposes would be to deal with the boundary conditions for our problem. We have defined our Dirichlet BCs to be the outer most edges of our mesh. We defined these edges to have a starting temperature of 100 degrees C or 25 degrees C. We have described our meshes edges 1 and 2 to have a temperature of 100C and edges 6 and 7 to have a temperature of 25C

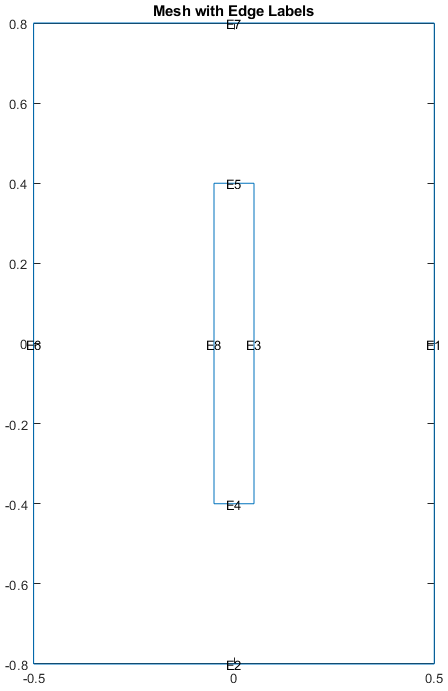


Figure 5. Mesh Edges

In Matlab, we can simply compute these temp contributions as a zero of the row they affect in the global matrix and a 1 on the nxn value of the row.We also will plug in the corresponding temperature for the forcing vector. Of course, these edges are created by the elements on them. The extremes of these edges help in understanding which nodes are affected by this BC.

Heat generation can be added as well. For our problem we were given a heat source of the following magnitude $s(x, y) = 1000x - 5000y$

To apply to parametric space we will do the following

$$s_g = s(x = x_g, y = y_g) = s(x_g, y_g) \tag{20}$$

We can do the same type of Gauss quadrature for the the element forcing vector and source terms. This will be added to our algorithm when needed

$$f_s^e = \int_{-1}^{1}\int_{-1}^{1}[N]s(\varepsilon,\eta)det(J)d\varepsilon d\eta = \sum_{g=1}^{N_g}(B_g' k B_g det(J)_g w_g) \tag{21}$$

Lastly, we can also plug in Neumann condition for this problem using what a line integral or a 1D integral. We will define the Neumann edges as 3,4 5, and 8 of our Mesh edges. A value must be applied to these edges. We will use a heat flux of 10000 W/m^2 for all Neumann edges

$$\oint \begin{bmatrix} N1(x,y) \\ N2(x,y) \\ N3(x,y) \end{bmatrix} \bar{q} ds \rightarrow \oint \begin{bmatrix} N1(\varepsilon,\eta) \\ N2(\varepsilon,\eta) \\ N3(\varepsilon,\eta) \end{bmatrix} \bar{q} \tag{22}$$

This is a bit challenging to compute in Matlab, we will use the following rules when solving for our Neumann elements with two edge nodes. These edge nodes will describe our Gauss integration points

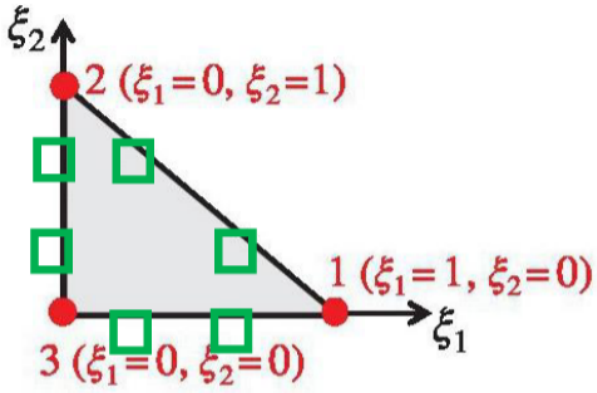


Figure 6. Gauss Integration Points

$$
\begin{aligned}
Edge1-2 &: \left(\frac{\sqrt{3}-1}{2\sqrt{3}}, \frac{\sqrt{3}+1}{2\sqrt{3}}\right) and \left(\frac{\sqrt{3}+1}{2\sqrt{3}}, \frac{\sqrt{3}-1}{2\sqrt{3}}\right) \\
Edge2-3 &: \left(0, \frac{\sqrt{3}-1}{2\sqrt{3}}\right) and \left(0, \frac{\sqrt{3}+1}{2\sqrt{3}}\right) \\
Edge1-3 &: \left(\frac{\sqrt{3}-1}{2\sqrt{3}}, 0\right) and \left(\frac{\sqrt{3}+1}{2\sqrt{3}}, 0\right)
\end{aligned} \tag{23}
$$

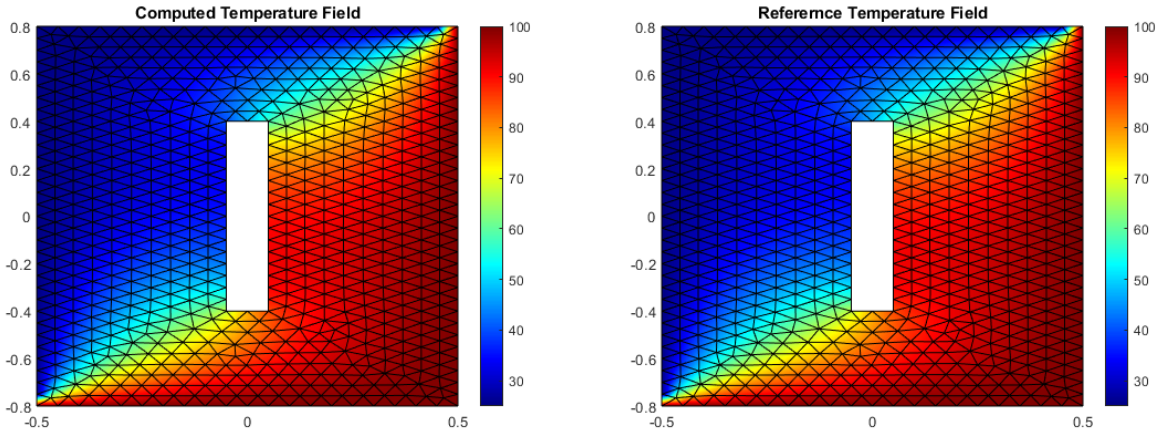
Add these contributions to the element forcing vector at the described element locations. This will give us a final solution we can apply to our mesh and finally, see results for our efforts.

Results and Discussion

After running through the various requirements needed to compute our scenarios for this project. The following results were gathered in order to compare.

Comparison of Reference and Computed Fields with Dirichlet BCs

The following figures represents a computed mesh of the described scenario. For this section we have chosen to only show the Dirichlet boundary conditions as those are foundational in describing a scenario. Notice how the initial temperature distribution field described in the methodology section is shown her as a hot and cold side. These distributions are our Dirichlet conditions described above.



((a)) Computed Plot ((b)) Reference Plot

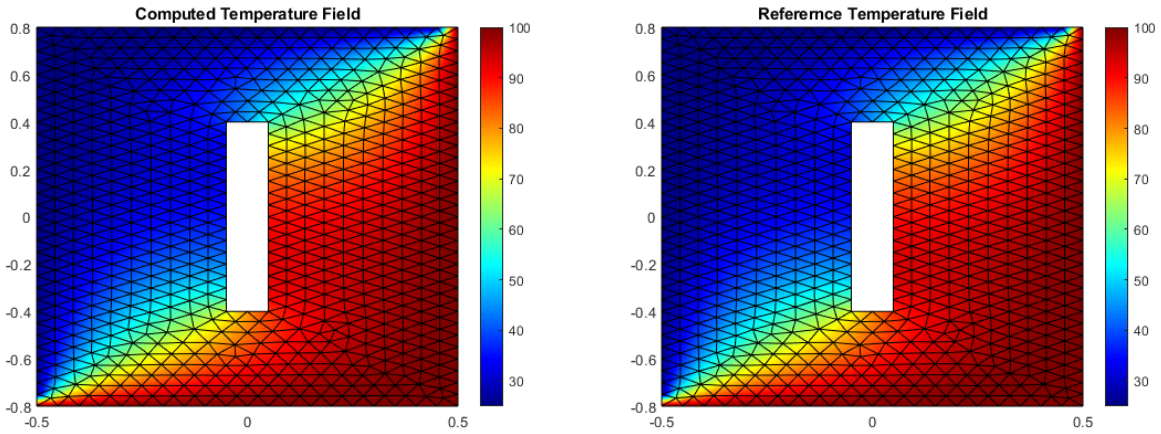
Figure 7. Comparison of Computed and Reference Fields

Here, we see our computed plot matches exactly with our reference plot. Our error is 5.0136e-11 which is extremely low. This means that our method for solving these problems works as intended. Furthermore, we should have seen a small deviation on the corners were the two temperatures met. This would alter our plot so that the lower temperature would supersede the higher temperature. We avoided this error by understanding that a higher temperature would most likely be the most prominent between the two. This was done by altering our code to solve for the

temperatures in order of lowest to highest.

Comparison of Reference and Computed Fields with heat generation

The following figures represents a computed mesh of the described scenario. For this section we use a heat generation value of 1. This heat generation is solved for using the described method above.



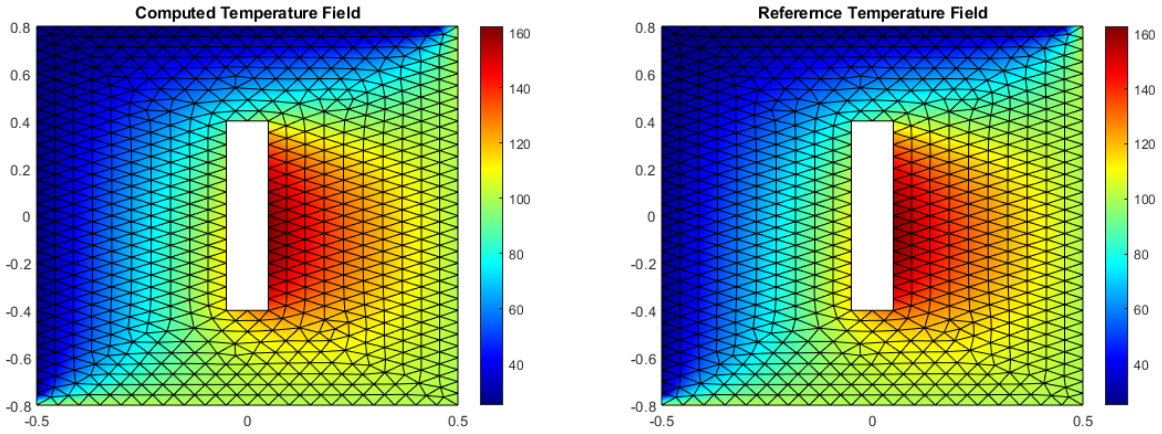
((a)) Computed Plot ((b)) Reference Plot

Figure 8. Comparison of Computed and Reference Fields

The heat generation in this scenario is set a value of 1. This is what was instructed of us to do so we have not changed this value. We can see from this plot that the heat generation reference plot is extremely similar to the plot with he Dirichlet BC's This is because we don't have much of a heat generation effect with a value of 1. However, our computed plot shows the exact same situation as our reference plot with an approximated error of 11.2703. This can most likely be attributed to the way the heat generation value was applied in our loop. We believe this error is acceptable

Comparison of Reference and Computed Fields with Neumann Boundary Condition (Heat Flux)

The following figures represents a computed mesh of the described scenario. For this section we use a heat flux BC value of 10000. This heat generation is solved for using the described method above.



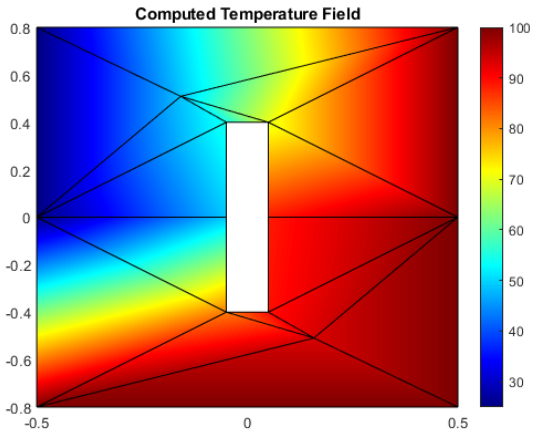
((a)) Computed Plot ((b)) Reference Plot

Figure 9. Comparison of Computed and Reference Fields

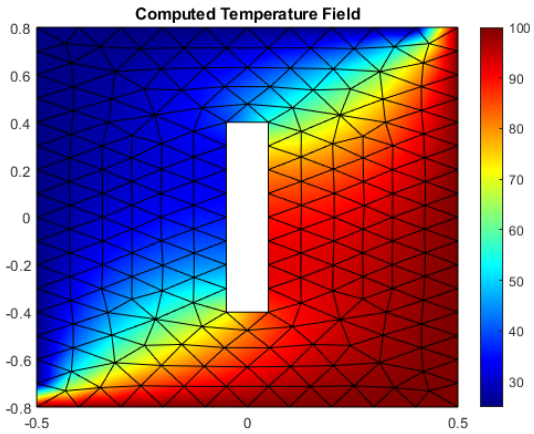
Our Neumann BC in this scenario is a heat flux of value 10000. A heat flux is the ability for an object to transfer heat efficiently from one area to another. We can see here that our computed field is similar to our reference field. the error, curiously, is the same as the previous section at 11.2703. This means that our heat flux was computed correctly. Our error still seems to come from the generated heat application. There is no extra error in this result

Comparison of Different Mesh Sizes

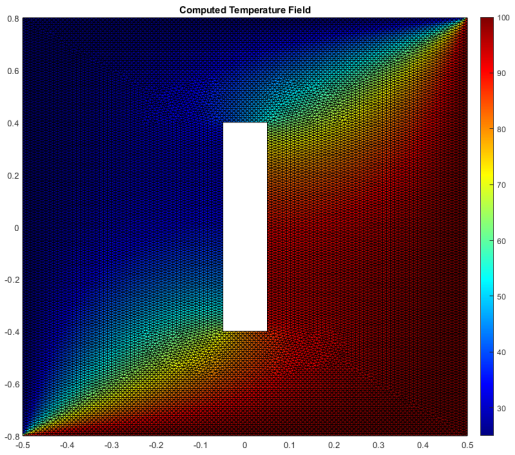
The following is a comparison of different mesh sizes. For this section, we have computed 3 plots that lower an order of magnitude as we move figures. The point of this section is to understand when a mesh size is fine enough for the purposes that we require Our largest mesh size for this



((a)) Mesh Density: 1.0



((b)) Mesh Density: 0.1



((c)) Mesh Density: 0.01

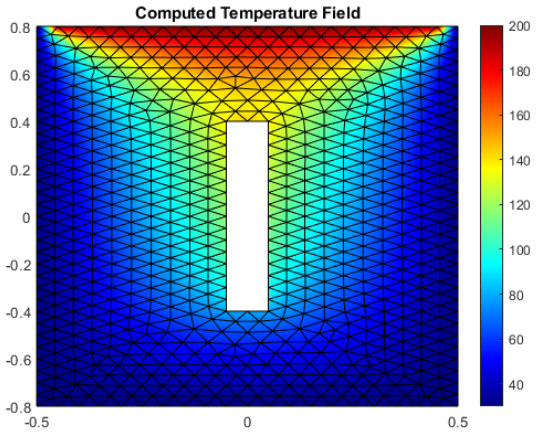
Figure 10. Comparison of Mesh Densities

section is a size of 1. This mesh density is interesting in that it describes am extremely simplified scenario. In fact, if compared to previous scenarios, it seems as if our initial temperature field has been modified. A cold side can be seen on the top left. and a general lack of detail is evident. This is not a satisfactory product. Moving on to figure b, we have a mesh density of 0.1. This mesh

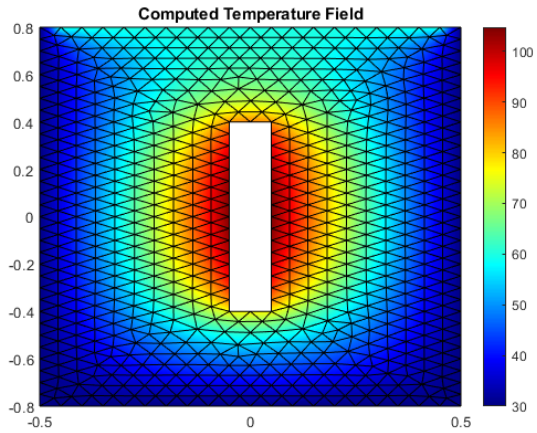
density shows an immediate different picture. We can now clearly see where our initial temperature field starts. We clearly see two cold edges and 2 hot edges. This is what we described. It is also important to note that this finer mesh meant a longer loading time when computing. A mesh of size 0.1 does not take too much time to compute. The same cannot be said for figure c. This figure has a mesh density of 0.01. This is an extremely fine mesh for this scenario. It can be seen that it follows the same look as figure b. However, it is definitely more detailed and smooth. While this may seem like a benefit, the computing time this took was far greater than the previous two meshes. This is because as you increase your mesh density, you exponentially increase solving time. In this scenario, a mesh density of 0.1 was sufficient to achieve the level of detail we need to analyze the result

Comparison of Different Boundary Conditions

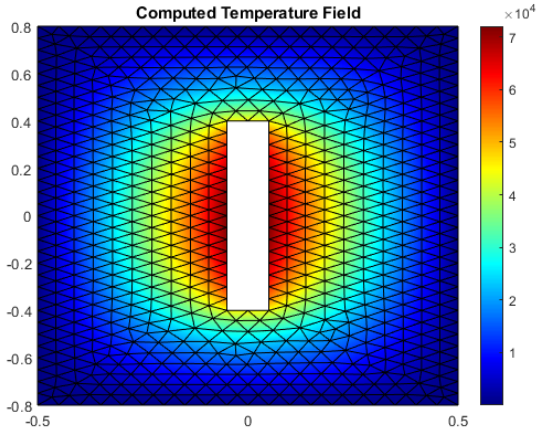
The following is a comparison of different mesh sizes. For this section, we have computed 3 plots that lower an order of magnitude as we move figures. The point of this section is to understand when a mesh size is fine enough for the purposes that we require



((a)) 200deg C on top 30deg C on other edges



((b)) 60deg C on top 30deg C on other edges



((c)) Original Temperature Field, Heat flux = 10000000

Figure 11. Comparison of Mesh Densities

The first figure, a, shows a situations where the top surface of the mesh is extremely hot, at 200deg C. We have also adjusted the other edges to have a value of 30deg C. We have kept the original heat flux of 10000. This result is incredibly interesting. We can see that the large heat flux allows the heat from the top edge boundary is getting sucked into the Neumann edges. This shows

the power of this method as we can describe large heat sinks and a large heat source.

Figure b has a similar set up as figure a. The difference is that the 200deg C value has been replaced with a 60deg C value. WE can see that the large heat flux still sucks most of the heat towards the Neumann edges. This is an example of tuning a system to get the scenario that we are after for

Figure c is simply a large increase in the heat flux to 10000000. This is an interesting result. We can see a large heat concentration around our Neumann condition. We can interpret this as all the energy in the system being sucked towards the heat sink in the middle. In fact, our temperature color distribution supports this. We can see that the outside temperature is incredibly cold. the center has more hat relative to the outside but most of the heat from the original temperature distribution as been sucked in my the large heat flux in the center

Conclusion

The finite element method is a useful tool every engineer should have an understanding of. It can help engineers solve complex problems quickly and efficiently given that the user knows what they are setting up for and what errors to look out for.

This project was an attempt to gather the key understandings needed in the Finite Element Analysis. These included understanding what meshing was and what elements and nodes to the idea of isopometry for normalizing elements onto a common space. We have learned the method and algorithm used to solve a finite element problem in parametric space. Furthermore, we have also shown that our method for solving finite element problems over a mesh is correct by comparing computed solutions to a known reference solution. This project was also an attempt to not only apply the basic understanding of meshes and the finite element method but to also experiment with different scenarios that are defined by the engineer using it. We experimented with different scenarios and explained what could possibly be happening in a given situation. The explanation for which became clear when you understand just how the underlying code for the meshes works. A higher heat flux would obviously suck heat towards it. We can see how this affects the mesh physically but also through the math.

Overall this project showed us the power if the finite element method and its applications in heat transfer. We learned methods and algorithms used to solve. In doing so, we also learned what to look out for when solving these problems to ensure that we are doing the best job that we can.